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 GDALAllRegister() which registers all format drivers built into GDAL/OGR.

In C++ :

```
#include "ogrsf_frmts.h"

int main()

{
    GDALAllRegister();
```

In C :

```
#include "gdal.h"

int main()

{
    GDALAllRegister();
```

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 (GDAL_OF_VECTOR) tells the **OGROpen()** (p. **??**) method that we want a vector driver to be use and that don't require update access. On failure NULL is returned, and we report an error.

In C++ :

```
GDALDataset       *poDS;

poDS = (GDALDataset*) GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
if( poDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

In C :

```
GDALDatasetH hDS;

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

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

In C++ :

```
OGRLayer  *poLayer;

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

In C :

```
OGRLayerH hLayer;

hLayer = GDALDatasetGetLayerByName( 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() == OFTInteger64 )
        printf( CPL_FRMT_GIB ",", poFeature->GetFieldAsInteger64( 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) == OFTInteger64 )
        printf( CPL_FRMT_GIB ",", OGR_F_GetFieldAsInteger64( 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,%3.f\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,%3.f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
}
else
{
    printf( "no point geometry\n" );
}
```

The **wkbFlatten()** (p. **??**) 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.

Starting with OGR 1.11, `several geometry fields` can be associated to a feature.

In C++ :

```
OGRGeometry *poGeometry;
int iGeomField;
int nGeomFieldCount;

nGeomFieldCount = poFeature->GetGeomFieldCount();
```

```
for(iGeomField = 0; iGeomField < nGeomFieldCount; iGeomField ++ )
{
    poGeometry = poFeature->GetGeomFieldRef(iGeomField);
    if( poGeometry != NULL
            && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
    {
        OGRPoint *poPoint = (OGRPoint *) poGeometry;

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

In C :

```
OGRGeometryH hGeometry;
int iGeomField;
int nGeomFieldCount;

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

In Python:

```
nGeomFieldCount = feat.GetGeomFieldCount()
for iGeomField in range(nGeomFieldCount):
    geom = feat.GetGeomFieldRef(iGeomField)
    if geom is not None and geom.GetGeometryType() == ogr.wkbPoint:
        print "%.3f, %.3f" % ( geom.GetX(), geom.GetY() )
    else:
        print "no point geometry\n"
```

Note that **OGRFeature::GetGeometryRef()** (p. **??**) and **OGRFeature::GetGeomFieldRef()** (p. **??**) return 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 GDALDataset::GetLayerByName() is also a reference to an internal layer owned by the GDALDataset 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 issus.

In C/C++ :

```
    GDALClose( poDS );
}
```

All together our program looks like this.

In C++ :

```
#include "ogrsf_frmts.h"

int main()

{
    GDALAllRegister();

    GDALDataset       *poDS;

    poDS = (GDALDataset*) GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
    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() == OFTInteger64 )
                printf( CPL_FRMT_GIB ",", poFeature->GetFieldAsInteger64( 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,%3.f\n", poPoint->getX(), poPoint->getY() );
        }
        else
        {
            printf( "no point geometry\n" );
        }
        OGRFeature::DestroyFeature( poFeature );
    }

    GDALClose( poDS );
}
```

In C :

```
#include "gdal.h"

int main()

{
    GDALAllRegister();

    GDALDatasetH hDS;
    OGRLayerH hLayer;
    OGRFeatureH hFeature;

    hDS = GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
    if( hDS == NULL )
    {
```

```
        printf( "Open failed.\n" );
        exit( 1 );
    }

    hLayer = GDALDatasetGetLayerByName( 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) == OFTInteger64 )
                printf( CPL_FRMT_GIB ",", OGR_F_GetFieldAsInteger64( 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,%3.f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
        }
        else
        {
            printf( "no point geometry\n" );
        }

        OGR_F_Destroy( hFeature );
    }

    GDALClose( hDS );
}
```

In Python:

```
import sys
from osgeo import gdal

ds = gdal.OpenEx( "point.shp", gdal.OF_VECTOR )
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 or field_defn.GetType() == ogr.OFTInteger64:
            print "%d" % feat.GetFieldAsInteger64(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 "ogrsf_frmts.h"

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

    GDALAllRegister();

    poDriver = GetGDALDriverManager()->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";
    GDALDriver *poDriver;

    GDALAllRegister();

    poDriver = (GDALDriver*) GDALGetDriverByName(pszDriverName );
    if( poDriver == 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, third, fourth and fifth argument are related to raster dimensions (in case the driver has raster capabilities). The last 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 ++ :

```
GDALDataset *poDS;

poDS = poDriver->Create( "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
if( poDS == NULL )
{
    printf( "Creation of output file failed.\n" );
    exit( 1 );
}
```

In C :

```
GDALDatasetH hDS;

hDS = GDALCreate( hDriver, "point_out.shp", 0, 0, 0, GDT_Unknown, 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 = GDALDatasetCreateLayer( 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 snipping 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::SetGeometryDirectly()** (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/C++ :

```
    GDALClose( poDS );
}
```

The same program all in one block looks like this:

In C++ :

```
#include "ogrsf_frmts.h"

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

    GDALAllRegister();

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

    GDALDataset *poDS;

    poDS = poDriver->Create( "point_out.shp", 0, 0, 0, GDT_Unknown, 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 );
    }

    GDALClose( poDS );
}
```

In C :

```
#include "gdal.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriverH hDriver;
    GDALDatasetH hDS;
    OGRLayerH hLayer;
    OGRFieldDefnH hFieldDefn;
```

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

    GDALAllRegister();

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

    hDS = GDALCreate( hDriver, "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
    if( hDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }

    hLayer = GDALDatasetCreateLayer( 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 );
   }

   GDALClose( hDS );
}
```

In Python :

```
import sys
from osgeo import gdal
from osgeo import ogr
import string

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

ds = drv.Create( "point_out.shp", 0, 0, 0, gdal.GDT_Unknown )
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
```

Starting with OGR 1.11, `several geometry fields` can be associated to a feature. This capability is just available for a few file formats, such as PostGIS.

To create such datasources, geometry fields must be first created. Spatial reference system objects can be associated to each geometry field.

In C++ :

```
OGRGeomFieldDefn oPointField( "PointField", wkbPoint );
OGRSpatialReference* poSRS = new OGRSpatialReference();
poSRS->importFromEPSG(4326);
oPointField.SetSpatialRef(poSRS);
poSRS->Release();

if( poLayer->CreateGeomField( &oPointField ) != OGRERR_NONE )
{
    printf( "Creating field PointField failed.\n" );
    exit( 1 );
}

OGRGeomFieldDefn oFieldPoint2( "PointField2", wkbPoint );
poSRS = new OGRSpatialReference();
poSRS->importFromEPSG(32631);
oPointField2.SetSpatialRef(poSRS);
poSRS->Release();

if( poLayer->CreateGeomField( &oPointField2 ) != OGRERR_NONE )
{
    printf( "Creating field PointField2 failed.\n" );
    exit( 1 );
}
```

In C :

```
OGRGeomFieldDefnH hPointField;
OGRGeomFieldDefnH hPointField2;
OGRSpatialReferenceH hSRS;

hPointField = OGR_GFld_Create( "PointField", wkbPoint );
hSRS = OSRNewSpatialReference( NULL );
OSRImportFromEPSG(hSRS, 4326);
OGR_GFld_SetSpatialRef(hPointField, hSRS);
OSRRelease(hSRS);
```

```
if( OGR_L_CreateGeomField( hLayer, hPointField ) != OGRERR_NONE )
{
    printf( "Creating field PointField failed.\n" );
    exit( 1 );
}

OGR_GFld_Destroy( hPointField );

hPointField2 = OGR_GFld_Create( "PointField2", wkbPoint );
OSRImportFromEPSG(hSRS, 32631);
OGR_GFld_SetSpatialRef(hPointField2, hSRS);
OSRRelease(hSRS);

if( OGR_L_CreateGeomField( hLayer, hPointField2 ) != OGRERR_NONE )
{
    printf( "Creating field PointField2 failed.\n" );
    exit( 1 );
}

OGR_GFld_Destroy( hPointField2 );
```

To write a feature to disk, we must create a local **OGRFeature** (p. **??**), set attributes and attach geometries 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;
OGRGeometry *poGeometry;
char* pszWKT;

poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );

pszWKT = (char*) "POINT (2 49)";
OGRGeometryFactory::createFromWkt( &pszWKT, NULL, &poGeometry );
poFeature->SetGeomFieldDirectly( "PointField", poGeometry );

pszWKT = (char*) "POINT (500000 4500000)";
OGRGeometryFactory::createFromWkt( &pszWKT, NULL, &poGeometry );
poFeature->SetGeomFieldDirectly( "PointField2", poGeometry );

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

OGRFeature::DestroyFeature( poFeature );
```

In C :

```
OGRFeatureH hFeature;
OGRGeometryH hGeometry;
char* pszWKT;

poFeature = OGR_F_Create( OGR_L_GetLayerDefn(hLayer) );

pszWKT = (char*) "POINT (2 49)";
OGR_G_CreateFromWkt( &pszWKT, NULL, &hGeometry );
OGR_F_SetGeomFieldDirectly( hFeature,
    OGR_F_GetGeomFieldIndex(hFeature, "PointField"), hGeometry );

pszWKT = (char*) "POINT (500000 4500000)";
OGR_G_CreateFromWkt( &pszWKT, NULL, &hGeometry );
OGR_F_SetGeomFieldDirectly( hFeature,
    OGR_F_GetGeomFieldIndex(hFeature, "PointField2"), hGeometry );

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

OGR_F_Destroy( hFeature );
```

In Python :

```
feat = ogr.Feature( lyr.GetLayerDefn() )
```

```
feat.SetGeomFieldDirectly( "PointField",
    ogr.CreateGeometryFromWkt( "POINT (2 49)" ) )
feat.SetGeomFieldDirectly( "PointField2",
    ogr.CreateGeometryFromWkt( "POINT (500000 4500000)" ) )

if lyr.CreateFeature( feat ) != 0 )
{
    print( "Failed to create feature.\n" );
    sys.exit( 1 );
}
```

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 SFCOM, 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** (`ogrsf_frmts.h`): **OGRLayer** (p. **??**) is an abstract base class represent a layer of features in an GDALDataset.
- **Dataset** (`gdal_priv.h`): A GDALDataset is an abstract base class representing a file or database containing one or more **OGRLayer** (p. **??**) objects.
- **Drivers** (`gdal_priv.h`): A GDALDriver represents a translator for a specific format, opening GDALDataset objects. All available drivers are managed by the GDALDriverManager.

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. **??**).

GDAL 2.0 extends those geometry type with non-linear geometries with the **OGRCircularString** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRMultiCurve** (p. **??**) and **OGRMultiSurface** (p. **??**) classes.

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.

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.

3.2.1 Compatibility issues with GDAL 2.0 non-linear geometries

Generic mechanisms have been introduced so that creating or modifying a feature with a non-linear geometry in a layer of a driver that does not support it will transform that geometry in the closest matching linear geometry.

On the other side, when retrieving data from the OGR C API, the **OGRSetNonLinearGeometriesEnabledFlag()** (p. **??**) function can be used, so that geometries and layer geometry type returned are also converted to their linear approximation if necessary.

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 as well supported in recent GDAL versions.

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 identifier. Starting with OGR 1.11, `several geometries` can be associated to a **OGRFeature** (p. **??**).

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 64-bit integer (GDAL 2.0 or later); however, this is not sufficiently expressive to model the natural feature ids in some formats. For instance, the GML feature id is a string.

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.

Starting with OGR 1.11, several geometry fields can be associated to a feature class. Each geometry field has its own indicator of geometry type allowed, returned by **OGRGeomFieldDefn::GetType()** (p. **??**), and its spatial reference system, returned by **OGRGeomFieldDefn::GetSpatialRef()** (p. **??**).

The **OGRFeatureDefn** (p. **??**) also contains 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 GDALDataset, and aren't instantiated or destroyed directly.

3.6 Dataset

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

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

A GDALDataset has a name (usually a filename) that can be used to reopen the data source with a GDALDriver.

The GDALDataset also has support for executing a datasource specific command, normally a form of SQL. This is accomplished via the GDALDataset::ExecuteSQL() 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

A GDALDriver object is instantiated for each file format supported. The GDALDriver objects are registered with the GDALDriverManager, a singleton class that is normally used to open new datasets.

It is intended that a new GDALDriver object is instanciated and define function pointers for operations like Identify(), Open() for each file format to be supported (along with a file format specific GDALDataset, and **OGRLayer** (p. **??**) classes).

On application startup registration functions are normally called for each desired file format. These functions instantiate the appropriate GDALDriver objects, and register them with the GDALDriverManager. When a dataset is to be opened, the driver manager will normally try each GDALDataset in turn, until one succeeds, returning a GDALDataset object.

Chapter 4

OGR Driver Implementation Tutorial

4.1 Overall Approach

In general new formats are added to OGR by implementing format specific drivers with intanciating a GDALDriver and subclasses of GDALDatasetand **OGRLayer** (p. **??**). The GDALDriver instance is registered with the GDAL↩ DriverManager 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 GDALDriver** (p. **??**)
2. **Basic Read Only Data Source** (p. **??**)
3. **Read Only Layer** (p. **??**)

4.3 Implementing GDALDriver

The format specific driver class is implemented as a instance of GDALDriver. One instance of the driver will normally be created, and registered with the GDALDriverManager. 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()

{
    GDALDriver  *poDriver;

    if( GDALGetDriverByName( "SPF" ) == NULL )
    {
        poDriver = new GDALDriver();

        poDriver->SetDescription( "SPF" );
        poDriver->SetMetadataItem( GDAL_DCAP_VECTOR, "YES" );
        poDriver->SetMetadataItem( GDAL_DMD_LONGNAME,
                                   "Long name for SPF driver" );
        poDriver->SetMetadataItem( GDAL_DMD_EXTENSION, "spf" );
        poDriver->SetMetadataItem( GDAL_DMD_HELPTOPIC,
                                   "drv_spf.html" );

        poDriver->pfnOpen = OGRSPFDriverOpen;
        poDriver->pfnIdentify = OGRSPFDriverIdentify;
        poDriver->pfnCreate = OGRSPFDriverCreate;

        poDriver->SetMetadataItem( GDAL_DCAP_VIRTUALIO, "YES" );
```

```
        GetGDALDriverManager()->RegisterDriver( poDriver );
    }
}
```

The SetDescription() sets 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.

SetMetadataItem(GDAL_DCAP_VECTOR, "YES") is specified to indicate that the driver will handle vector data.

SetMetadataItem(GDAL_DCAP_VIRTUALIO, "YES") is specified to indicate that the driver can deal with files opened with the VSI∗L GDAL API. Otherwise this metadata item should not be defined.

The driver declaration generally looks something like this for a format with read or read and update access (the Open() method) and creation support (the Create() method).

```
static GDALDataset* OGRSPFDriverOpen(GDALOpenInfo* poOpenInfo);
static int          OGRSPFDriverIdentify(GDALOpenInfo* poOpenInfo);
static GDALDataset* OGRSPFDriverCreate(const char* pszName, int nXSize, int nYSize,
                                       int nBands, GDALDataType eDT, char** papszOptions);
```

The Open() method is called by GDALOpenEx(). It 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 GDALDataset object for the dataset should be returned.

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

```
static GDALDataset *OGRSPFDriverOpen( GDALOpenInfo* poOpenInfo )
{
    if( !OGRSPFDriverIdentify(poOpenInfo) )
        return NULL;

    OGRSPFDataSource   *poDS = new OGRSPFDataSource();
    if( !poDS->Open( poOpenInfo->pszFilename, poOpenInfo->eAcces == GA_Update ) )
    {
        delete poDS;
        return NULL;
    }
    else
        return poDS;
}
```

The Identify() method is implemented as such :

```
static int OGRSPFDriverIdentify( GDALOpenInfo* poOpenInfo )
{
// ----------------------------------------------------------------------
//      Does this appear to be an .spf file?
// ----------------------------------------------------------------------
    return EQUAL( CPLGetExtension(poOpenInfo->pszFilename), "spf" );
}
```

Examples of the Create() method is 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 GDALDataset 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 Test↩ Capability() method is pure virtual, so it does need to be implemented.

```
class OGRSPFDataSource : public GDALDataset
{
    OGRSPFLayer       **papoLayers;
    int                 nLayers;

  public:
                        OGRSPFDataSource();
                        ~OGRSPFDataSource();

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

    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;
}

OGRSPFDataSource::~OGRSPFDataSource()

{
    for( int i = 0; i < nLayers; i++ )
        delete papoLayers[i];
    CPLFree( papoLayers );
}
```

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 preferrable.

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 )

{
    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 is 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 ) );
    SetDescription(poFeatureDefn->GetName());
    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 **OGRLayer** (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 GDALDataset supports executing commands against a datasource via the GDALDataset::ExecuteSQL() 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 **OGR↩ Layer::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 EA↩ S_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 alot 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 sumarization 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. Starting with GDAL 2.0, boolean resulting expressions (comparisons, logical operators) can also be used.

```
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 as 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'
```

Starting with OGR 2.0, the *hstore_get_value()* function can be used to extract a value associate to a key from a HSTORE string, formatted like 'key=>value,other_key=>other_value,...'

```
SELECT hstore_get_value('a => b, "key with space"=> "value with space"', 'key with space') FROM xxx --> '
      value with space'
```

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. boolean (GDAL >= 2.0)
2. character(field_length). By default, field_length=1.
3. float(field_length)
4. numeric(field_length, field_precision)
5. smallint(field_length) : 16 bit signed integer (GDAL >= 2.0)
6. integer(field_length)
7. bigint(field_length), 64 bit integer, extension to SQL92 (GDAL >= 2.0)
8. date(field_length)
9. time(field_length)
10. timestamp(field_length)
11. geometry, geometry(geometry_type), geometry(geometry_type,epsg_code)

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.

Starting with OGR 1.11, casting a WKT string to a geometry is allowed. geometry_type can be POINT[Z], LINES↩ TRING[Z], POLYGON[Z], MULTIPOINT[Z], MULTILINESTRING[Z], MULTIPOLYGON[Z], GEOMETRYCOLLEC↩ TION[Z] or GEOMETRY[Z].

5.1.1.4 String literals and identifiers quoting

Starting with GDAL 2.0 (see `RFC 52 - Strict OGR SQL quoting`), strict SQL92 rules are applied regarding string literals and identifiers quoting.

String literals (constants) must be surrounded with single-quote characters. e.g. WHERE a_field = 'a_value'

Identifiers (column names and tables names) can be used unquoted if they don't contain special characters or are not a SQL reserved keyword. Otherwise they must be surroundered with double-quote characters. e.g. WHERE "from" = 5

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 explanitory, but is is worth nothing 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 ES↩
CAPE 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
N0N 1P0            %N0N%         Yes
L4C 5E2            %N0N%         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 'N0N%')
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 asterix 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
```

Before GDAL 2.0, the expression after ON should necessarily be of the form "{primary_table}.{field_name} = {secondary_table}.{field_name}", and in that order. Starting with GDAL 2.0, it is possible to use a more complex boolean expression, involving multiple comparison operators, but with the restrictions mentionned in the below "JOIN limitations" section. In particular, in case of multiple joins (3 tables or more) the fields compared in a JOIN must belong to the primary table (the one after FROM) and the table of the active JOIN.

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 OLCAlterFieldDefn capability.
3. "ALTER TABLE tablename ALTER [COLUMN] columnname TYPE columntype" to change the type of an existing field. Supported if the layer declares the OLCAlterFieldDefn 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 GDALDataset, not against a specific layer. The call looks like this:

```
OGRLayer * GDALDataset::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 GDALDataset::ReleaseResultsSet() 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 GDALDataset::ExecuteSQL() 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 (>= 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 GDALDataset::ExecuteSQL() 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/UPDAT↩ E/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::GetGeometry↩ Ref()** (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. **??**).

6.1.6.1 Other functions

Starting with OGR 2.0, the *hstore_get_value()* function can be used to extract a value associate to a key from a HSTORE string, formatted like "key=>value,other_key=>other_value,..."

```
SELECT hstore_get_value('a => b, "key with space"=> "value with space"', 'key with space') --> 'value with
       space'
```

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 VirtualSpatialIndex 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 and 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[0,0,0,0,0,0,0],
            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'ed 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 ogrspatialreference.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 **OGR↩ CreateCoordinateTransformation()** (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 differents 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 assume that the point is on the geoide.

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

globalScope> Member OGR_Dr_CopyDataSource (p. ??) (OGRSFDriverH, OGRDataSourceH, const char *, char **) CPL_WARN_UNUSED_RESULT

Use GDALCreateCopy() in GDAL 2.0

globalScope> Member OGR_Dr_CreateDataSource (p. ??) (OGRSFDriverH, const char *, char **) CPL_↩ WARN_UNUSED_RESULT

Use GDALCreate() in GDAL 2.0

globalScope> Member OGR_Dr_DeleteDataSource (p. ??) (OGRSFDriverH, const char *)

Use GDALDeleteDataset() in GDAL 2

globalScope> Member OGR_Dr_Open (p. ??) (OGRSFDriverH, const char *, int) CPL_WARN_UNUSED_R↩ ESULT

Use GDALOpenEx() in GDAL 2.0

globalScope> Member OGR_Dr_TestCapability (p. ??) (OGRSFDriverH, const char *)

Use GDALGetMetadataItem(hDriver, GDAL_DCAP_CREATE) in GDAL 2.0

globalScope> Member OGR_DS_CopyLayer (p. ??) (OGRDataSourceH, OGRLayerH, const char *, char **)

Use GDALDatasetCopyLayer() in GDAL 2.0

globalScope> Member OGR_DS_CreateLayer (p. ??) (OGRDataSourceH, const char *, OGRSpatial↩ ReferenceH, OGRwkbGeometryType, char **)

Use GDALDatasetCreateLayer() in GDAL 2.0

globalScope> Member OGR_DS_DeleteLayer (p. ??) (OGRDataSourceH, int)

Use GDALDatasetDeleteLayer() in GDAL 2.0

globalScope> Member OGR_DS_Destroy (p. ??) (OGRDataSourceH)

Use GDALClose() in GDAL 2.0

globalScope> Member OGR_DS_ExecuteSQL (p. ??) (OGRDataSourceH, const char *, OGRGeometryH, const char *)

Use GDALDatasetExecuteSQL() in GDAL 2.0

globalScope> Member OGR_DS_GetDriver (p. ??) (OGRDataSourceH)

Use GDALGetDatasetDriver() in GDAL 2.0

globalScope> Member OGR_DS_GetLayer (p. ??) (OGRDataSourceH, int)

Use GDALDatasetGetLayer() in GDAL 2.0

globalScope> Member OGR_DS_GetLayerByName (p. ??) (OGRDataSourceH, const char *)

Use GDALDatasetGetLayerByName() in GDAL 2.0

globalScope> Member OGR_DS_GetLayerCount (p. ??) (OGRDataSourceH)

Use GDALDatasetGetLayerCount() in GDAL 2.0

globalScope> Member OGR_DS_GetName (p. **??**) **(OGRDataSourceH)**

Use GDALGetDescription() in GDAL 2.0

globalScope> Member OGR_DS_ReleaseResultSet (p. **??**) **(OGRDataSourceH, OGRLayerH)**

Use GDALDatasetReleaseResultSet() in GDAL 2.0

globalScope> Member OGR_DS_TestCapability (p. **??**) **(OGRDataSourceH, const char *)**

Use GDALDatasetTestCapability() in GDAL 2.0

See also

globalScope> Member OGR_G_GetArea (p. **??**) **(OGRGeometryH) CPL_WARN_DEPRECATED("Non standard method. Use OGR_G_Area() instead")**

OGR_G_Area() (p. **??**)

globalScope> Member OGR_G_GetBoundary (p. **??**) **(OGRGeometryH) CPL_WARN_DEPRECATED("Non standard method. Use OGR_G_Boundary() instead")**

globalScope> Member OGR_G_SymmetricDifference (p. **??**) **(OGRGeometryH, OGRGeometryH) CPL_WARN_DEPRECATED("Non standard method. Use OGR_G_SymDifference() instead")**

Class OGRDataSource (p. **??**)

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")**

globalScope> Member OGRGetDriver (p. **??**) **(int)**

Use GDALGetDriver() in GDAL 2.0

globalScope> Member OGRGetDriverByName (p. **??**) **(const char *)**

Use GDALGetDriverByName() in GDAL 2.0

globalScope> Member OGRGetDriverCount (p. **??**) **(void)**

Use GDALGetDriverCount() in GDAL 2.0

globalScope> Member OGROpen (p. **??**) **(const char *, int, OGRSFDriverH *) CPL_WARN_UNUSED_RESULT**

Use GDALOpenEx() in GDAL 2.0

globalScope> Member OGRRegisterAll (p. **??**) **()**

Use GDALAllRegister() in GDAL 2.0

globalScope> Member OGRRegisterAll (p. **??**) **()**

Use GDALAllRegister() in GDAL 2.0

globalScope> Member OGRReleaseDataSource (p. **??**) **(OGRDataSourceH)**

Use GDALClose() in GDAL 2.0

Class OGRSFDriver (p. **??**)

Class OGRSFDriverRegistrar (p. **??**)

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

globalScope> Member wkb25DBit (p. **??**)

in GDAL 2.0. Use **wkbHasZ()** (p. **??**) or **wkbSetZ()** (p. **??**) instead

Chapter 9

Hierarchical Index

9.1 Class Hierarchy

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

Chapter 10

Class Index

10.1 Class List

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

Chapter 11

File Index

11.1 File List

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

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(), CPL↩ ListGetData(), 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(), CPL↩ ListCount(), CPLListDestroy(), CPLListGet(), CPLListGetLast(), CPLListGetNext(), CPLListInsert(), and CPLList↩ Remove().

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

- **cpl_list.h**

12.3 _CPLLock Struct Reference

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

- cpl_multiproc.cpp

12.4 _CPLQuadTree Struct Reference

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

- cpl_quad_tree.cpp

12.5 _CPLSpawnedProcess Struct Reference

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

- cpl_spawn.cpp

12.6 _CPLSpinLock Struct Reference

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

- cpl_multiproc.cpp

12.7 _MutexLinkedElt Struct Reference

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

- cpl_multiproc.cpp

12.8 _OGRGeocodingSessionHS Struct Reference

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

- ogr_geocoding.cpp

12.9 _QuadTreeNode Struct Reference

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

- cpl_quad_tree.cpp

12.10 _sPolyExtended Struct Reference

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

- ogrgeometryfactory.cpp

12.11 CachedConnection Struct Reference

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

- cpl_vsil_curl.cpp

12.12 CachedDirList Struct Reference

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

- cpl_vsil_curl.cpp

12.13 CachedFileProp Struct Reference

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

- cpl_vsil_curl.cpp
- cpl_vsil_curl_streaming.cpp

12.14 CachedRegion Struct Reference

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

- cpl_vsil_curl.cpp

12.15 CPLErrorContext Struct Reference

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

- cpl_error.cpp

12.16 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**

12.16.1 Detailed Description

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

12.16.2 Member Data Documentation

12.16.2.1 int CPLHTTPResult::nDataLen

Length of the pabyData buffer

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

12.16.2.2 int CPLHTTPResult::nMimePartCount

Number of parts in a multipart message

Referenced by CPLHTTPDestroyResult(), and CPLHTTPParseMultipartMime().

12.16.2.3 int CPLHTTPResult::nStatus

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

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

12.16.2.4 GByte* CPLHTTPResult::pabyData

Buffer with downloaded data

Referenced by CPLHTTPDestroyResult(), CPLHTTPFetch(), CPLHTTPParseMultipartMime(), GOA2GetAccess↩Token(), GOA2GetRefreshToken(), and OGRSpatialReference::importFromUrl().

12.16.2.5 char** CPLHTTPResult::papszHeaders

Headers returned

Referenced by CPLHTTPDestroyResult(), and CPLHTTPFetch().

12.16.2.6 CPLMimePart* CPLHTTPResult::pasMimePart

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

Referenced by CPLHTTPDestroyResult(), and CPLHTTPParseMultipartMime().

12.16.2.7 char* CPLHTTPResult::pszContentType

Content-Type of the response

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

12.16.2.8 char* CPLHTTPResult::pszErrBuf

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**

12.17 CPLKeywordParser Class Reference

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

- cplkeywordparser.h
- cplkeywordparser.cpp

12.18 CPLLocaleC Class Reference

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

- **cpl_conv.h**
- cpl_conv.cpp

12.19 CPLLockHolder Class Reference

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

- cpl_multiproc.h
- cpl_multiproc.cpp

12.20 CPLMimePart Struct Reference

`#include <cpl_http.h>`

Public Attributes

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

12.20.1 Detailed Description

Describe a part of a multipart message

12.20.2 Member Data Documentation

12.20.2.1 int CPLMimePart::nDataLen

Buffer length

Referenced by CPLHTTPParseMultipartMime().

12.20.2.2 GByte* CPLMimePart::pabyData

Buffer with data of the part

Referenced by CPLHTTPParseMultipartMime().

12.20.2.3 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**

12.21 CPLMutexHolder Class Reference

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

- cpl_multiproc.h
- cpl_multiproc.cpp

12.22 CPLODBCDriverInstaller 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)

12.22.1 Detailed Description

A class providing functions to install or remove ODBC driver.

12.22.2 Member Function Documentation

12.22.2.1 int CPLODBCDriverInstaller::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

pszDriver	- The driver definition as a list of keyword-value pairs describing the driver (See ODBC API Reference).
pszPathIn	- Full path of the target directory of the installation, or a null pointer (for unixODBC, NULL is passed).
fRequest	- The fRequest argument must contain one of the following values: ODBC_INSTALL_C↩ OMPLETE - (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().

12.22.2.2 int CPLODBCDriverInstaller::RemoveDriver (const char * *pszDriverName,* int *fRemoveDSN* = 0)

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

Parameters

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

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

12.23 CPLODBCSession Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

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

12.23.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

12.23.2 Member Function Documentation

12.23.2.1 int CPLODBCSession::EstablishSession (const char * *pszDSN,* const char * *pszUserid,* const char * *pszPassword*)

Connect to database and logon.

Parameters

pszDSN	The name of the DSN being used to connect. This is not optional.
pszUserid	the userid to logon as, may be NULL if not not required, or provided by the DSN.
pszPassword	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().

12.23.2.2 const char * CPLODBCSession::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

12.24 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)
- const char * **GetColColumnDef** (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)

12.24.1 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.

12.24.2 Member Function Documentation

12.24.2.1 void CPLODBCStatement::Append (const char * *pszText*)

Append text to internal command.

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

Parameters

pszText	text to append.

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

12.24.2.2 void CPLODBCStatement::Append (int *nValue*)

Append to internal command.

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

Parameters

nValue	value to append to the command.

References Append().

12.24.2.3 void CPLODBCStatement::Append (double *dfValue*)

Append to internal command.

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

Parameters

dfValue	value to append to the command.

References Append().

12.24.2.4 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

pszText	text to append.

References Append().

12.24.2.5 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

pszFormat	printf() style format string.

Returns

FALSE if formatting fails dueto result being too large.

References Append().

12.24.2.6 void CPLODBCStatement::Clear ( )

Clear internal command text and result set definitions.

References CSLDestroy().

Referenced by ExecuteSQL().

12.24.2.7 void CPLODBCStatement::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

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

References Fetch(), GetColCount(), GetColData(), GetColName(), GetColNullable(), GetColPrecision(), GetCol↩Size(), GetColType(), and GetTypeName().

12.24.2.8 int CPLODBCStatement::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

pszStatement	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().

12.24.2.9 int CPLODBCStatement::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 occured if **Fetch()** (p. **??**) fails.

Parameters

nOrientation	One of SQL_FETCH_NEXT, SQL_FETCH_LAST, SQL_FETCH_PRIOR, SQL_FETCH_ABSOLUTE, or SQL_FETCH_RELATIVE (default is SQL_FETCH_NEXT).
nOffset	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().

12.24.2.10 const char * CPLODBCStatement::GetColColumnDef (int *iCol*)

Fetch a column default value.

Returns the default value of a column.

Parameters

iCol	the zero based column index.

Returns

NULL if the default value is not dpecified or the internal copy of the default value.

12.24.2.11 int CPLODBCStatement::GetColCount ( )

Fetch the resultset column count.

Returns

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

Referenced by DumpResult().

12.24.2.12 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

iCol	the zero based column to fetch.
pszDefault	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().

12.24.2.13 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

pszColName	the name of the column requested.
pszDefault	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().

12.24.2.14 int CPLODBCStatement::GetColId (const char * *pszColName*)

Fetch column index.

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

Parameters

pszColName	the name to search for.

Returns

the column index, or -1 if not found.

Referenced by GetColData().

12.24.2.15 const char * CPLODBCStatement::GetColName (int *iCol*)

Fetch a column name.

Parameters

iCol	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().

12.24.2.16 short CPLODBCStatement::GetColNullable (int *iCol*)

Fetch the column nullability.

Parameters

iCol	the zero based column index.

Returns

TRUE if the column may contains or FALSE otherwise.

Referenced by DumpResult().

12.24.2.17 short CPLODBCStatement::GetColPrecision (int *iCol*)

Fetch the column precision.

Parameters

iCol	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().

12.24.2.18 short CPLODBCStatement::GetColSize (int *iCol*)

Fetch the column width.

Parameters

iCol	the zero based column index.

Returns

column width, zero for unknown width columns.

Referenced by DumpResult().

12.24.2.19 short CPLODBCStatement::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_TIMESTAMPT.

Parameters

iCol	the zero based column index.

Returns

type code or -1 if the column is illegal.

Referenced by DumpResult().

12.24.2.20 const char * CPLODBCStatement::GetColTypeName (int *iCol*)

Fetch a column data type name.

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

Parameters

iCol	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.

12.24.2.21 int CPLODBCStatement::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 **CPLODBCStatement** (p. **??**) column definitions essentially as if a "SELECT * FROM tablename" had been done; however, no resultset will be available.

Parameters

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

12.24.2.22 int CPLODBCStatement::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

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

Returns

TRUE on success or FALSE on failure.

12.24.2.23 int CPLODBCStatement::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

pszCatalog	the catalog to find the table in, use NULL (the default) if no catalog is available.
pszSchema	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().

12.24.2.24 SQLSMALLINT CPLODBCStatement::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 **CPLODBC↩ Statement::GetColType()** (p. **??**)).

Parameters

nTypeCode	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().

12.24.2.25 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

nTypeCode	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

12.25 CPLRectObj Struct Reference

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

- **cpl_quad_tree.h**

12.26 CPLSharedFileInfo Struct Reference

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

- **cpl_conv.h**

12.27 CPLSharedFileInfoExtra Struct Reference

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

- cpl_conv.cpp

12.28 CPLStdCallThreadInfo Struct Reference

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

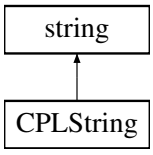
- cpl_multiproc.cpp

12.29 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)

12.29.1 Detailed Description

Convenient string class based on std::string.

12.29.2 Member Function Documentation

12.29.2.1 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 seperator) and appended to the target **CPLString** (p. **??**).

Parameters

dfValue	the value to format.
pszFormat	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. **??**).

References CPLsprintf().

12.29.2.2 size_t CPLString::ifind (const std::string & *str*, size_t *pos* = 0) const

Case insensitive find() alternative.

Parameters

str	substring to find.
pos	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().

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

Case insensitive find() alternative.

Parameters

s	substring to find.
nPos	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().

12.29.2.4 CPLString & CPLString::tolower (void)

Convert to lower case in place.

Referenced by ifind().

12.29.2.5 CPLString & CPLString::toupper (void)

Convert to upper case in place.

12.29.2.6 CPLString & CPLString::Trim ( )

Trim white space.

Trims white space off the let 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

12.30 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 ∗∗papszListIn, int bTakeOwnership=1)
- char ∗ **operator[ ]** (int i)
- char ∗∗ **StealList** ()
- **CPLStringList** & **Sort** ()

12.30.1 Detailed Description

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

12.30.2 Constructor & Destructor Documentation

12.30.2.1 CPLStringList::CPLStringList (char ∗∗ *papszListIn,* int *bTakeOwnership* = `1`)

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

Parameters

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

References Assign().

12.30.3 Member Function Documentation

12.30.3.1 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

pszKey	the key name to add.
pszValue	the key value to add.

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

Referenced by SetNameValue().

12.30.3.2 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

pszNewString	the string to add to the list.

References AddStringDirectly(), and CPLStrdup().

Referenced by CSLTokenizeString2(), GOA2GetAccessToken(), GOA2GetRefreshToken(), and VSIReadDir↩Recursive().

12.30.3.3 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

pszNewString	the string to add to the list.

References Count().

Referenced by AddNameValue(), and AddString().

12.30.3.4 CPLStringList & CPLStringList::Assign (char ** *papszListIn,* int *bTakeOwnership* = `1`)

Assign a list of strings.

Parameters

papszListIn	the NULL terminated list of strings to consume.
bTakeOwnership	TRUE if the **CPLStringList** (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().

12.30.3.5 CPLStringList & CPLStringList::Clear ( )

Clear the string list.

References CSLDestroy().

Referenced by Assign().

12.30.3.6 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().

12.30.3.7 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

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

Returns

TRUE or FALSE

References CSLTestBoolean(), and FetchNameValue().

12.30.3.8 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

pszName	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().

12.30.3.9 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

pszName	the key name to search for.
pszDefault	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().

12.30.3.10 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

pszKey	the name to search for.

Returns

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

References CSLFindName().

Referenced by FetchNameValue(), and SetNameValue().

12.30.3.11 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

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

References CPLStrdup().

12.30.3.12 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

nInsertAtLineNo	the line to insert at, zero to insert at front.
pszNewLine	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().

12.30.3.13 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	the index of the list item to return.

Returns

selected entry in the list.

References Count().

12.30.3.14 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

pszKey	the key name to add.
pszValue	the key value to add.

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

12.30.3.15 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 **Find↩ Name()** (p. **??**), **FetchNameValue()** (p. **??**) and **FetchNameValueDef()** (p. **??**) will do a binary search to find the key, substantially improve lookup performance in large lists.

References Count().

12.30.3.16 char ** CPLStringList::StealList ( )

Seize ownership of underlying string array.

This method is simmilar 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 VSIReadDirRecursive().

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

- **cpl_string.h**
- cplstringlist.cpp

12.31 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.

12.31.1 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.

12.31.2 Member Data Documentation

12.31.2.1 CPLXMLNodeType CPLXMLNode::eType

Node type.

One of CXT_Element, CXT_Text, CXT_Attribute, CXT_Comment, or CXT_Literal.

Referenced by CPLAddXMLChild(), CPLCloneXMLTree(), CPLCreateXMLNode(), CPLGetXMLNode(), CPLGet↩XMLValue(), CPLSearchXMLNode(), CPLSetXMLValue(), and CPLStripXMLNamespace().

12.31.2.2 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(), CP↩LGetXMLNode(), CPLGetXMLValue(), CPLRemoveXMLChild(), CPLSearchXMLNode(), CPLSetXMLValue(), and CPLStripXMLNamespace().

12.31.2.3 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(), CPL↩DestroyXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLRemoveXMLChild(), CPLSearchXMLNode(), CPLSerializeXMLTree(), CPLSetXMLValue(), CPLStripXMLNamespace(), and OGRSpatialReference::import↩FromXML().

12.31.2.4 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(), CPL↩ GetXMLValue(), CPLParseXMLString(), CPLSearchXMLNode(), CPLSetXMLValue(), CPLStripXMLNamespace(), and OGRSpatialReference::importFromXML().

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

- **cpl_minixml.h**

12.32 CPLZip Struct Reference

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

- cpl_minizip_zip.cpp

12.33 ctb Struct Reference

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

- cpl_csv.cpp

12.34 curfile_info Struct Reference

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

- cpl_minizip_zip.cpp

12.35 DefaultCSVFileNameTLS Struct Reference

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

- cpl_csv.cpp

12.36 errHandler Struct Reference

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

- cpl_error.cpp

12.37 file_in_zip_read_info_s Struct Reference

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

- cpl_minizip_unzip.cpp

12.38 FindFileTLS Struct Reference

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

- cpl_findfile.cpp

12.39 GDALScaledProgressInfo Struct Reference

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

- cpl_progress.cpp

12.40 GZipSnapshot Struct Reference

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

- cpl_vsil_gzip.cpp

12.41 IOGRTransactionBehaviour Class Reference

`#include <ogremulatedtransaction.h>`

Public Member Functions

- virtual OGRErr **StartTransaction** (**OGRDataSource** *&poDSInOut, int &bOutHasReopenedDS)=0
- virtual OGRErr **CommitTransaction** (**OGRDataSource** *&poDSInOut, int &bOutHasReopenedDS)=0
- virtual OGRErr **RollbackTransaction** (**OGRDataSource** *&poDSInOut, int &bOutHasReopenedDS)=0

12.41.1 Detailed Description

IOGRTransactionBehaviour (p. **??**) is an interface that a driver must implement to provide emulation of transactions.

Since

GDAL 2.0

12.41.2 Member Function Documentation

12.41.2.1 virtual OGRErr IOGRTransactionBehaviour::CommitTransaction (OGRDataSource *& *poDSInOut,* int & *bOutHasReopenedDS*) [pure virtual]

Commit a transaction.

The implementation may update the poDSInOut reference by closing and reopening the datasource (or assigning it to NULL in case of error). In which case bOutHasReopenedDS must be set to TRUE.

The implementation can for example remove the backup it may have done at **StartTransaction()** (p. **??**) time.

Parameters

poDSInOut	datasource handle that may be modified
bOutHas↩ ReopenedDS	output boolean to indicate if datasource has been closed

Returns

OGRERR_NONE in case of success

12.41.2.2 virtual OGRErr IOGRTransactionBehaviour::RollbackTransaction (OGRDataSource *& *poDSInOut*, int & *bOutHasReopenedDS*) [pure virtual]

Rollback a transaction.

The implementation may update the poDSInOut reference by closing and reopening the datasource (or assigning it to NULL in case of error). In which case bOutHasReopenedDS must be set to TRUE.

The implementation can for example restore the backup it may have done at **StartTransaction()** (p. **??**) time.

Parameters

poDSInOut	datasource handle that may be modified
bOutHas↩ ReopenedDS	output boolean to indicate if datasource has been closed

Returns

OGRERR_NONE in case of success

12.41.2.3 virtual OGRErr IOGRTransactionBehaviour::StartTransaction (OGRDataSource *& *poDSInOut*, int & *bOutHasReopenedDS*) [pure virtual]

Start a transaction.

The implementation may update the poDSInOut reference by closing and reopening the datasource (or assigning it to NULL in case of error). In which case bOutHasReopenedDS must be set to TRUE.

The implementation can for example backup the existing files/directories that compose the current datasource.

Parameters

poDSInOut	datasource handle that may be modified
bOutHas↩ ReopenedDS	output boolean to indicate if datasource has been closed

Returns

OGRERR_NONE in case of success

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

- ogremulatedtransaction.h
- ogremulatedtransaction.cpp

12.42 linkedlist_data_s Struct Reference

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

- cpl_minizip_zip.cpp

12.43 linkedlist_datablock_internal_s Struct Reference

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

- cpl_minizip_zip.cpp

12.44 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 n↩ StepSize=1, int bChildOfHit=FALSE)

12.44.1 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 accessable for detailed access to components of an SRS definition.

12.44.2 Constructor & Destructor Documentation

12.44.2.1 OGR_SRSNode::OGR_SRSNode (const char * *pszValueIn* = NULL)

Constructor.

Parameters

pszValueIn	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().

12.44.3 Member Function Documentation

12.44.3.1 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

poNew	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().

12.44.3.2 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

pszNode	Restrict remapping to children of this type of node (eg. "PROJECTION")
papszSrcValues	a NULL terminated array of source string. If the node value matches one of these (case insensitive) then replacement occurs.
papszDstValues	an array of destination strings. On a match, the one corresponding to a source value will be used to replace a node.
nStepSize	increment when stepping through source and destination arrays, allowing source and destination arrays to be one interleaved array for instances. Defaults to 1.
bChildOfHit	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 occured.

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

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

12.44.3.3 OGR_SRSNode * OGR_SRSNode::Clone ( ) const

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

Returns

a new node tree, which becomes the responsiblity 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::SetGeocCS(), and OGRSpatialReference::StripVertical().

12.44.3.4 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

iChild	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().

12.44.3.5 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

ppszResult	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().

12.44.3.6 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

pszValue	the node value being searched for.

Returns

the child index, or -1 on failure.

Referenced by OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::Fixup(), OGRSpatial↩ Reference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::morph↩ FromESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatial↩ Reference::SetAxes(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetStatePlane(), OGR↩ SpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and StripNodes().

12.44.3.7 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 Get↩ Value().

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

12.44.3.8 OGR_SRSNode * OGR_SRSNode::GetChild (int *iChild*)

Fetch requested child.

Parameters

iChild	the index of the child to fetch, from 0 to **GetChildCount()** (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::EPS↩ GTreatsAsNorthingEasting(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToProj4(), O↩ GRSpatialReference::FindProjParm(), FixupOrdering(), OGRSpatialReference::GetAngularUnits(), OGRSpatial↩ Reference::GetAttrValue(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthority↩ Name(), OGRSpatialReference::GetAxis(), OGRSpatialReference::GetExtension(), OGRSpatialReference::↩ GetInvFlattening(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OG↩ RSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTargetLinearUnits(), OGRSpatialReference↩ ::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), O↩ GRSpatialReference::IsSame(), MakeValueSafe(), OGRSpatialReference::morphFromESRI(), OGRSpatial↩ Reference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), O↩ GRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTargetLinear↩ Units(), StripNodes(), and OGRSpatialReference::StripVertical().

12.44.3.9 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().

12.44.3.10 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

pszName	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(), and OGRSpatialReference::SetGeocCS().

12.44.3.11 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

::importFromURN(), OGRSpatialReference::IsCompound(), OGRSpatialReference::IsGeocentric(), OGRSpatial↩ Reference::IsGeographic(), OGRSpatialReference::IsProjected(), OGRSpatialReference::IsSame(), OGRSpatial↩ Reference::IsVertical(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), O↩ GRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::Set↩ GeocCS(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), OGRSpatialReference::SetProj↩ Parm(), OGRSpatialReference::SetVertCS(), and OGRSpatialReference::StripCTParms().

12.44.3.12 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

ppszInput	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.

12.44.3.13 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

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

References CPLRealloc().

Referenced by AddChild(), OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::morphFromESR↩ I(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetProj↩ CS(), OGRSpatialReference::SetProjection(), and OGRSpatialReference::SetTOWGS84().

12.44.3.14 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().

12.44.3.15 void OGR_SRSNode::SetValue (const char * *pszNewValue*)

Set the node value.

Parameters

pszNewValue	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().

12.44.3.16 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

pszName	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

12.45 ogr_style_param Struct Reference

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

- **ogr_featurestyle.h**

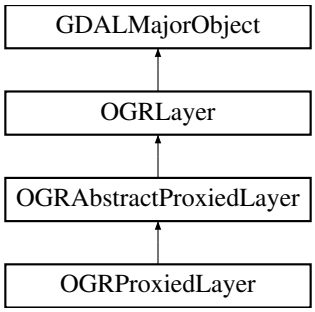
12.46 ogr_style_value Struct Reference

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

- **ogr_featurestyle.h**

12.47 OGRAbstractProxiedLayer Class Reference

Inheritance diagram for OGRAbstractProxiedLayer:



Friends

- class **OGRLayerPool**

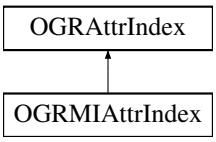
Additional Inherited Members

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

- ogrlayerpool.h
- ogrlayerpool.cpp

12.48 OGRAttrIndex Class Reference

Inheritance diagram for OGRAttrIndex:



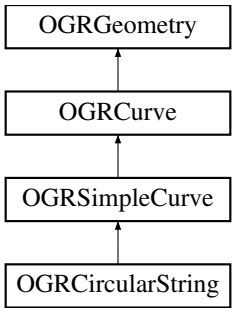
The documentation for this class was generated from the following files:

- ogr_attrind.h
- ogr_attrind.cpp

12.49 OGRCircularString Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRCircularString:



Public Member Functions

- **OGRCircularString** ()

  Create an empty circular string.
- virtual OGRErr **importFromWkb** (unsigned char ∗, int=-1, **OGRwkbVariant**=**wkbVariantOldOgc**)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char ∗, **OGRwkbVariant**=**wkbVariantOldOgc**) 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, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- virtual OGRBoolean **IsValid** () const

  Test if the geometry is valid.
- 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 double **get_Length** () const

  Returns the length of the curve.
- virtual **OGRLineString** ∗ **CurveToLine** (double dfMaxAngleStepSizeDegrees=0, const char ∗const ∗papsz↩ Options=NULL) const

  Return a linestring from a curve geometry.
- virtual void **Value** (double, **OGRPoint** ∗) const

  Fetch point at given distance along curve.
- virtual double **get_Area** () const

  Get the area of the (closed) curve.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual const char ∗ **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual void **segmentize** (double dfMaxLength)

  Modify the geometry such it has no segment longer then the given distance.
- virtual OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.
- virtual **OGRGeometry** ∗ **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char ∗const ∗papszOptions=NULL) const

  Return, possibly approximate, non-curve version of this geometry.

Protected Member Functions

- virtual int **ContainsPoint** (const **OGRPoint** ∗p) const

  Returns if a point is contained in a (closed) curve.
- virtual double **get_AreaOfCurveSegments** () const

  Get the area of the purely curve portions of a (closed) curve.

Additional Inherited Members

12.49.1 Detailed Description

Concrete representation of a circular string, that is to say a curve made of one or several arc circles.

Note: for implementation convenience, we make it inherit from **OGRSimpleCurve** (p. **??**) whereas SQL/MM only makes it inherits from **OGRCurve** (p. **??**).

Compatibility: ISO SQL/MM Part 3.

Since

GDAL 2.0

12.49.2 Member Function Documentation

12.49.2.1 int OGRCircularString::ContainsPoint (const OGRPoint ∗ *p*) const [protected], [virtual]

Returns if a point is contained in a (closed) curve.

Final users should use **OGRGeometry::Contains()** (p. **??**) instead.

Parameters

p	the point to test

Returns

TRUE if it is inside the curve, FALSE otherwise or -1 if unknown.

Since

GDAL 2.0

Reimplemented from **OGRCurve** (p. **??**).

References OGRPoint::getX(), and OGRPoint::getY().

12.49.2.2 OGRLineString ∗ OGRCircularString::CurveToLine (double *dfMaxAngleStepSizeDegrees* = 0, const char ∗const ∗ *papszOptions* = NULL) const [virtual]

Return a linestring from a curve geometry.

The returned geometry is a new instance whose ownership belongs to the caller.

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.

This method relates to the ISO SQL/MM Part 3 ICurve::CurveToLine() method.

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

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings or NULL. See **OGRGeometryFactory::curveTo↩ LineString()** (p. **??**) for valid options.

Returns

a line string approximating the curve

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

References OGRSimpleCurve::addSubLineString(), OGRGeometry::assignSpatialReference(), OGRGeometry↩ Factory::curveToLineString(), OGRGeometry::getCoordinateDimension(), and OGRGeometry::getSpatial↩ Reference().

Referenced by get_Area(), and getLinearGeometry().

12.49.2.3 OGRErr OGRCircularString::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGR↩ Geometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRSimpleCurve** (p. **??**).

References OGRSimpleCurve::exportToWkb(), wkbVariantIso, and wkbVariantOldOgc.

12.49.2.4 OGRErr OGRCircularString::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, ∗ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRSimpleCurve** (p. **??**).

References OGRSimpleCurve::exportToWkt(), and wkbVariantIso.

12.49.2.5 double OGRCircularString::get_Area ( ) const [virtual]

Get the area of the (closed) curve.

This method is designed to be used by **OGRCurvePolygon::get_Area()** (p. **??**).

Returns

the area of the feature in square units of the spatial reference system in use.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

References CurveToLine(), OGRLineString::get_Area(), get_AreaOfCurveSegments(), OGRCurve::get_IsClosed(), OGRSimpleCurve::get_LinearArea(), OGRCurve::IsConvex(), and OGRSimpleCurve::IsEmpty().

12.49.2.6 double OGRCircularString::get_AreaOfCurveSegments ( ) const [protected], [virtual]

Get the area of the purely curve portions of a (closed) curve.

This method is designed to be used on a closed convex curve.

Returns

the area of the feature in square units of the spatial reference system in use.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

References OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getX(), and OGRSimpleCurve::getY().

Referenced by get_Area().

12.49.2.7 double OGRCircularString::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.

Reimplemented from **OGRSimpleCurve** (p. **??**).

12.49.2.8 void OGRCircularString::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

psEnvelope	the structure in which to place the results.

Reimplemented from **OGRSimpleCurve** (p. **??**).

References OGRSimpleCurve::getEnvelope().

12.49.2.9 void OGRCircularString::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

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Reimplemented from **OGRSimpleCurve** (p. **??**).

References OGRSimpleCurve::getEnvelope().

12.49.2.10 const char * OGRCircularString::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. **??**).

12.49.2.11 OGRwkbGeometryType OGRCircularString::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()** (p. **??**) 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(), wkbCircularString, and wkbCircularStringZ.

12.49.2.12 OGRGeometry * OGRCircularString::getLinearGeometry (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return, possibly approximate, non-curve version of this geometry.

Returns a geometry that has no CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by approximating curve geometries.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getCurveGeometry()** (p. **??**).

This method is the same as the C function **OGR_G_GetLinearGeometry()** (p. **??**).

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings. See **OGRGeometryFactory::curveToLine↩ String()** (p. **??**) for valid options.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

References CurveToLine().

12.49.2.13 OGRBoolean OGRCircularString::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

12.49.2.14 OGRErr OGRCircularString::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRSimpleCurve** (p. **??**).

References OGRSimpleCurve::empty(), and OGRSimpleCurve::importFromWkb().

12.49.2.15 OGRErr OGRCircularString::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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRSimpleCurve** (p. **??**).

References OGRSimpleCurve::empty(), and OGRSimpleCurve::importFromWkt().

12.49.2.16 OGRBoolean OGRCircularString::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.

Reimplemented from **OGRGeometry** (p. **??**).

References OGRGeometry::IsValid().

12.49.2.17 void OGRCircularString::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Add intermediate vertices to a geometry.

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

dfMaxLength	the maximum distance between 2 points after segmentization

This method modifies the geometry to add intermediate vertices if necessary so that the maximum length between 2 consecutives vertices is lower than dfMaxLength.

Parameters

dfMaxLength	maximum length between 2 consecutives vertices.

Reimplemented from **OGRSimpleCurve** (p. **??**).

References OGRSimpleCurve::reversePoints().

12.49.2.18 void OGRCircularString::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.

This function is the same as the C function **OGR_G_Value()** (p. **??**).

Parameters

dfDistance	distance along the curve at which to sample position. This distance should be between zero and **get_Length()** (p. **??**) for this curve.
poPoint	the point to be assigned the curve position.

Reimplemented from **OGRSimpleCurve** (p. **??**).

References OGRSimpleCurve::EndPoint(), OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGR↩Point::setY(), OGRPoint::setZ(), and OGRSimpleCurve::StartPoint().

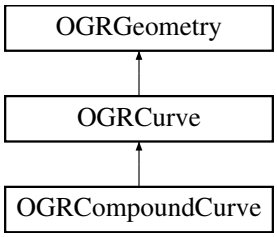
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrcircularstring.cpp

12.50 OGRCompoundCurve Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRCompoundCurve:



Public Member Functions

- **OGRCompoundCurve** ()

  Create an empty compound curve.
- virtual int **WkbSize** () const

  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char ∗, int=-1, **OGRwkbVariant**=**wkbVariantOldOgc**)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char ∗, **OGRwkbVariant**=**wkbVariantOldOgc**) 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, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- 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.
- virtual **OGRLineString** * **CurveToLine** (double dfMaxAngleStepSizeDegrees=0, const char *const *papsz↩ Options=NULL) const

  Return a linestring from a curve geometry.
- virtual int **getNumPoints** () const

  Return the number of points of a curve geometry.
- virtual double **get_AreaOfCurveSegments** () const

  Get the area of the purely curve portions of a (closed) curve.
- virtual double **get_Area** () const

  Get the area of the (closed) curve.
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const

  Returns TRUE if two geometries are equivalent.
- int **getNumCurves** () const

  Return the number of curves.
- **OGRCurve** * **getCurve** (int)

  Fetch reference to indicated internal ring.
- const **OGRCurve** * **getCurve** (int) const

  Fetch reference to indicated internal ring.
- virtual void **setCoordinateDimension** (int nDimension)

  Set the coordinate dimension.
- OGRErr **addCurve** (**OGRCurve** *, double dfToleranceEps=1e-14)

  Add a curve to the container.
- OGRErr **addCurveDirectly** (**OGRCurve** *, double dfToleranceEps=1e-14)

  Add a curve directly to the container.
- virtual **OGRPointIterator** * **getPointIterator** () const

  Returns a point iterator over the curve.
- 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 OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.
- virtual **OGRGeometry** * **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char *const *papszOptions=NULL) const

  Return, possibly approximate, non-curve version of this geometry.
- virtual void **swapXY** ()

  Swap x and y coordinates.

Static Protected Member Functions

- static **OGRLinearRing** ∗ **CastToLinearRing** (**OGRCompoundCurve** ∗poCC)

  Cast to linear ring.

Additional Inherited Members

12.50.1 Detailed Description

Concrete representation of a compound curve, made of curves: **OGRLineString** (p. **??**) and **OGRCircularString** (p. **??**). Each curve is connected by its first point to the last point of the previous curve.

Compatibility: ISO SQL/MM Part 3.

Since

GDAL 2.0

12.50.2 Member Function Documentation

12.50.2.1 OGRErr OGRCompoundCurve::addCurve (OGRCurve ∗ *poCurve*, double *dfToleranceEps* = 1e−14)

Add a curve to the container.

The passed geometry is cloned to make an internal copy.

There is no ISO SQL/MM analog to this method.

This method is the same as the C function **OGR_G_AddGeometry()** (p. **??**).

Parameters

poCurve	geometry to add to the container.
dfToleranceEps	tolerance when checking that the first point of a segment matches then end point of the previous one. Default value: 1e-14.

Returns

OGRERR_NONE if successful, or OGRERR_FAILURE in case of error (for example if curves are not contiguous)

References addCurveDirectly(), and OGRGeometry::clone().

Referenced by clone().

12.50.2.2 OGRErr OGRCompoundCurve::addCurveDirectly (OGRCurve ∗ *poCurve*, double *dfToleranceEps* = 1e−14)

Add a curve directly to the container.

Ownership of the passed geometry is taken by the container rather than cloning as **addCurve()** (p. **??**) does.

There is no ISO SQL/MM analog to this method.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. **??**).

Parameters

poCurve	geometry to add to the container.
dfToleranceEps	tolerance when checking that the first point of a segment matches then end point of the previous one. Default value: 1e-14.

Returns

OGRERR_NONE if successful, or OGRERR_FAILURE in case of error (for example if curves are not contiguous)

Referenced by addCurve(), OGRCurve::CastToCompoundCurve(), and OGRGeometryFactory::curveFromLine↩String().

12.50.2.3 OGRLinearRing * OGRCompoundCurve::CastToLinearRing (OGRCompoundCurve * *poCC*) [static],[protected]

Cast to linear ring.

The passed in geometry is consumed and a new one returned (or NULL in case of failure)

Parameters

poCC	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRGeometry::assignSpatialReference(), OGRCurve::CastToLinearRing(), OGRCurve::CastToLine↩String(), and OGRGeometry::getSpatialReference().

12.50.2.4 OGRGeometry * OGRCompoundCurve::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 addCurve(), OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), and OGRCompoundCurve().

12.50.2.5 OGRLineString * OGRCompoundCurve::CurveToLine (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return a linestring from a curve geometry.

The returned geometry is a new instance whose ownership belongs to the caller.

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.

This method relates to the ISO SQL/MM Part 3 ICurve::CurveToLine() method.

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

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings or NULL. See **OGRGeometryFactory::curveTo↩ LineString()** (p. **??**) for valid options.

Returns

a line string approximating the curve

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

Referenced by get_Area(), and getLinearGeometry().

12.50.2.6 void OGRCompoundCurve::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. **??**).

12.50.2.7 void OGRCompoundCurve::EndPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve end point.

This method relates to the SF COM ICurve::get_EndPoint() method.

Parameters

poPoint	the point to be assigned the end location.

Implements **OGRCurve** (p. **??**).

References OGRCurve::EndPoint().

Referenced by Value().

12.50.2.8 OGRBoolean OGRCompoundCurve::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(), and getGeometryType().

12.50.2.9 OGRErr OGRCompoundCurve::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

References wkbVariantIso, and wkbVariantOldOgc.

12.50.2.10 OGRErr OGRCompoundCurve::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

12.50.2.11 void OGRCompoundCurve::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. **??**).

12.50.2.12 double OGRCompoundCurve::get_Area ( ) const [virtual]

Get the area of the (closed) curve.

This method is designed to be used by **OGRCurvePolygon::get_Area()** (p. **??**).

Returns

the area of the feature in square units of the spatial reference system in use.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

References CurveToLine(), OGRLineString::get_Area(), get_AreaOfCurveSegments(), OGRCurve::get_IsClosed(), OGRPointIterator::getNextPoint(), getNumPoints(), getPointIterator(), OGRPoint::getX(), OGRPoint::getY(), OGR↩Curve::IsConvex(), IsEmpty(), OGRSimpleCurve::setNumPoints(), and OGRSimpleCurve::setPoint().

12.50.2.13 double OGRCompoundCurve::get_AreaOfCurveSegments ( ) const [virtual]

Get the area of the purely curve portions of a (closed) curve.

This method is designed to be used on a closed convex curve.

Returns

the area of the feature in square units of the spatial reference system in use.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

References OGRCurve::get_AreaOfCurveSegments(), getCurve(), and getNumCurves().

Referenced by get_Area().

12.50.2.14 double OGRCompoundCurve::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. **??**).

References OGRCurve::get_Length().

12.50.2.15 OGRCurve * OGRCompoundCurve::getCurve (int *i*)

Fetch reference to indicated internal ring.

Note that the returned curve pointer is to an internal data object of the **OGRCompoundCurve** (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 ISO SQL/MM ST_CurveN() function.

Parameters

iRing	curve index from 0 to **getNumCurves()** (p. **??**) - 1.

Returns

pointer to curve. May be NULL.

Referenced by OGRGeometry::dumpReadable(), get_AreaOfCurveSegments(), and OGRCompoundCurvePoint↩ Iterator::getNextPoint().

12.50.2.16 const OGRCurve * OGRCompoundCurve::getCurve (int *i*) const

Fetch reference to indicated internal ring.

Note that the returned curve pointer is to an internal data object of the **OGRCompoundCurve** (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 ISO SQL/MM ST_CurveN() function.

Parameters

iRing	curve index from 0 to **getNumCurves()** (p. **??**) - 1.

Returns

pointer to curve. May be NULL.

12.50.2.17 void OGRCompoundCurve::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

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. **??**).

12.50.2.18 void OGRCompoundCurve::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

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. **??**).

12.50.2.19 const char * OGRCompoundCurve::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. **??**).

12.50.2.20 OGRwkbGeometryType OGRCompoundCurve::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()** (p. **??**) 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(), wkbCompoundCurve, and wkbCompoundCurveZ.

Referenced by Equals().

12.50.2.21 OGRGeometry * OGRCompoundCurve::getLinearGeometry (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return, possibly approximate, non-curve version of this geometry.

Returns a geometry that has no CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by approximating curve geometries.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getCurveGeometry()** (p. **??**).

This method is the same as the C function **OGR_G_GetLinearGeometry()** (p. **??**).

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings. See **OGRGeometryFactory::curveToLine↩ String()** (p. **??**) for valid options.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

References CurveToLine().

12.50.2.22 int OGRCompoundCurve::getNumCurves ( ) const

Return the number of curves.

Note that the number of curves making this compound curve.

Relates to the ISO SQL/MM ST_NumCurves() function.

Returns

number of curves.

Referenced by OGRGeometryFactory::curveFromLineString(), OGRGeometry::dumpReadable(), get_AreaOf↩CurveSegments(), and OGRCompoundCurvePointIterator::getNextPoint().

12.50.2.23 int OGRCompoundCurve::getNumPoints ( ) const [virtual]

Return the number of points of a curve geometry.

This method, as a method of **OGRCurve** (p. **??**), does not relate to a standard. For circular strings or linestrings, it returns the number of points, conforming to SF COM NumPoints(). For compound curves, it returns the sum of the number of points of each of its components (non including intermediate starting/ending points of the different parts).

Returns

the number of points of the curve.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

References OGRCurve::getNumPoints().

Referenced by get_Area().

12.50.2.24 OGRPointIterator * OGRCompoundCurve::getPointIterator ( ) const [virtual]

Returns a point iterator over the curve.

The curve must not be modified while an iterator exists on it.

The iterator must be destroyed with **OGRPointIterator::destroy()** (p. **??**).

Returns

a point iterator over the curve.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

Referenced by get_Area().

12.50.2.25 OGRBoolean OGRCompoundCurve::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

12.50.2.26 OGRErr OGRCompoundCurve::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

12.50.2.27 OGRErr OGRCompoundCurve::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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

12.50.2.28 OGRBoolean OGRCompoundCurve::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 get_Area().

12.50.2.29 void OGRCompoundCurve::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Add intermediate vertices to a geometry.

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

dfMaxLength	the maximum distance between 2 points after segmentization

This method modifies the geometry to add intermediate vertices if necessary so that the maximum length between 2 consecutives vertices is lower than dfMaxLength.

Parameters

dfMaxLength	maximum length between 2 consecutives vertices.

Reimplemented from **OGRGeometry** (p. **??**).

12.50.2.30 void OGRCompoundCurve::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

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. **??**).

12.50.2.31 void OGRCompoundCurve::StartPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve start point.

This method relates to the SF COM ICurve::get_StartPoint() method.

Parameters

poPoint	the point to be assigned the start location.

Implements **OGRCurve** (p. **??**).

References OGRCurve::StartPoint().

Referenced by Value().

12.50.2.32 void OGRCompoundCurve::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. **??**).

12.50.2.33 OGRErr OGRCompoundCurve::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

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. **??**).

12.50.2.34 void OGRCompoundCurve::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.

This function is the same as the C function **OGR_G_Value()** (p. **??**).

Parameters

dfDistance	distance along the curve at which to sample position. This distance should be between zero and **get_Length()** (p. **??**) for this curve.
poPoint	the point to be assigned the curve position.

Implements **OGRCurve** (p. **??**).

References EndPoint(), OGRCurve::get_Length(), StartPoint(), and OGRCurve::Value().

12.50.2.35 int OGRCompoundCurve::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.

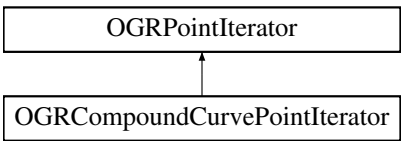
Implements **OGRGeometry** (p. **??**).

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

- **ogr_geometry.h**
- ogrcompoundcurve.cpp

12.51 OGRCompoundCurvePointIterator Class Reference

Inheritance diagram for OGRCompoundCurvePointIterator:



Public Member Functions

- virtual OGRBoolean **getNextPoint** (**OGRPoint** *p)

  Returns the next point followed by the iterator.

Additional Inherited Members

12.51.1 Member Function Documentation

12.51.1.1 OGRBoolean OGRCompoundCurvePointIterator::getNextPoint (OGRPoint * p) [virtual]

Returns the next point followed by the iterator.

Parameters

p	point to fill.

Returns

TRUE in case of success, or FALSE if the end of the curve is reached.

Since

GDAL 2.0

Implements **OGRPointIterator** (p. **??**).

References OGRCompoundCurve::getCurve(), OGRPointIterator::getNextPoint(), OGRCompoundCurve::getNumCurves(), and OGRCurve::getPointIterator().

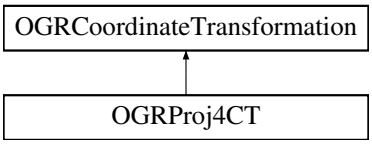
The documentation for this class was generated from the following file:

- ogrcompoundcurve.cpp

12.52 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.*

12.52.1 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.

12.52.2 Member Function Documentation

12.52.2.1 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

poCT	the object to delete

Since

GDAL 1.7.0

12.52.2.2 virtual OGRSpatialReference* OGRCoordinateTransformation::GetSourceCS ( ) [pure virtual]

Fetch internal source coordinate system.

Implemented in **OGRProj4CT** (p. **??**).

12.52.2.3 virtual OGRSpatialReference* OGRCoordinateTransformation::GetTargetCS ( ) [pure virtual]

Fetch internal target coordinate system.

Implemented in **OGRProj4CT** (p. **??**).

Referenced by OGRPoint::transform(), OGRSimpleCurve::transform(), and OGRGeometryCollection::transform().

12.52.2.4 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

nCount	number of points to transform.
x	array of nCount X vertices, modified in place.
y	array of nCount Y vertices, modified in place.
z	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().

12.52.2.5 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

nCount	number of points to transform.
x	array of nCount X vertices, modified in place.
y	array of nCount Y vertices, modified in place.
z	array of nCount Z vertices, modified in place.
pabSuccess	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 OGRSimpleCurve::transform().

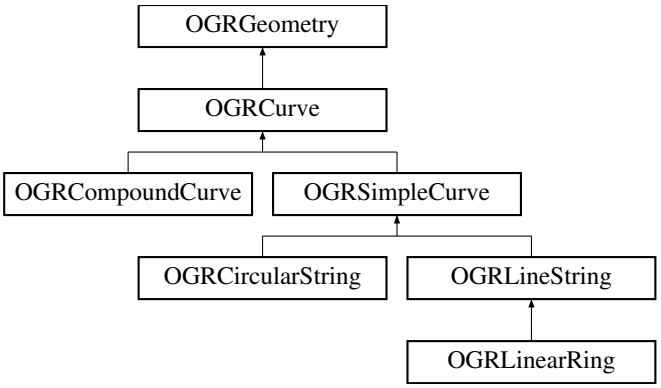
The documentation for this class was generated from the following files:

- **ogr_spatialref.h**
- ogrct.cpp

12.53 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.

- virtual **OGRLineString** ∗ **CurveToLine** (double dfMaxAngleStepSizeDegrees=0, const char ∗const ∗papsz↩ Options=NULL) const =0

  Return a linestring from a curve geometry.

- virtual int **getDimension** () const

  Get the dimension of this object.

- virtual int **getNumPoints** () const =0

  Return the number of points of a curve geometry.

- virtual **OGRPointIterator** ∗ **getPointIterator** () const =0

  Returns a point iterator over the curve.

- virtual OGRBoolean **IsConvex** () const

  Returns if a (closed) curve forms a convex shape.

- virtual double **get_Area** () const =0

  Get the area of the (closed) curve.

Static Public Member Functions

- static **OGRCompoundCurve** ∗ **CastToCompoundCurve** (**OGRCurve** ∗puCurve)

  Cast to compound curve.

- static **OGRLineString** ∗ **CastToLineString** (**OGRCurve** ∗poCurve)

  Cast to linestring.

- static **OGRLinearRing** ∗ **CastToLinearRing** (**OGRCurve** ∗poCurve)

  Cast to linear ring.

Protected Member Functions

- virtual int **ContainsPoint** (const **OGRPoint** ∗p) const

  Returns if a point is contained in a (closed) curve.

- virtual double **get_AreaOfCurveSegments** () const =0

  Get the area of the purely curve portions of a (closed) curve.

Friends

- class **OGRCurvePolygon**
- class **OGRCompoundCurve**

12.53.1 Detailed Description

Abstract curve base class for **OGRLineString** (p. **??**), **OGRCircularString** (p. **??**) and **OGRCompoundCurve** (p. **??**)

12.53.2 Member Function Documentation

12.53.2.1 OGRCompoundCurve ∗ OGRCurve::CastToCompoundCurve (OGRCurve ∗ *poCurve*) [static]

Cast to compound curve.

The passed in geometry is consumed and a new one returned (or NULL in case of failure)

Parameters

poCurve	the input geometry - ownership is passed to the method.

Returns

new geometry

Since

GDAL 2.0

References OGRCompoundCurve::addCurveDirectly(), OGRGeometry::assignSpatialReference(), CastToLine↩String(), OGRGeometry::getGeometryType(), OGRGeometry::getSpatialReference(), OGRGeometry::IsEmpty(), and wkbLineString.

Referenced by OGRGeometryFactory::forceTo().

12.53.2.2 OGRLinearRing * OGRCurve::CastToLinearRing (OGRCurve * *poCurve*) [static]

Cast to linear ring.

The passed in geometry is consumed and a new one returned (or NULL in case of failure)

Parameters

poCurve	the input geometry - ownership is passed to the method.

Returns

new geometry.

Since

GDAL 2.0

Referenced by OGRCompoundCurve::CastToLinearRing(), OGRCurvePolygon::CastToPolygon(), OGRCurve↩Polygon::CurvePolyToPoly(), and OGRGeometryFactory::forceToPolygon().

12.53.2.3 OGRLineString * OGRCurve::CastToLineString (OGRCurve * *poCurve*) [static]

Cast to linestring.

The passed in geometry is consumed and a new one returned (or NULL in case of failure)

Parameters

poCurve	the input geometry - ownership is passed to the method.

Returns

new geometry.

Since

GDAL 2.0

Referenced by CastToCompoundCurve(), OGRCompoundCurve::CastToLinearRing(), OGRMultiCurve::CastTo↩MultiLineString(), OGRGeometryFactory::forceTo(), and OGRGeometryFactory::forceToLineString().

12.53.2.4 int OGRCurve::ContainsPoint (const OGRPoint * p) const [protected],[virtual]

Returns if a point is contained in a (closed) curve.

Final users should use **OGRGeometry::Contains()** (p. **??**) instead.

Parameters

p	the point to test

Returns

TRUE if it is inside the curve, FALSE otherwise or -1 if unknown.

Since

GDAL 2.0

Reimplemented in **OGRCircularString** (p. **??**).

12.53.2.5 OGRLineString * OGRCurve::CurveToLine (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [pure virtual]

Return a linestring from a curve geometry.

The returned geometry is a new instance whose ownership belongs to the caller.

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.

This method relates to the ISO SQL/MM Part 3 ICurve::CurveToLine() method.

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

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings or NULL. See **OGRGeometryFactory::curveTo↩ LineString()** (p. **??**) for valid options.

Returns

a line string approximating the curve

Since

GDAL 2.0

Implemented in **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), and **OGRLineString** (p. **??**).

Referenced by OGRCurvePolygon::CurvePolyToPoly().

12.53.2.6 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

poPoint	the point to be assigned the end location.

Implemented in **OGRCompoundCurve** (p. **??**), and **OGRSimpleCurve** (p. **??**).

Referenced by OGRCompoundCurve::EndPoint(), and get_IsClosed().

12.53.2.7 double OGRCurve::get_Area ( ) const [pure virtual]

Get the area of the (closed) curve.

This method is designed to be used by **OGRCurvePolygon::get_Area()** (p. **??**).

Returns

the area of the feature in square units of the spatial reference system in use.

Since

GDAL 2.0

Implemented in **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), and **OGRLineString** (p. **??**).

Referenced by OGRCurvePolygon::get_Area().

12.53.2.8 double OGRCurve::get_AreaOfCurveSegments ( ) const [protected], [pure virtual]

Get the area of the purely curve portions of a (closed) curve.

This method is designed to be used on a closed convex curve.

Returns

the area of the feature in square units of the spatial reference system in use.

Since

GDAL 2.0

Implemented in **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), and **OGRLineString** (p. **??**).

Referenced by OGRCompoundCurve::get_AreaOfCurveSegments().

12.53.2.9 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().

Referenced by OGRLineString::CastToLinearRing(), OGRGeometryFactory::forceTo(), OGRCircularString::get_↩ Area(), and OGRCompoundCurve::get_Area().

12.53.2.10 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 **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), and **OGRSimpleCurve** (p. **??**).

Referenced by OGRCompoundCurve::get_Length(), and OGRCompoundCurve::Value().

12.53.2.11 int OGRCurve::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::getCoordinate↩ Dimension()** (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. **??**).

12.53.2.12 int OGRCurve::getNumPoints ( ) const [pure virtual]

Return the number of points of a curve geometry.

This method, as a method of **OGRCurve** (p. **??**), does not relate to a standard. For circular strings or linestrings, it returns the number of points, conforming to SF COM NumPoints(). For compound curves, it returns the sum of the number of points of each of its components (non including intermediate starting/ending points of the different parts).

Returns

the number of points of the curve.

Since

GDAL 2.0

Implemented in **OGRCompoundCurve** (p. **??**), and **OGRSimpleCurve** (p. **??**).

Referenced by OGRGeometry::dumpReadable(), OGRGeometryFactory::forceTo(), and OGRCompoundCurve↩ ::getNumPoints().

12.53.2.13 OGRPointIterator * OGRCurve::getPointIterator ( ) const [pure virtual]

Returns a point iterator over the curve.

The curve must not be modified while an iterator exists on it.

The iterator must be destroyed with **OGRPointIterator::destroy()** (p. **??**).

Returns

a point iterator over the curve.

Since

GDAL 2.0

Implemented in **OGRCompoundCurve** (p. **??**), and **OGRSimpleCurve** (p. **??**).

Referenced by OGRCompoundCurvePointIterator::getNextPoint(), and IsConvex().

12.53.2.14 OGRBoolean OGRCurve::IsConvex ( ) const [virtual]

Returns if a (closed) curve forms a convex shape.

Returns

TRUE if the curve forms a convex shape.

Since

GDAL 2.0

References OGRPointIterator::getNextPoint(), getPointIterator(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::setX(), and OGRPoint::setY().

Referenced by OGRCircularString::get_Area(), and OGRCompoundCurve::get_Area().

12.53.2.15 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

poPoint	the point to be assigned the start location.

Implemented in **OGRCompoundCurve** (p. **??**), and **OGRSimpleCurve** (p. **??**).

Referenced by get_IsClosed(), and OGRCompoundCurve::StartPoint().

12.53.2.16 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.

This function is the same as the C function **OGR_G_Value()** (p. **??**).

Parameters

dfDistance	distance along the curve at which to sample position. This distance should be between zero and **get_Length()** (p. **??**) for this curve.
poPoint	the point to be assigned the curve position.

Implemented in **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), and **OGRSimpleCurve** (p. **??**).

Referenced by OGRCompoundCurve::Value().

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

- **ogr_geometry.h**
- ogrcurve.cpp

12.54 OGRCurveCollection Class Reference

`#include <ogr_geometry.h>`

Friends

- class **OGRCompoundCurve**
- class **OGRCurvePolygon**
- class **OGRPolygon**

12.54.1 Detailed Description

Utility class to store a collection of curves. Used as a member of **OGRCompoundCurve** (p. **??**) and **OGRCurve↩ Polygon** (p. **??**).

This class is only exported because of linking issues. It should never be directly used.

Since

GDAL 2.0

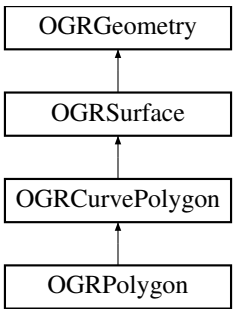
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrcurvecollection.cpp

12.55 OGRCurvePolygon Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRCurvePolygon:



Public Member Functions

- **OGRCurvePolygon** ()

  Create an empty curve 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 OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.
- virtual **OGRGeometry** * **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char *const *papszOptions=NULL) const

  Return, possibly approximate, non-curve version of this geometry.
- virtual double **get_Area** () const

  Get the area of the surface object.
- 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, **OGRwkbVariant**=**wkbVariantOldOgc**)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *, **OGRwkbVariant**=**wkbVariantOldOgc**) 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, **OGRwkbVariant** eWkbVariant=**wkbVariantOldOgc**) 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 **OGRPolygon** * **CurvePolyToPoly** (double dfMaxAngleStepSizeDegrees=0, const char *const *papszOptions=NULL) const

  Return a polygon from a curve polygon.
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const

  Returns TRUE if two geometries are equivalent.
- virtual OGRBoolean **Intersects** (const **OGRGeometry** *) const

  Do these features intersect?
- virtual OGRBoolean **Contains** (const **OGRGeometry** *) const

  Test for containment.
- virtual void **setCoordinateDimension** (int nDimension)

  Set the coordinate dimension.
- OGRErr **addRing** (**OGRCurve** *)

  Add a ring to a polygon.

- OGRErr **addRingDirectly** (**OGRCurve** ∗)

  Add a ring to a polygon.
- **OGRCurve** ∗ **getExteriorRingCurve** ()

  Fetch reference to external polygon ring.
- int **getNumInteriorRings** () const

  Fetch the number of internal rings.
- **OGRCurve** ∗ **getInteriorRingCurve** (int)

  Fetch reference to indicated internal ring.
- **OGRCurve** ∗ **stealExteriorRingCurve** ()

  "Steal" reference to external ring.
- virtual void **swapXY** ()

  Swap x and y coordinates.

Static Protected Member Functions

- static **OGRPolygon** ∗ **CastToPolygon** (**OGRCurvePolygon** ∗poCP)

  Convert to polygon.

Friends

- class **OGRPolygon**

Additional Inherited Members

12.55.1 Detailed Description

Concrete class representing curve polygons.

Note that curve polygons consist of one outer (curve) ring, and zero or more inner rings. A curve polygon cannot represent disconnected regions (such as multiple islands in a political body). The **OGRMultiSurface** (p. **??**) must be used for this.

Compatibility: ISO SQL/MM Part 3.

Since

GDAL 2.0

12.55.2 Member Function Documentation

12.55.2.1 OGRErr OGRCurvePolygon::addRing (OGRCurve ∗ *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 **OGRCurve** (p. **??**) remains the responsibility of the caller (an internal copy is made).

This method has no SFCOM analog.

Parameters

poNewRing	ring to be added to the polygon.

Returns

OGRERR_NONE in case of success

References addRingDirectly(), and OGRGeometry::clone().

Referenced by clone(), and OGRLayer::SetSpatialFilterRect().

12.55.2.2 OGRErr OGRCurvePolygon::addRingDirectly (OGRCurve * *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 **OGRCurvePolygon** (p. **??**), but otherwise this method operates the same as OGRCurvePolygon::AddRing().

This method has no SFCOM analog.

Parameters

poNewRing	ring to be added to the polygon.

Returns

OGRERR_NONE in case of success

Referenced by addRing(), CurvePolyToPoly(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToPolygon(), OGRPolygon::getCurveGeometry(), OGRBuildPolygonFromEdges(), and OGRGeometryFactory::organizePolygons().

12.55.2.3 OGRPolygon * OGRCurvePolygon::CastToPolygon (OGRCurvePolygon * *poCP*) [static], [protected]

Convert to polygon.

This method should only be called if the curve polygon actually only contains instances of **OGRLineString** (p. **??**). This can be verified if hasCurveGeometry(TRUE) returns FALSE. It is not intended to approximate curve polygons. For that use **getLinearGeometry()** (p. **??**).

The passed in geometry is consumed and a new one returned (or NULL in case of failure).

Parameters

poMS	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRGeometry::assignSpatialReference(), OGRCurve::CastToLinearRing(), OGRGeometry::getCoordinateDimension(), OGRGeometry::getSpatialReference(), and setCoordinateDimension().

12.55.2.4 OGRGeometry * OGRCurvePolygon::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(), OGRGeometryFactory::createGeometry(), getGeometryType(), and OGRGeometry::getSpatialReference().

Referenced by OGRPolygon::CurvePolyToPoly(), and OGRPolygon::getCurveGeometry().

12.55.2.5 OGRBoolean OGRCurvePolygon::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

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if poOtherGeom contains this geometry, otherwise FALSE.

Reimplemented from **OGRGeometry** (p. **??**).

References OGRGeometry::Contains(), OGRGeometry::getGeometryType(), IsEmpty(), wkbFlatten, and wkbPoint.

12.55.2.6 OGRPolygon * OGRCurvePolygon::CurvePolyToPoly (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return a polygon from a curve polygon.

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

The returned geometry is a new instance whose ownership belongs to the caller.

Parameters

dfMaxAngleStepSizeDegrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

Returns

a linestring

Since

OGR 2.0

Reimplemented in **OGRPolygon** (p. **??**).

References addRingDirectly(), OGRGeometry::assignSpatialReference(), OGRCurve::CastToLinearRing(), OGRCurve::CurveToLine(), and OGRGeometry::getSpatialReference().

Referenced by getLinearGeometry(), and PointOnSurface().

12.55.2.7 void OGRCurvePolygon::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. **??**).

12.55.2.8 OGRBoolean OGRCurvePolygon::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(), OGRGeometry::IsEmpty(), and IsEmpty().

12.55.2.9 OGRErr OGRCurvePolygon::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRPolygon** (p. **??**).

References wkbVariantIso, and wkbVariantOldOgc.

12.55.2.10 OGRErr OGRCurvePolygon::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, ∗ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRPolygon** (p. **??**).

12.55.2.11 void OGRCurvePolygon::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. **??**).

12.55.2.12 double OGRCurvePolygon::get_Area ( ) const [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.

Implements **OGRSurface** (p. **??**).

References OGRCurve::get_Area(), getExteriorRingCurve(), getInteriorRingCurve(), and getNumInteriorRings().

Referenced by OGRGeometryFactory::organizePolygons().

12.55.2.13 int OGRCurvePolygon::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::getCoordinate↩ Dimension()** (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. **??**).

12.55.2.14 void OGRCurvePolygon::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

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. **??**).

12.55.2.15 void OGRCurvePolygon::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

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. **??**).

12.55.2.16 OGRCurve * OGRCurvePolygon::getExteriorRingCurve ( )

Fetch reference to external polygon ring.

Note that the returned ring pointer is to an internal data object of the **OGRCurvePolygon** (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 **OGRCurvePolygon** (p. **??**) is empty.

Referenced by OGRGeometry::dumpReadable(), OGRGeometryFactory::forceTo(), and get_Area().

12.55.2.17 const char * OGRCurvePolygon::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 **OGRPolygon** (p. **??**).

12.55.2.18 OGRwkbGeometryType OGRCurvePolygon::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()** (p. **??**) 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 **OGRPolygon** (p. **??**).

References wkbCurvePolygon, and wkbCurvePolygonZ.

Referenced by clone(), and Equals().

12.55.2.19 OGRCurve * OGRCurvePolygon::getInteriorRingCurve (int *iRing*)

Fetch reference to indicated internal ring.

Note that the returned ring pointer is to an internal data object of the **OGRCurvePolygon** (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

iRing	internal ring index from 0 to getNumInternalRings() - 1.

Returns

pointer to interior ring. May be NULL.

Referenced by OGRGeometry::dumpReadable(), and get_Area().

12.55.2.20 OGRGeometry * OGRCurvePolygon::getLinearGeometry (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return, possibly approximate, non-curve version of this geometry.

Returns a geometry that has no CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by approximating curve geometries.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getCurveGeometry()** (p. **??**).

This method is the same as the C function **OGR_G_GetLinearGeometry()** (p. **??**).

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.

papszOptions	options as a null-terminated list of strings. See **OGRGeometryFactory::curveToLine↩ String()** (p. **??**) for valid options.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

Reimplemented in **OGRPolygon** (p. **??**).

References CurvePolyToPoly().

12.55.2.21 int OGRCurvePolygon::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(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::force↩ ToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), and get_Area().

12.55.2.22 OGRBoolean OGRCurvePolygon::hasCurveGeometry (int *bLookForNonLinear* = `FALSE`) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

Reimplemented in **OGRPolygon** (p. **??**).

12.55.2.23 OGRErr OGRCurvePolygon::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRPolygon** (p. **??**).

12.55.2.24 OGRErr OGRCurvePolygon::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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRPolygon** (p. **??**).

12.55.2.25 OGRBoolean OGRCurvePolygon::Intersects (const OGRGeometry * *poOtherGeom*) const [virtual]

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigerous 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

poOtherGeom	the other geometry to test against.

Returns

TRUE if the geometries intersect, otherwise FALSE.

Reimplemented from **OGRGeometry** (p. **??**).

References OGRGeometry::getGeometryType(), OGRGeometry::Intersects(), IsEmpty(), wkbFlatten, and wkb↩Point.

12.55.2.26 OGRBoolean OGRCurvePolygon::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 Contains(), Equals(), OGRPolygon::exportToWkt(), and Intersects().

12.55.2.27 int OGRCurvePolygon::PointOnSurface (OGRPoint * *poPoint*) const [virtual]

This method relates to the SFCOM ISurface::get_PointOnSurface() method.

NOTE: Only implemented when GEOS included in build.

Parameters

poPoint	point to be set with an internal point.

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implements **OGRSurface** (p. **??**).

Reimplemented in **OGRPolygon** (p. **??**).

References CurvePolyToPoly(), and OGRPolygon::PointOnSurface().

12.55.2.28 void OGRCurvePolygon::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Add intermediate vertices to a geometry.

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

dfMaxLength	the maximum distance between 2 points after segmentization

This method modifies the geometry to add intermediate vertices if necessary so that the maximum length between 2 consecutives vertices is lower than dfMaxLength.

Parameters

dfMaxLength	maximum length between 2 consecutives vertices.

Reimplemented from **OGRGeometry** (p. **??**).

12.55.2.29 void OGRCurvePolygon::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

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. **??**).

Referenced by OGRPolygon::CastToCurvePolygon(), and CastToPolygon().

12.55.2.30 OGRCurve * OGRCurvePolygon::stealExteriorRingCurve ( )

"Steal" reference to external ring.

After the call to that function, only call to stealInteriorRing() or destruction of the **OGRCurvePolygon** (p. **??**) is valid. Other operations may crash.

Returns

pointer to external ring. May be NULL if the **OGRCurvePolygon** (p. **??**) is empty.

Referenced by OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToLineString(), and OGRPolygon↩ ::stealExteriorRing().

12.55.2.31 void OGRCurvePolygon::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. **??**).

12.55.2.32 OGRErr OGRCurvePolygon::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

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. **??**).

12.55.2.33 int OGRCurvePolygon::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 **OGRPolygon** (p. **??**).

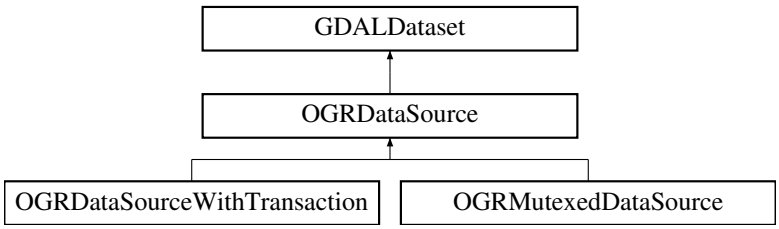
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrcurvepolygon.cpp

12.56 OGRDataSource Class Reference

`#include <ogrsf_frmts.h>`

Inheritance diagram for OGRDataSource:



12.56.1 Detailed Description

LEGACY class. Use GDALDataset in your new code ! This class may be removed in a later release.

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.

NOTE: Starting with GDAL 2.0, it is *NOT* safe to cast the handle of a C function that returns a OGRDataSourceH to a OGRDataSource∗. If a C++ object is needed, the handle should be cast to GDALDataset∗.

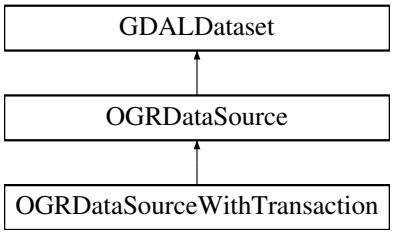
Deprecated

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

- **ogrsf_frmts.h**
- ogrdatasource.cpp

12.57 OGRDataSourceWithTransaction Class Reference

Inheritance diagram for OGRDataSourceWithTransaction:



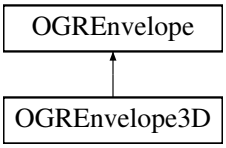
The documentation for this class was generated from the following file:

- ogremulatedtransaction.cpp

12.58 OGREnvelope Class Reference

`#include <ogr_core.h>`

Inheritance diagram for OGREnvelope:



12.58.1 Detailed Description

Simple container for a bounding region.

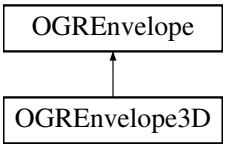
The documentation for this class was generated from the following file:

- **ogr_core.h**

12.59 OGREnvelope3D Class Reference

`#include <ogr_core.h>`

Inheritance diagram for OGREnvelope3D:



12.59.1 Detailed Description

Simple container for a bounding region in 3D.

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

- **ogr_core.h**

12.60 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.
- int **GetGeomFieldCount** ()
  *Fetch number of geometry fields on this feature. This will always be the same as the geometry field count for the **OGRFeatureDefn** (p. **??**).*
- **OGRGeomFieldDefn** ∗ **GetGeomFieldDefnRef** (int iField)
  Fetch definition for this geometry field.
- int **GetGeomFieldIndex** (const char ∗pszName)
  Fetch the geometry field index given geometry field name.
- **OGRGeometry** ∗ **GetGeomFieldRef** (int iField)
  Fetch pointer to feature geometry.
- **OGRGeometry** ∗ **GetGeomFieldRef** (const char ∗pszFName)
  Fetch pointer to feature geometry.
- OGRErr **SetGeomFieldDirectly** (int iField, **OGRGeometry** ∗)
  Set feature geometry of a specified geometry field.

- OGRErr **SetGeomField** (int iField, **OGRGeometry** ∗)

  Set feature geometry of a specified geometry field.
- **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)

  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.
- GIntBig **GetFieldAsInteger64** (int i)

  Fetch field value as integer 64 bit.
- 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 GIntBig ∗ **GetFieldAsInteger64List** (int i, int ∗pnCount)

  Fetch field value as a list of 64 bit integers.
- const double ∗ **GetFieldAsDoubleList** (int i, int ∗pnCount)

  Fetch field value as a list of doubles.
- char ∗∗ **GetFieldAsStringList** (int i)

  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, float ∗pf↩ Second, int ∗pnTZFlag)

  Fetch field value as date and time.
- void **SetField** (int i, int nValue)

  Set field to integer value.
- void **SetField** (int i, GIntBig nValue)

  Set field to 64 bit 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, const GIntBig ∗panValues)

Set field to list of 64 bit 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, float fSecond=0.f, int nTZ↩ Flag=0)

  Set field to date.

- GIntBig **GetFID** ()

  Get feature identifier.

- virtual OGRErr **SetFID** (GIntBig nFIDIn)

  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.

- int **Validate** (int nValidateFlags, int bEmitError)

  Validate that a feature meets constraints of its schema.

- void **FillUnsetWithDefault** (int bNotNullableOnly, char **papszOptions)

  Fill unset fields with default values that might be defined.

- 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.

12.60.1 Detailed Description

A simple feature, including geometry and attributes.

12.60.2 Constructor & Destructor Documentation

12.60.2.1 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

poDefnIn	feature class (layer) definition to which the feature will adhere.

References CPLCalloc(), CPLMalloc(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetGeomFieldCount(), and OGRFeatureDefn::Reference().

Referenced by Clone(), and CreateFeature().

12.60.3 Member Function Documentation

12.60.3.1 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(), OGRFeatureDefn::GetGeomFieldCount(), GetStyleString(), OGRFeature(), SetFID(), SetField(), SetGeomField(), and SetStyleString().

Referenced by OGRGenSQLResultsLayer::GetFeature().

12.60.3.2 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

poDefn	Feature definition defining schema.

Returns

new feature object with null fields and no geometry. May be deleted with delete.

References OGRFeature().

12.60.3.3 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

poFeature	the feature to delete.

12.60.3.4 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

fpOut	the stream to write to, such as stdout. If NULL stdout will be used.
papszOptions	NULL terminated list of options (may be NULL)

References CSLTestBoolean(), OGRGeometry::dumpReadable(), GetFID(), GetFieldAsString(), GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetFieldSubTypeName(), OGRFieldDefn::GetFieldTypeName(), GetGeomFieldCount(), OGRFeatureDefn::GetGeomFieldDefn(), OGRFeatureDefn::GetName(), OGRFieldDefn::GetNameRef(), OGRGeomFieldDefn::GetNameRef(), GetStyleString(), OGRFieldDefn::GetSubType(), OGRFieldDefn::GetType(), IsFieldSet(), and OFSTNone.

12.60.3.5 OGRBoolean OGRFeature::Equal (OGRFeature * *poFeature*) [virtual]

Test if two features are the same.

Two features are considered equal if the share them (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

poFeature	the other feature to test this one against.

Returns

TRUE if they are equal, otherwise FALSE.

References CSLCount(), OGRGeometry::Equals(), GetDefnRef(), GetFID(), GetFieldAsBinary(), GetFieldAsDouble(), GetFieldAsDoubleList(), GetFieldAsInteger(), GetFieldAsInteger64(), GetFieldAsInteger64List(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldAsStringList(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetGeomFieldCount(), GetGeomFieldRef(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

12.60.3.6 void OGRFeature::FillUnsetWithDefault (int *bNotNullableOnly,* char ** *papszOptions*)

Fill unset fields with default values that might be defined.

This method is the same as the C function **OGR_F_FillUnsetWithDefault()** (p. **??**).

Parameters

bNotNullable↩ Only	if we should fill only unset fields with a not-null constraint.
papszOptions	unused currently. Must be set to NULL.

Since

GDAL 2.0

References CPLUnescapeString(), OGRFieldDefn::GetDefault(), OGRFeatureDefn::GetFieldCount(), OGR↩ FeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OGRFieldDefn::IsNullable(), OFTDate, OFTDateTime, OFTString, OFTTime, and SetField().

12.60.3.7 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().

12.60.3.8 GIntBig OGRFeature::GetFID ( ) [inline]

Get feature identifier.

This method is the same as the C function **OGR_F_GetFID()** (p. **??**). Note: since GDAL 2.0, this method returns a GIntBig (previously a long)

Returns

feature id or OGRNullFID if none has been assigned.

Referenced by Clone(), DumpReadable(), Equal(), OGRLayerWithTransaction::GetFeature(), OGRLayer::Get↩ Feature(), GetFieldAsDouble(), GetFieldAsString(), OGRLayerWithTransaction::GetNextFeature(), OGRLayer↩ WithTransaction::ICreateFeature(), OGRUnionLayer::ICreateFeature(), OGRLayerWithTransaction::ISetFeature(), and OGRUnionLayer::ISetFeature().

12.60.3.9 GByte * OGRFeature::GetFieldAsBinary (int *iField,* int * *pnBytes*)

Fetch field value as binary data.

This method only works for OFTBinary and OFTString fields.

This method is the same as the C function **OGR_F_GetFieldAsBinary()** (p. **??**).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
pnBytes	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(), OFTBinary, and OFTString.

Referenced by Equal().

12.60.3.10 int OGRFeature::GetFieldAsDateTime (int *iField*, int * *pnYear*, int * *pnMonth*, int * *pnDay*, int * *pnHour*, int * *pnMinute*, float * *pfSecond*, 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
pnYear	(including century)
pnMonth	(1-12)
pnDay	(1-31)
pnHour	(0-23)
pnMinute	(0-59)
pfSecond	(0-59 with millisecond accuracy)
pnTZFlag	(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.

12.60.3.11 double OGRFeature::GetFieldAsDouble (int *iField*)

Fetch field value as a double.

OFTString features will be translated using **CPLAtof()** (p. **??**). OFTInteger and OFTInteger64 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.

Returns

the field value.

References CPLAtof(), GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetGeomFieldCount(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTInteger64, OFTReal, OFTString, and OGR_G_Area().

Referenced by Equal(), and SetFieldsFrom().

12.60.3.12 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
pnCount	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().

12.60.3.13 int OGRFeature::GetFieldAsInteger (int *iField*)

Fetch field value as integer.

OFTString features will be translated using atoi(). OFTReal fields will be cast to integer. OFTInteger64 are demoted to 32 bit, with clamping if out-of-range. 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.

Returns

the field value.

References CPLError(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetGeomFieldCount(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTInteger64, OFTReal, OFTString, and OGR_G_Area().

Referenced by Equal(), and SetFieldsFrom().

12.60.3.14 GIntBig OGRFeature::GetFieldAsInteger64 (int *iField*)

Fetch field value as integer 64 bit.

OFTInteger are promoted to 64 bit. OFTString features will be translated using **CPLAtoGIntBig()** (p. **??**). 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_GetFieldAsInteger64()** (p. **??**).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.

Returns

the field value.

Since

GDAL 2.0

References CPLAtoGIntBigEx(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetGeomFieldCount(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTInteger64, OFTReal, OFTString, and OGR_G_Area().

Referenced by Equal(), and SetFieldsFrom().

12.60.3.15 const GIntBig * OGRFeature::GetFieldAsInteger64List (int *iField,* int * *pnCount*)

Fetch field value as a list of 64 bit integers.

Currently this method only works for OFTInteger64List fields.

This method is the same as the C function **OGR_F_GetFieldAsInteger64List()** (p. **??**).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
pnCount	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.

Since

GDAL 2.0

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTInteger64List.

Referenced by Equal(), and SetFieldsFrom().

12.60.3.16 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
pnCount	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().

12.60.3.17 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

iField	the field to fetch, from 0 to **GetFieldCount()** (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(), CPLsnprintf(), CPLStrdup(), GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRGeometry::getGeometryName(), GetGeomFieldCount(), OGRFieldDefn::GetPrecision(), GetStyleString(), OGRFieldDefn::GetType(), OGRFieldDefn::GetWidth(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, OFTTime, and OGR_G_Area().

Referenced by DumpReadable(), Equal(), GetStyleString(), OGRUnionLayer::ICreateFeature(), OGRUnionLayer::ISetFeature(), SetFieldsFrom(), and Validate().

12.60.3.18 char ** OGRFeature::GetFieldAsStringList (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 method is the same as the C function **OGR_F_GetFieldAsStringList()** (p. **??**).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (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().

12.60.3.19 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.

References OGRFeatureDefn::GetFieldCount().

Referenced by DumpReadable(), OGR_F_IsFieldSet(), SetFieldsFrom(), and SetFrom().

12.60.3.20 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.

Returns

the field definition (from the **OGRFeatureDefn** (p. **??**)). This is an internal reference, and should not be deleted or modified.

References OGRFeatureDefn::GetFieldDefn().

Referenced by SetFieldsFrom(), and SetFrom().

12.60.3.21 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

pszName	the name of the field to search for.

Returns

the field index, or -1 if no matching field is found.

References OGRFeatureDefn::GetFieldIndex().

Referenced by GetStyleString(), and SetFrom().

12.60.3.22 OGRGeometry * OGRFeature::GetGeometryRef ( )

Fetch pointer to feature geometry.

This method is the same as the C function **OGR_F_GetGeometryRef()** (p. **??**).

Starting with GDAL 1.11, this is equivalent to calling OGRFeature::GetGeomFieldRef(0).

Returns

pointer to internal feature geometry. This object should not be modified.

References GetGeomFieldCount(), and GetGeomFieldRef().

Referenced by OGRLayer::Clip(), OGRLayer::Erase(), and OGR_F_GetGeometryRef().

12.60.3.23 int OGRFeature::GetGeomFieldCount ( ) [inline]

Fetch number of geometry fields on this feature. This will always be the same as the geometry field count for the **OGRFeatureDefn** (p. **??**).

This method is the same as the C function **OGR_F_GetGeomFieldCount()** (p. **??**).

Returns

count of geometry fields.

Since

GDAL 1.11

References OGRFeatureDefn::GetGeomFieldCount().

Referenced by DumpReadable(), Equal(), GetFieldAsDouble(), GetFieldAsInteger(), GetFieldAsInteger64(), GetFieldAsString(), GetGeometryRef(), GetGeomFieldRef(), IsFieldSet(), SetFrom(), SetGeometry(), SetGeometryDirectly(), SetGeomField(), SetGeomFieldDirectly(), and StealGeometry().

12.60.3.24 OGRGeomFieldDefn * OGRFeature::GetGeomFieldDefnRef (int *iGeomField*) [inline]

Fetch definition for this geometry field.

This method is the same as the C function **OGR_F_GetGeomFieldDefnRef()** (p. **??**).

Parameters

iGeomField	the field to fetch, from 0 to **GetGeomFieldCount()** (p. **??**)-1.

Returns

the field definition (from the **OGRFeatureDefn** (p. **??**)). This is an internal reference, and should not be deleted or modified.

Since

GDAL 1.11

References OGRFeatureDefn::GetGeomFieldDefn().

Referenced by SetFrom().

12.60.3.25 int OGRFeature::GetGeomFieldIndex (const char * *pszName*) [inline]

Fetch the geometry field index given geometry field name.

This is a cover for the **OGRFeatureDefn::GetGeomFieldIndex()** (p. **??**) method.

This method is the same as the C function **OGR_F_GetGeomFieldIndex()** (p. **??**).

Parameters

pszName	the name of the geometry field to search for.

Returns

the geometry field index, or -1 if no matching geometry field is found.

Since

GDAL 1.11

References OGRFeatureDefn::GetGeomFieldIndex().

Referenced by GetGeomFieldRef(), and SetFrom().

12.60.3.26 OGRGeometry * OGRFeature::GetGeomFieldRef (int *iField*)

Fetch pointer to feature geometry.

This method is the same as the C function **OGR_F_GetGeomFieldRef()** (p. **??**).

Parameters

iField	geometry field to get.

Returns

pointer to internal feature geometry. This object should not be modified.

Since

GDAL 1.11

References GetGeomFieldCount().

Referenced by Equal(), GetGeometryRef(), OGRWarpedLayer::GetNextFeature(), OGRGenSQLResultsLayer::↩ GetNextFeature(), OGRUnionLayer::GetNextFeature(), OGR_F_GetGeomFieldRef(), SetFrom(), and Validate().

12.60.3.27 OGRGeometry * OGRFeature::GetGeomFieldRef (const char * *pszFName*)

Fetch pointer to feature geometry.

Parameters

pszFName	name of geometry field to get.

Returns

pointer to internal feature geometry. This object should not be modified.

Since

GDAL 1.11

References GetGeomFieldIndex().

12.60.3.28 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.

Returns

the returned pointer is to an internal data structure, and should not be freed, or modified.

Referenced by SetFieldsFrom().

12.60.3.29 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().

12.60.3.30 int OGRFeature::IsFieldSet (int *iField*)

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

iField	the field to test.

Returns

TRUE if the field has been set, otherwise false.

References OGRFeatureDefn::GetFieldCount(), GetGeomFieldCount(), and OGR_G_Area().

Referenced by DumpReadable(), Equal(), FillUnsetWithDefault(), GetFieldAsBinary(), GetFieldAsDateTime(), GetFieldAsDouble(), GetFieldAsDoubleList(), GetFieldAsInteger(), GetFieldAsInteger64(), GetFieldAsInteger64↩List(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldAsStringList(), OGRUnionLayer::ICreateFeature(), OGRUnionLayer::ISetFeature(), OGR_F_IsFieldSet(), SetField(), SetFieldsFrom(), UnsetField(), and Validate().

12.60.3.31 OGRErr OGRFeature::SetFID (GIntBig *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

nFID	the new feature identifier value to assign.

Returns

On success OGRERR_NONE, or on failure some other value.

Referenced by Clone(), OGRLayerWithTransaction::GetFeature(), OGRGenSQLResultsLayer::GetFeature(), O↩GRLayerWithTransaction::GetNextFeature(), OGRLayerWithTransaction::ICreateFeature(), OGRUnionLayer::I↩CreateFeature(), OGRLayerWithTransaction::ISetFeature(), OGRUnionLayer::ISetFeature(), and SetFrom().

12.60.3.32 void OGRFeature::SetField (int *iField,* int *nValue*)

Set field to integer value.

OFTInteger, OFTInteger64 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
nValue	the value to assign.

References CPLStrdup(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, and OFTString.

Referenced by Clone(), FillUnsetWithDefault(), OGRGenSQLResultsLayer::GetFeature(), SetField(), and SetFieldsFrom().

12.60.3.33 void OGRFeature::SetField (int *iField*, GIntBig *nValue*)

Set field to 64 bit integer value.

OFTInteger, OFTInteger64 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_SetFieldInteger64()** (p. **??**).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
nValue	the value to assign.

Since

GDAL 2.0

References CPLError(), CPLStrdup(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, and SetField().

12.60.3.34 void OGRFeature::SetField (int *iField*, double *dfValue*)

Set field to double value.

OFTInteger, OFTInteger64 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
dfValue	the value to assign.

References CPLsprintf(), CPLStrdup(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, and SetField().

12.60.3.35 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. OFTInteger64 fields will be set based on an **CPLAtoGIntBig()** (p. **??**) conversion of the string. OFTReal fields will be set based on an **CPLAtof()** (p. **??**) conversion of the string. Other field types may be unaffected.

This method is the same as the C function **OGR_F_SetFieldString()** (p. **??**).

Parameters

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
pszValue	the value to assign.

References CPLAtof(), CPLAtoGIntBigEx(), CPLError(), CPLGetConfigOption(), CPLStrdup(), CPLStrtod(), CSLCount(), CSLDestroy(), CSLTestBoolean(), CSLTokenizeString2(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetName(), OGRFieldDefn::GetNameRef(), OGRFieldDefn::GetType(), IsFieldSet(), OFTDate, OFTDateTime, OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, OFTTime, and SetField().

12.60.3.36 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, OFTInteger64List and OFTRealList fields.

This method is the same as the C function **OGR_F_SetFieldIntegerList()** (p. **??**).

Parameters

iField	the field to set, from 0 to **GetFieldCount()** (p. **??**)-1.
nCount	the number of values in the list being assigned.
panValues	the values to assign.

References CPLMalloc(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetSubType(), OGRFieldDefn::GetType(), OFSTBoolean, OFSTInt16, OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, and SetField().

12.60.3.37 void OGRFeature::SetField (int *iField*, int *nCount*, const GIntBig * *panValues*)

Set field to list of 64 bit integers value.

This method currently on has an effect of OFTIntegerList, OFTInteger64List and OFTRealList fields.

This method is the same as the C function OGR_F_SetFieldIntege64rList().

Parameters

iField	the field to set, from 0 to **GetFieldCount()** (p. **??**)-1.
nCount	the number of values in the list being assigned.
panValues	the values to assign.

Since

GDAL 2.0

References CPLError(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, and SetField().

12.60.3.38 void OGRFeature::SetField (int *iField*, int *nCount*, double * *padfValues*)

Set field to list of doubles value.

This method currently on has an effect of OFTIntegerList, OFTInteger64List, OFTRealList fields.

This method is the same as the C function **OGR_F_SetFieldDoubleList()** (p. **??**).

Parameters

iField	the field to set, from 0 to **GetFieldCount()** (p. **??**)-1.
nCount	the number of values in the list being assigned.
padfValues	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, and SetField().

12.60.3.39 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

iField	the field to set, from 0 to **GetFieldCount()** (p. **??**)-1.
papszValues	the values to assign.

References CSLCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTStringList, and SetField().

12.60.3.40 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

iField	the field to fetch, from 0 to **GetFieldCount()** (p. **??**)-1.
puValue	the value to assign.

References CPLMalloc(), CPLStrdup(), CSLCount(), CSLDestroy(), CSLDuplicate(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

12.60.3.41 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

iField	the field to set, from 0 to **GetFieldCount()** (p. **??**)-1.
nBytes	bytes of data being set.
pabyData	the raw data being applied.

References CPLMalloc(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTBinary, OFTString, and SetField().

12.60.3.42 void OGRFeature::SetField (int *iField,* int *nYear,* int *nMonth,* int *nDay,* int *nHour* = 0, int *nMinute* = 0, float *fSecond* = 0.f, 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

iField	the field to set, from 0 to **GetFieldCount()** (p. **??**)-1.
nYear	(including century)
nMonth	(1-12)
nDay	(1-31)
nHour	(0-23)
nMinute	(0-59)
fSecond	(0-59, with millisecond accuracy)
nTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

References CPLError(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTDate, OFTDateTime, and OFTTime.

12.60.3.43 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

poSrcFeature	the feature from which geometry, and field values will be copied.
panMap	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.
bForgiving	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(), GetFieldAsInteger64(), GetField↩ AsInteger64List(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldCount(), GetFieldDefnRef(), GetRaw↩ FieldRef(), OGRFieldDefn::GetType(), IsFieldSet(), OFTDate, OFTDateTime, OFTInteger, OFTInteger64, OFT↩ Integer64List, 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().

12.60.3.44 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

poSrcFeature	the feature from which geometry, and field values will be copied.
bForgiving	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 OGRLayerWithTransaction::GetFeature(), OGRLayerWithTransaction::GetNextFeature(), OGR↩LayerWithTransaction::ICreateFeature(), OGRUnionLayer::ICreateFeature(), OGRLayerWithTransaction::ISet↩Feature(), and OGRUnionLayer::ISetFeature().

12.60.3.45 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

poSrcFeature	the feature from which geometry, and field values will be copied.
panMap	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.
bForgiving	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 GetGeomFieldCount(), GetGeomFieldDefnRef(), GetGeomFieldIndex(), GetGeomFieldRef(), OG↩RGeomFieldDefn::GetNameRef(), GetStyleString(), SetFID(), SetFieldsFrom(), SetGeomField(), and SetStyle↩String().

12.60.3.46 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

poGeomIn	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 GetGeomFieldCount(), and SetGeomField().

Referenced by OGRLayer::Update().

12.60.3.47 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 (even in case of failure of that function).

This method is the same as the C function **OGR_F_SetGeometryDirectly()** (p. **??**).

Parameters

poGeomIn	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 GetGeomFieldCount(), and SetGeomFieldDirectly().

Referenced by OGRLayer::Clip(), OGRLayer::Erase(), OGRLayer::Identity(), OGRLayer::Intersection(), OGRLayer::SymDifference(), OGRLayer::Union(), and OGRLayer::Update().

12.60.3.48 OGRErr OGRFeature::SetGeomField (int *iField,* OGRGeometry * *poGeomIn*)

Set feature geometry of a specified geometry field.

This method updates the features geometry, and operate exactly as **SetGeomFieldDirectly()** (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_SetGeomField()** (p. **??**).

Parameters

iField	geometry field to set.
poGeomIn	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 OGRERR_FAILURE if the index is invalid, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. **??**) (checking not yet implemented).

Since

GDAL 1.11

References OGRGeometry::clone(), and GetGeomFieldCount().

Referenced by Clone(), SetFrom(), and SetGeometry().

12.60.3.49 OGRErr OGRFeature::SetGeomFieldDirectly (int *iField,* OGRGeometry * *poGeomIn*)

Set feature geometry of a specified geometry field.

This method updates the features geometry, and operate exactly as **SetGeomField()** (p. **??**), except that this method assumes ownership of the passed geometry (even in case of failure of that function).

This method is the same as the C function **OGR_F_SetGeomFieldDirectly()** (p. **??**).

Parameters

iField	geometry field to set.
poGeomIn	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 OGRERR_FAILURE if the index is invalid, or OGR_UNSUPPORTED_↩ GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. **??**) (checking not yet implemented).

Since

GDAL 1.11

References GetGeomFieldCount().

Referenced by OGR_F_GetGeometryRef(), OGR_F_GetGeomFieldRef(), and SetGeometryDirectly().

12.60.3.50 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

pszString	the style string to apply to this feature, cannot be NULL.

References CPLStrdup().

Referenced by Clone(), OGRStyleMgr::SetFeatureStyleString(), and SetFrom().

12.60.3.51 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

pszString	the style string to apply to this feature, cannot be NULL.

12.60.3.52 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 ownship 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.

References GetGeomFieldCount().

Referenced by OGR_F_GetGeometryRef(), and OGR_F_GetGeomFieldRef().

12.60.3.53 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

iField	the field to unset.

References CSLDestroy(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTInteger64List, OFTIntegerList, OFTRealList, OFTString, and OFTStringList.

Referenced by OGRGenSQLResultsLayer::GetFeature(), and SetFieldsFrom().

12.60.3.54 int OGRFeature::Validate (int *nValidateFlags,* int *bEmitError*)

Validate that a feature meets constraints of its schema.

The scope of test is specified with the nValidateFlags parameter.

Regarding OGR_F_VAL_WIDTH, the test is done assuming the string width must be interpreted as the number of UTF-8 characters. Some drivers might interpret the width as the number of bytes instead. So this test is rather conservative (if it fails, then it will fail for all interpretations).

This method is the same as the C function **OGR_F_Validate()** (p. **??**).

Parameters

nValidateFlags	OGR_F_VAL_ALL or combination of OGR_F_VAL_NULL, OGR_F_VAL_GEOM_TYPE, O↩ GR_F_VAL_WIDTH and OGR_F_VAL_ALLOW_NULL_WHEN_DEFAULT with '|' operator
bEmitError	TRUE if a **CPLError()** (p. **??**) must be emitted when a check fails

Returns

TRUE if all enabled validation tests pass.

Since

GDAL 2.0

References CPLError(), CPLIsUTF8(), CPLStrlenUTF8(), OGRFieldDefn::GetDefault(), GetFieldAsString(), O↩ GRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRGeometry::getGeometryType(), O↩ GRFeatureDefn::GetGeomFieldCount(), OGRFeatureDefn::GetGeomFieldDefn(), GetGeomFieldRef(), OGR↩ FieldDefn::GetNameRef(), OGRGeomFieldDefn::GetNameRef(), OGRFieldDefn::GetType(), OGRGeomField↩ Defn::GetType(), OGRFieldDefn::GetWidth(), IsFieldSet(), OGRFieldDefn::IsNullable(), OGRGeomFieldDefn::Is↩ Nullable(), OFTString, OGR_F_VAL_ALLOW_NULL_WHEN_DEFAULT, OGR_F_VAL_GEOM_TYPE, OGR_F_↩ VAL_NULL, OGR_F_VAL_WIDTH, OGRGeometryTypeToName(), wkbHasZ, wkbSetZ, and wkbUnknown.

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

- **ogr_feature.h**
- ogrfeature.cpp

12.61 OGRFeatureDefn Class Reference

`#include <ogr_feature.h>`

Public Member Functions

- **OGRFeatureDefn** (const char *pszName=NULL)

  Constructor.
- virtual const char * **GetName** ()

  *Get name of this **OGRFeatureDefn** (p. **??**).*
- virtual int **GetFieldCount** ()

  Fetch number of fields on this feature.
- virtual **OGRFieldDefn** * **GetFieldDefn** (int i)

  Fetch field definition.
- virtual int **GetFieldIndex** (const char *)

  Find field by name.
- virtual void **AddFieldDefn** (**OGRFieldDefn** *)

  Add a new field definition.
- virtual OGRErr **DeleteFieldDefn** (int iField)

  Delete an existing field definition.
- virtual OGRErr **ReorderFieldDefns** (int *panMap)

  Reorder the field definitions in the array of the feature definition.
- virtual int **GetGeomFieldCount** ()

  Fetch number of geometry fields on this feature.
- virtual **OGRGeomFieldDefn** * **GetGeomFieldDefn** (int i)

  Fetch geometry field definition.
- virtual int **GetGeomFieldIndex** (const char *)

  Find geometry field by name.
- virtual void **AddGeomFieldDefn** (**OGRGeomFieldDefn** *, int bCopy=TRUE)

  Add a new geometry field definition.
- virtual OGRErr **DeleteGeomFieldDefn** (int iGeomField)

  Delete an existing geometry field definition.
- virtual **OGRwkbGeometryType GetGeomType** ()

  Fetch the geometry base type.
- virtual void **SetGeomType** (**OGRwkbGeometryType**)

  Assign the base geometry type for this layer.
- virtual **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.
- virtual int **IsGeometryIgnored** ()

  Determine whether the geometry can be omitted when fetching features.
- virtual void **SetGeometryIgnored** (int bIgnore)

  Set whether the geometry can be omitted when fetching features.

- virtual int **IsStyleIgnored** ()

  Determine whether the style can be omitted when fetching features.
- virtual void **SetStyleIgnored** (int bIgnore)

  Set whether the style can be omitted when fetching features.
- virtual int **IsSame** (**OGRFeatureDefn** *poOtherFeatureDefn)

  Test if the feature definition is identical to the other one.

12.61.1 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 and potentially other metadata.

It is essentially a collection of field descriptions (**OGRFieldDefn** (p. **??**) class). Starting with GDAL 1.11, in addition to attribute fields, it can also contain multiple geometry fields (**OGRGeomFieldDefn** (p. **??**) class).

It is reasonable for different translators to derive classes from **OGRFeatureDefn** (p. **??**) with additional translator specific information.

12.61.2 Constructor & Destructor Documentation

12.61.2.1 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

pszName	the name to be assigned to this layer/class. It does not need to be unique.

References CPLMalloc(), CPLStrdup(), and wkbUnknown.

Referenced by Clone().

12.61.3 Member Function Documentation

12.61.3.1 void OGRFeatureDefn::AddFieldDefn (OGRFieldDefn * *poNewDefn*) [virtual]

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 existance 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

poNewDefn	the definition of the new field.

References CPLRealloc(), and GetFieldCount().

Referenced by Clone(), OGRLayerWithTransaction::CreateField(), and OGRUnionLayer::GetLayerDefn().

12.61.3.2 void OGRFeatureDefn::AddGeomFieldDefn (OGRGeomFieldDefn * *poNewDefn*, int *bCopy* = TRUE) [virtual]

Add a new geometry field definition.

To add a new geometry field definition to a layer definition, do not use this function directly, but use **OGRLayer::CreateGeomField()** (p. **??**) instead.

This method does an internal copy of the passed geometry field definition, unless bCopy is set to FALSE (in which case it takes ownership of the field definition.

This method should only be called while there are no **OGRFeature** (p. **??**) objects in existance based on this **OGRFeatureDefn** (p. **??**). The **OGRGeomFieldDefn** (p. **??**) passed in is copied, and remains the responsibility of the caller.

This method is the same as the C function **OGR_FD_AddGeomFieldDefn()** (p. **??**).

Parameters

poNewDefn	the definition of the new geometry field.
bCopy	whether poNewDefn should be copied.

Since

GDAL 1.11

References CPLRealloc(), and GetGeomFieldCount().

Referenced by Clone(), OGRLayerWithTransaction::CreateGeomField(), OGRUnionLayer::GetLayerDefn(), and SetGeomType().

12.61.3.3 OGRFeatureDefn * OGRFeatureDefn::Clone ( ) [virtual]

Create a copy of this feature definition.

Creates a deep copy of the feature definition.

Returns

the copy.

References AddFieldDefn(), AddGeomFieldDefn(), DeleteGeomFieldDefn(), GetFieldCount(), GetFieldDefn(), GetGeomFieldCount(), GetGeomFieldDefn(), GetName(), and OGRFeatureDefn().

Referenced by OGRLayerWithTransaction::GetLayerDefn(), and OGRWarpedLayer::GetLayerDefn().

12.61.3.4 OGRErr OGRFeatureDefn::DeleteFieldDefn (int *iField*) [virtual]

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 existance based on this **OGRFeatureDefn** (p. **??**).

This method is the same as the C function **OGR_FD_DeleteFieldDefn()** (p. **??**).

Parameters

iField	the index of the field defintion.

Returns

OGRERR_NONE in case of success.

Since

OGR 1.9.0

References GetFieldCount().

Referenced by OGRLayerWithTransaction::DeleteField(), and OGRUnionLayer::GetLayerDefn().

12.61.3.5 OGRErr OGRFeatureDefn::DeleteGeomFieldDefn (int *iGeomField*) [virtual]

Delete an existing geometry field definition.

To delete an existing field definition from a layer definition, do not use this function directly, but use OGRLayer::↩ DeleteGeomField() instead.

This method should only be called while there are no **OGRFeature** (p. **??**) objects in existance based on this **OG↩ RFeatureDefn** (p. **??**).

This method is the same as the C function **OGR_FD_DeleteGeomFieldDefn()** (p. **??**).

Parameters

iGeomField	the index of the geometry field defintion.

Returns

OGRERR_NONE in case of success.

Since

GDAL 1.11

References GetGeomFieldCount().

Referenced by Clone(), OGRUnionLayer::GetLayerDefn(), and SetGeomType().

12.61.3.6 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().

12.61.3.7 int OGRFeatureDefn::GetFieldCount ( ) [virtual]

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 AddFieldDefn(), Clone(), OGRFeature::Clone(), OGRLayerWithTransaction::CreateField(), DeleteFieldDefn(), OGRFeature::Equal(), OGRFeature::FillUnsetWithDefault(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsInteger64(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldCount(), GetFieldDefn(), GetFieldIndex(), OGRUnionLayer::GetLayerDefn(), OGRFeature::IsFieldSet(), IsSame(), OGRFeature::OGRFeature(), OGRLayer::ReorderField(), ReorderFieldDefns(), OGRLayer::SetIgnoredFields(), and OGRFeature::Validate().

12.61.3.8 OGRFieldDefn * OGRFeatureDefn::GetFieldDefn (int *iField*) [virtual]

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

iField	the field to fetch, between 0 and **GetFieldCount()** (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(), and GetFieldCount().

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), Clone(), OGRLayerWithTransaction::CreateField(), OGRFeature::DumpReadable(), OGRFeature::Equal(), OGRFeature::FillUnsetWithDefault(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsInteger64(), OGRFeature::GetFieldAsInteger64List(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRFeature::GetFieldDefnRef(), GetFieldIndex(), OGRUnionLayer::GetLayerDefn(), IsSame(), OGRFeature::SetField(), OGRLayer::SetIgnoredFields(), OGRFeature::UnsetField(), and OGRFeature::Validate().

12.61.3.9 int OGRFeatureDefn::GetFieldIndex (const char * *pszFieldName*) [virtual]

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

pszFieldName	the field name to search for.

Returns

the field index, or -1 if no match found.

References GetFieldCount(), and GetFieldDefn().

Referenced by OGRLayer::FindFieldIndex(), OGRFeature::GetFieldIndex(), OGRUnionLayer::GetLayerDefn(), and OGRLayer::SetIgnoredFields().

12.61.3.10 int OGRFeatureDefn::GetGeomFieldCount ( ) [virtual]

Fetch number of geometry fields on this feature.

This method is the same as the C function **OGR_FD_GetGeomFieldCount()** (p. **??**).

Returns

count of geometry fields.

Since

GDAL 1.11

Referenced by AddGeomFieldDefn(), Clone(), OGRFeature::Clone(), OGRLayerWithTransaction::CreateGeom↩ Field(), DeleteGeomFieldDefn(), OGRGenSQLResultsLayer::GetExtent(), OGRFeature::GetGeomFieldCount(), GetGeomFieldDefn(), GetGeomFieldIndex(), GetGeomType(), OGRUnionLayer::GetLayerDefn(), IsGeometry↩ Ignored(), IsSame(), OGRFeature::OGRFeature(), SetGeometryIgnored(), SetGeomType(), and OGRFeature::↩ Validate().

12.61.3.11 OGRGeomFieldDefn * OGRFeatureDefn::GetGeomFieldDefn (int *iGeomField*) [virtual]

Fetch geometry field definition.

This method is the same as the C function **OGR_FD_GetGeomFieldDefn()** (p. **??**).

Parameters

iGeomField	the geometry field to fetch, between 0 and **GetGeomFieldCount()** (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.

Since

GDAL 1.11

References CPLError(), and GetGeomFieldCount().

Referenced by Clone(), OGRLayerWithTransaction::CreateGeomField(), OGRFeature::DumpReadable(), O↩ GRGenSQLResultsLayer::GetExtent(), OGRLayer::GetGeometryColumn(), OGRFeature::GetGeomFieldDefn↩ Ref(), GetGeomFieldIndex(), GetGeomType(), OGRUnionLayer::GetLayerDefn(), OGRLayer::GetSpatialRef(), IsGeometryIgnored(), IsSame(), SetGeometryIgnored(), SetGeomType(), OGRLayer::SetIgnoredFields(), and OGRFeature::Validate().

12.61.3.12 int OGRFeatureDefn::GetGeomFieldIndex (const char * *pszGeomFieldName*) [virtual]

Find geometry field by name.

The geometry field index of the first geometry field matching the passed field name (case insensitively) is returned.

This method is the same as the C function **OGR_FD_GetGeomFieldIndex()** (p. **??**).

Parameters

pszGeomFieldName	the geometry field name to search for.

Returns

the geometry field index, or -1 if no match found.

References GetGeomFieldCount(), and GetGeomFieldDefn().

Referenced by OGRUnionLayer::GetExtent(), OGRFeature::GetGeomFieldIndex(), OGRUnionLayer::GetLayerDefn(), and OGRLayer::SetIgnoredFields().

12.61.3.13 OGRwkbGeometryType OGRFeatureDefn::GetGeomType ( ) [virtual]

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.

Starting with GDAL 1.11, this method returns GetGeomFieldDefn(0)->GetType().

This method is the same as the C function **OGR_FD_GetGeomType()** (p. **??**).

Returns

the base type for all geometry related to this definition.

References CPLGetConfigOption(), CSLTestBoolean(), GetGeomFieldCount(), GetGeomFieldDefn(), OGRGeomFieldDefn::GetType(), wkbNone, and wkbUnknown.

Referenced by OGRLayer::GetGeomType().

12.61.3.14 const char * OGRFeatureDefn::GetName ( ) [virtual]

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(), OGRFeature::DumpReadable(), OGRLayer::GetName(), IsSame(), OGR_Dr_CopyDataSource(), and OGRFeature::SetField().

12.61.3.15 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.

12.61.3.16 int OGRFeatureDefn::IsGeometryIgnored ( ) [virtual]

Determine whether the geometry can be omitted when fetching features.

This method is the same as the C function **OGR_FD_IsGeometryIgnored()** (p. **??**).

Starting with GDAL 1.11, this method returns GetGeomFieldDefn(0)->IsIgnored().

Returns

ignore state

References GetGeomFieldCount(), GetGeomFieldDefn(), and OGRGeomFieldDefn::IsIgnored().

12.61.3.17 int OGRFeatureDefn::IsSame (OGRFeatureDefn * *poOtherFeatureDefn*) [virtual]

Test if the feature definition is identical to the other one.

Parameters

poOther↩ FeatureDefn	the other feature definition to compare to.

Returns

TRUE if the feature definition is identical to the other one.

References GetFieldCount(), GetFieldDefn(), GetGeomFieldCount(), GetGeomFieldDefn(), GetName(), OGR↩ FieldDefn::IsSame(), and OGRGeomFieldDefn::IsSame().

12.61.3.18 int OGRFeatureDefn::IsStyleIgnored ( ) [inline], [virtual]

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

12.61.3.19 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 OGRLayerWithTransaction::GetLayerDefn(), OGRWarpedLayer::GetLayerDefn(), OGRProxied↩ Layer::GetLayerDefn(), OGRUnionLayer::GetLayerDefn(), and OGRFeature::OGRFeature().

12.61.3.20 OGRErr OGRFeatureDefn::ReorderFieldDefns (int * *panMap*) [virtual]

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 existance based on this **OGRFeatureDefn** (p. **??**).

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

Parameters

panMap	an array of **GetFieldCount()** (p. **??**) elements which is a permutation of [0, **GetFieldCount()** (p. **??**)-1]. panMap is such that, for each field definition at position i after reordering, its position before reordering was panMap[i].

Returns

OGRERR_NONE in case of success.

Since

OGR 1.9.0

References CPLMalloc(), and GetFieldCount().

Referenced by OGRLayerWithTransaction::ReorderFields().

12.61.3.21 void OGRFeatureDefn::SetGeometryIgnored (int *bIgnore*) [virtual]

Set whether the geometry can be omitted when fetching features.

This method is the same as the C function **OGR_FD_SetGeometryIgnored()** (p. **??**).

Starting with GDAL 1.11, this method calls GetGeomFieldDefn(0)->SetIgnored().

Parameters

bIgnore	ignore state

References GetGeomFieldCount(), GetGeomFieldDefn(), and OGRGeomFieldDefn::SetIgnored().

Referenced by OGRLayer::SetIgnoredFields().

12.61.3.22 void OGRFeatureDefn::SetGeomType (OGRwkbGeometryType *eNewType*) [virtual]

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. **??**).

Starting with GDAL 1.11, this method calls GetGeomFieldDefn(0)->SetType().

Parameters

eNewType	the new type to assign.

References AddGeomFieldDefn(), DeleteGeomFieldDefn(), GetGeomFieldCount(), GetGeomFieldDefn(), OGRGeomFieldDefn::SetType(), and wkbNone.

Referenced by OGRUnionLayer::GetLayerDefn().

12.61.3.23 void OGRFeatureDefn::SetStyleIgnored (int *bIgnore*) [inline],[virtual]

Set whether the style can be omitted when fetching features.

This method is the same as the C function **OGR_FD_SetStyleIgnored()** (p. **??**).

Parameters

bIgnore	ignore state

Referenced by OGRLayer::SetIgnoredFields().

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

- **ogr_feature.h**
- ogrfeaturedefn.cpp

12.62 OGRFeatureQuery Class Reference

Public Member Functions

- char ** **GetUsedFields** ()

12.62.1 Member Function Documentation

12.62.1.1 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

12.63 OGRField Union Reference

`#include <ogr_core.h>`

12.63.1 Detailed Description

OGRFeature (p. **??**) field attribute value union.

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

- **ogr_core.h**

12.64 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 **OGRFeatureDefn** (p. **??**).*
- **OGRFieldSubType GetSubType** ()
  Fetch subtype of this field.
- void **SetSubType** (**OGRFieldSubType** eSubTypeIn)
  *Set the subtype of this field. This should never be done to an **OGRFieldDefn** (p. **??**) that is already part of an **OGRFeatureDefn** (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**=OJUndefined)
  Set defining parameters for a field in one call.
- void **SetDefault** (const char *)
  Set default field value.
- const char * **GetDefault** () const
  Get default field value.
- int **IsDefaultDriverSpecific** () const
  Returns whether the default value is driver specific.
- int **IsIgnored** ()
  Return whether this field should be omitted when fetching features.
- void **SetIgnored** (int bIgnoreIn)
  Set whether this field should be omitted when fetching features.
- int **IsNullable** () const
  Return whether this field can receive null values.
- void **SetNullable** (int bNullableIn)
  Set whether this field can receive null values.
- 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.
- static const char * **GetFieldSubTypeName** (**OGRFieldSubType**)

  Fetch human readable name for a field subtype.

12.64.1 Detailed Description

Definition of an attribute of an **OGRFeatureDefn** (p. **??**). A field is described by :

- a name. See **SetName()** (p. **??**) / **GetNameRef()** (p. **??**)
- a type: OFTString, OFTInteger, OFTReal, ... See **SetType()** (p. **??**) / **GetType()** (p. **??**)
- a subtype (optional): OFSTBoolean, ... See **SetSubType()** (p. **??**) / **GetSubType()** (p. **??**)
- a width (optional): maximal number of characters. See **SetWidth()** (p. **??**) / **GetWidth()** (p. **??**)
- a precision (optional): number of digits after decimal point. See **SetPrecision()** (p. **??**) / **GetPrecision()** (p. **??**)
- a NOT NULL constraint (optional). See **SetNullable()** (p. **??**) / **IsNullable()** (p. **??**)
- a default value (optional). See **SetDefault()** (p. **??**) / **GetDefault()** (p. **??**)
- a boolean to indicate whether it should be ignored when retrieving features. See **SetIgnored()** (p. **??**) / **IsIgnored()** (p. **??**)

12.64.2 Constructor & Destructor Documentation

12.64.2.1 OGRFieldDefn::OGRFieldDefn (const char * *pszNameIn*, OGRFieldType *eTypeIn*)

Constructor.

By default, fields have no width, precision, are nullable and not ignored.

Parameters

pszNameIn	the name of the new field.
eTypeIn	the type of the new field.

12.64.2.2 OGRFieldDefn::OGRFieldDefn (OGRFieldDefn * *poPrototype*)

Constructor.

Create by cloning an existing field definition.

Parameters

poPrototype	the field definition to clone.

References GetDefault(), GetJustify(), GetNameRef(), GetPrecision(), GetSubType(), GetType(), GetWidth(), IsNullable(), SetDefault(), SetJustify(), SetNullable(), SetPrecision(), SetSubType(), and SetWidth().

12.64.3 Member Function Documentation

12.64.3.1 const char * OGRFieldDefn::GetDefault ( ) const

Get default field value.

This function is the same as the C function **OGR_Fld_GetDefault()** (p. **??**).

Returns

default field value or NULL.

Since

GDAL 2.0

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), OGRFeature::FillUnsetWithDefault(), OGRFieldDefn(), and OGRFeature::Validate().

12.64.3.2 const char * OGRFieldDefn::GetFieldSubTypeName (OGRFieldSubType *eSubType*) [static]

Fetch human readable name for a field subtype.

This static method is the same as the C function **OGR_GetFieldSubTypeName()** (p. **??**).

Parameters

eSubType	the field subtype to get name for.

Returns

pointer to an internal static name string. It should not be modified or freed.

Since

GDAL 2.0

References OFSTBoolean, OFSTFloat32, OFSTInt16, and OFSTNone.

Referenced by OGRFeature::DumpReadable(), and OGR_GetFieldSubTypeName().

12.64.3.3 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

eType	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, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

Referenced by OGRFeature::DumpReadable(), and OGR_GetFieldTypeName().

12.64.3.4 OGRJustification OGRFieldDefn::GetJustify ( ) [inline]

Get the justification for this field.

Note: no driver is know to use the concept of field justification.

This method is the same as the C function **OGR_Fld_GetJustify()** (p. **??**).

Returns

the justification.

Referenced by OGRFieldDefn().

12.64.3.5 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 OGRLayerWithTransaction::AlterFieldDefn(), OGRFeature::DumpReadable(), OGRUnionLayer::↩ GetLayerDefn(), OGRFieldDefn(), OGRFeature::SetField(), OGRFeature::SetFrom(), and OGRFeature::Validate().

12.64.3.6 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 OGRLayerWithTransaction::AlterFieldDefn(), OGRFeature::GetFieldAsString(), and OGRField↩ Defn().

12.64.3.7 OGRFieldSubType OGRFieldDefn::GetSubType ( ) [inline]

Fetch subtype of this field.

This method is the same as the C function **OGR_Fld_GetSubType()** (p. **??**).

Returns

field subtype.

Since

GDAL 2.0

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), OGRFeature::DumpReadable(), OGRFieldDefn(), and OGRFeature::SetField().

12.64.3.8 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 OGRLayerWithTransaction::AlterFieldDefn(), OGRFeature::DumpReadable(), OGRFeature::Equal(), OGRFeature::FillUnsetWithDefault(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsInteger64(), OGRFeature::GetFieldAsInteger64List(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRFieldDefn(), OGRFeature::SetField(), OGRFeature::SetFieldsFrom(), OGRFeature::UnsetField(), and OGRFeature::Validate().

12.64.3.9 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 OGRLayerWithTransaction::AlterFieldDefn(), OGRFeature::GetFieldAsString(), OGRFieldDefn(), and OGRFeature::Validate().

12.64.3.10 int OGRFieldDefn::IsDefaultDriverSpecific ( ) const

Returns whether the default value is driver specific.

Driver specific default values are those that are *not* NULL, a numeric value, a litteral value enclosed between single quote characters, CURRENT_TIMESTAMP, CURRENT_TIME, CURRENT_DATE or datetime literal value.

This method is the same as the C function **OGR_Fld_IsDefaultDriverSpecific()** (p. **??**).

Returns

TRUE if the default value is driver specific.

Since

GDAL 2.0

References CPLStrtod().

12.64.3.11 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

12.64.3.12 int OGRFieldDefn::IsNullable ( ) const [inline]

Return whether this field can receive null values.

By default, fields are nullable.

Even if this method returns FALSE (i.e not-nullable field), it doesn't mean that **OGRFeature::IsFieldSet()** (p. **??**) will necessary return TRUE, as fields can be temporary unset and null/not-null validation is usually done when **OGRLayer::CreateFeature()** (p. **??**)/SetFeature() is called.

This method is the same as the C function **OGR_Fld_IsNullable()** (p. **??**).

Returns

TRUE if the field is authorized to be null.

Since

GDAL 2.0

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), OGRFeature::FillUnsetWithDefault(), OGRFieldDefn(), and OGRFeature::Validate().

12.64.3.13 int OGRFieldDefn::IsSame (const OGRFieldDefn * *poOtherFieldDefn*) const

Test if the field definition is identical to the other one.

Parameters

poOtherFieldDefn	the other field definition to compare to.

Returns

TRUE if the field definition is identical to the other one.

Referenced by OGRFeatureDefn::IsSame().

12.64.3.14 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

pszNameIn	the new name to assign.
eTypeIn	the new type (one of the OFT values like OFTInteger).
nWidthIn	the preferred formatting width. Defaults to zero indicating undefined.
nPrecisionIn	number of decimals places for formatting, defaults to zero indicating undefined.
eJustifyIn	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References SetJustify(), SetName(), SetPrecision(), SetType(), and SetWidth().

12.64.3.15 void OGRFieldDefn::SetDefault (const char * *pszDefaultIn*)

Set default field value.

The default field value is taken into account by drivers (generally those with a SQL interface) that support it at field creation time. OGR will generally not automatically set the default field value to null fields by itself when calling

OGRFeature::CreateFeature() (p. **??**) / OGRFeature::SetFeature(), but will let the low-level layers to do the job. So retrieving the feature from the layer is recommended.

The accepted values are NULL, a numeric value, a litteral value enclosed between single quote characters (and inner single quote characters escaped by repetition of the single quote character), CURRENT_TIMESTAMP, C↩ URRENT_TIME, CURRENT_DATE or a driver specific expression (that might be ignored by other drivers). For a datetime literal value, format should be 'YYYY/MM/DD HH:MM:SS[.sss]' (considered as UTC time).

Drivers that support writing DEFAULT clauses will advertize the GDAL_DCAP_DEFAULT_FIELDS driver metadata item.

This function is the same as the C function **OGR_Fld_SetDefault()** (p. **??**).

Parameters

pszDefault	new default field value or NULL pointer.

Since

GDAL 2.0

References CPLError(), and CPLStrdup().

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), and OGRFieldDefn().

12.64.3.16 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

ignore	ignore state

Referenced by OGRLayer::SetIgnoredFields().

12.64.3.17 void OGRFieldDefn::SetJustify (OGRJustification *eJustify*) [inline]

Set the justification for this field.

Note: no driver is know to use the concept of field justification.

This method is the same as the C function **OGR_Fld_SetJustify()** (p. **??**).

Parameters

eJustify	the new justification.

Referenced by OGRFieldDefn(), and Set().

12.64.3.18 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

pszNameIn	the new name to apply.

References CPLStrdup().

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), and Set().

12.64.3.19 void OGRFieldDefn::SetNullable (int *bNullableIn*) [inline]

Set whether this field can receive null values.

By default, fields are nullable, so this method is generally called with FALSE to set a not-null constraint.

Drivers that support writing not-null constraint will advertize the GDAL_DCAP_NOTNULL_FIELDS driver metadata item.

This method is the same as the C function **OGR_Fld_SetNullable()** (p. **??**).

Parameters

bNullableIn	FALSE if the field must have a not-null constraint.

Since

GDAL 2.0

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), and OGRFieldDefn().

12.64.3.20 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

nPrecision	the new precision.

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), OGRFieldDefn(), and Set().

12.64.3.21 void OGRFieldDefn::SetSubType (OGRFieldSubType *eSubTypeIn*)

Set the subtype of this field. This should never be done to an **OGRFieldDefn** (p. **??**) that is already part of an **OGRFeatureDefn** (p. **??**).

This method is the same as the C function **OGR_Fld_SetSubType()** (p. **??**).

Parameters

eSubType	the new field subtype.

Since

GDAL 2.0

References CPLError(), OFSTNone, and OGR_AreTypeSubTypeCompatible().

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), and OGRFieldDefn().

12.64.3.22 void OGRFieldDefn::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. **??**).

This method is the same as the C function **OGR_Fld_SetType()** (p. **??**).

Parameters

eType	the new field type.

References CPLError(), OFSTNone, and OGR_AreTypeSubTypeCompatible().

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), and Set().

12.64.3.23 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

nWidth	the new width.

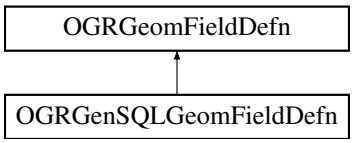
Referenced by OGRLayerWithTransaction::AlterFieldDefn(), OGRFieldDefn(), and Set().

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

- **ogr_feature.h**
- ogrfielddefn.cpp

12.65 OGRGenSQLGeomFieldDefn Class Reference

Inheritance diagram for OGRGenSQLGeomFieldDefn:



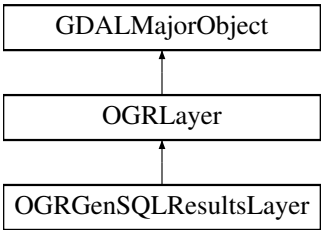
Additional Inherited Members

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

- ogr_gensql.cpp

12.66 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** (GIntBig nIndex)

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

  Fetch a feature by its identifier.
- virtual **OGRFeatureDefn** ∗ **GetLayerDefn** ()

  Fetch the schema information for this layer.
- virtual GIntBig **GetFeatureCount** (int bForce=TRUE)

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

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

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

  Test if this layer supported the named capability.
- virtual void **SetSpatialFilter** (**OGRGeometry** ∗poGeom)

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

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

  Set a new attribute query.

Additional Inherited Members

12.66.1 Member Function Documentation

12.66.1.1 virtual OGRErr OGRGenSQLResultsLayer::GetExtent (OGREnvelope ∗ *psExtent,* int *bForce* = TRUE) [inline], [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

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. **??**).

References GetExtent().

Referenced by GetExtent().

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

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

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

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (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.

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

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

Parameters

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. **??**).

References CPLError(), OGRLayer::GetExtent(), OGRFeatureDefn::GetGeomFieldCount(), OGRFeatureDefn::GetGeomFieldDefn(), GetLayerDefn(), OGRGeomFieldDefn::GetType(), and wkbNone.

12.66.1.3 OGRFeature * OGRGenSQLResultsLayer::GetFeature (GIntBig *nFID*) [virtual]

Fetch a feature by its identifier.

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

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (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 (with **GetNextFeature()** (p. **??**)) 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

nFID	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().

12.66.1.4 GIntBig 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. **??**).

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

Parameters

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

Returns

feature count, -1 if count not known.

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetFeatureCount().

12.66.1.5 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. **??**).

Referenced by GetExtent().

12.66.1.6 OGRFeature * OGRGenSQLResultsLayer::GetNextFeature ( ) [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity 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.

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

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(), OGRFeature::GetGeomFieldRef(), and OGRLayer::GetNextFeature().

12.66.1.7 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. **??**).

12.66.1.8 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. **??**).

12.66.1.9 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 interprete the WHERE clause in which case the capabilities will be broader than those of OGR SQL.

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

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

Parameters

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

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetAttributeFilter().

12.66.1.10 OGRErr OGRGenSQLResultsLayer::SetNextByIndex (GIntBig *nIndex*) [virtual]

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

This method allows positioning of a layer such that the **GetNextFeature()** (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

nIndex	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().

12.66.1.11 virtual void OGRGenSQLResultsLayer::SetSpatialFilter (OGRGeometry * *poFilter*) [inline], [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

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

Reimplemented from **OGRLayer** (p. **??**).

References SetSpatialFilter().

Referenced by SetSpatialFilter().

12.66.1.12 void OGRGenSQLResultsLayer::SetSpatialFilter (int *iGeomField,* OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**GetSpatialRef()** (p. **??**)). In the future this may be generalized.

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

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

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

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetSpatialFilter().

12.66.1.13 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.
- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using **CreateGeomField()** (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.
- **OLCCurveGeometries** / "CurveGeometries": TRUE if this layer supports writing curve geometries or may return such geometries. (GDAL 2.0).

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

Parameters

pszCap	the name of the capability to test.

Returns

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

Implements **OGRLayer** (p. **??**).

References OGRLayer::TestCapability().

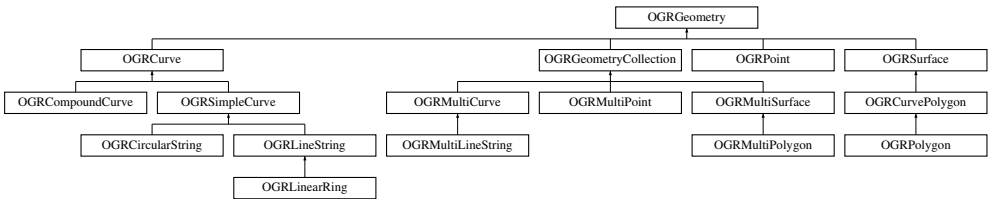
The documentation for this class was generated from the following files:

- ogr_gensql.h
- ogr_gensql.cpp

12.67 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, **OGRwkbVariant**=**wkbVariantOldOgc**)=0

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *, **OGRwkbVariant**=**wkbVariantOldOgc**) 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, **OGRwkbVariant**=**wkbVariantOldOgc**) const =0

  Convert a geometry into well known text format.
- virtual **OGRwkbGeometryType getGeometryType** () const =0

  Fetch geometry type.
- **OGRwkbGeometryType getIsoGeometryType** () const

  Get the geometry type that conforms with ISO SQL/MM Part3.
- virtual const char * **getGeometryName** () const =0

  Fetch WKT name for geometry type.
- virtual void **dumpReadable** (FILE *, const char *=NULL, char **papszOptions=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 *papszOptions=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 OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.
- virtual **OGRGeometry** * **getCurveGeometry** (const char *const *papszOptions=NULL) const

  Return curve version of this geometry.
- virtual **OGRGeometry** * **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char *const *papszOptions=NULL) const

  Return, possibly approximate, non-curve version of this geometry.
- 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** (const **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_DEPRECAT↩ ED("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.

Friends

- class **OGRCurveCollection**

12.67.1 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 06-103r4`)

In GDAL 2.0, the hierarchy of classes has been extended with `(working draft) ISO SQL/MM Part 3 (ISO/IEC 13249-3)` curve geometries : CIRCULARSTRING (**OGRCircularString** (p. **??**)), COMPOUNDC↩URVE (**OGRCompoundCurve** (p. **??**)), CURVEPOLYGON (**OGRCurvePolygon** (p. **??**)), MULTICURVE (**OGR↩MultiCurve** (p. **??**)) and MULTISURFACE (**OGRMultiSurface** (p. **??**)).

12.67.2 Member Function Documentation

12.67.2.1 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

poSR	new spatial reference system to apply.

References OGRSpatialReference::Reference(), and OGRSpatialReference::Release().

Referenced by Boundary(), Buffer(), OGRCurve::CastToCompoundCurve(), OGRPolygon::CastToCurvePolygon(), OGRCompoundCurve::CastToLinearRing(), OGRCurvePolygon::CastToPolygon(), Centroid(), OGRPoint::clone(), OGRSimpleCurve::clone(), OGRLinearRing::clone(), OGRCompoundCurve::clone(), OGRCurvePolygon::clone(), OGRGeometryCollection::clone(), ConvexHull(), OGRGeometryFactory::createFromWkb(), OGRGeometry↩Factory::createFromWkt(), OGRGeometryFactory::curveFromLineString(), OGRCurvePolygon::CurvePolyToPoly(), OGRCircularString::CurveToLine(), Difference(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::force↩ToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OG↩RGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRPolygon::getCurve↩Geometry(), OGRGeometryCollection::getCurveGeometry(), OGRGeometryCollection::getLinearGeometry(), OGRSimpleCurve::getSubLine(), Intersection(), OGR_G_PointOnSurface(), Polygonize(), Simplify(), Simplify↩PreserveTopology(), SymDifference(), OGRPoint::transform(), OGRSimpleCurve::transform(), OGRGeometry↩Collection::transform(), Union(), and UnionCascaded().

12.67.2.2 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().

12.67.2.3 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 nQuad↩ Segs parameter can be used to control how many segements 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

dfDist	the buffer distance to be applied. Should be expressed into the same unit as the coordinates of the geometry.
nQuadSegs	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(), CPLError(), and getSpatialReference().

12.67.2.4 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(), CPLError(), OGRPoint::empty(), getGeometryType(), getSpatialReference(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::IsEmpty(), OGRPoint::setX(), OGRPoint::setY(), wkbFlatten, and wkbPoint.

Referenced by OGR_G_Centroid().

12.67.2.5 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 **OGRGeometryCollection** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRLinearRing** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRCompoundCurve::addCurve(), OGRGeometryCollection::addGeometry(), OGRCurvePolygon::addRing(), OGRLayer::Clip(), OGRLayer::Erase(), OGRGeometryFactory::forceTo(), getCurveGeometry(), OGRLayer::GetFeature(), getLinearGeometry(), OGRLayer::Identity(), OGRFeature::SetGeomField(), OGRLayer::SymDifference(), OGRLayer::Union(), and OGRLayer::Update().

12.67.2.6 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. **??**).

Referenced by OGRPolygon::closeRings().

12.67.2.7 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

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if poOtherGeom contains this geometry, otherwise FALSE.

Reimplemented in **OGRCurvePolygon** (p. **??**).

References CPLError().

Referenced by OGRCurvePolygon::Contains(), and OGRPoint::Within().

12.67.2.8 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().

12.67.2.9 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

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if they are crossing, otherwise FALSE.

References CPLError().

12.67.2.10 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

poOtherGeom	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().

12.67.2.11 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

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if they are disjoint, otherwise FALSE.

References CPLError().

12.67.2.12 double OGRGeometry::Distance (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute distance between two geometries.

Returns the shortest distance between the two geometries. The distance is expressed into the same unit as the coordinates of the 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

poOtherGeom	the other geometry to compare against.

Returns

the distance between the geometries or -1 if an error occurs.

References CPLDebug(), and CPLError().

12.67.2.13 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

fp	the text file to write the geometry to.
pszPrefix	the prefix to put on each line of output.
papszOptions	NULL terminated list of options (may be NULL)

References CSLTestBoolean(), dumpReadable(), exportToWkt(), OGRCompoundCurve::getCurve(), OGRCurvePolygon::getExteriorRingCurve(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), getGeometryType(), OGRCurvePolygon::getInteriorRingCurve(), OGRCompoundCurve::getNumCurves(), OGRGeometryCollection::getNumGeometries(), OGRCurvePolygon::getNumInteriorRings(), OGRCurve::getNumPoints(), OGRSimpleCurve::getNumPoints(), wkbCircularString, wkbCircularStringZ, wkbCompoundCurve, wkbCompoundCurveZ, wkbCurvePolygon, wkbCurvePolygonZ, wkbFlatten, wkbGeometryCollection, wkbGeometryCollection25D, wkbLinearRing, wkbLineString, wkbLineString25D, wkbMultiCurve, wkbMultiCurveZ, wkbMultiLineString, wkbMultiLineString25D, wkbMultiPoint, wkbMultiPoint25D, wkbMultiPolygon, wkbMultiPolygon25D, wkbMultiSurface, wkbMultiSurfaceZ, wkbNone, wkbPoint, wkbPoint25D, wkbPolygon, wkbPolygon25D, and wkbUnknown.

Referenced by dumpReadable(), and OGRFeature::DumpReadable().

12.67.2.14 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. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

12.67.2.15 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. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRFeature::Equal(), and OGRGeometryCollection::Equals().

12.67.2.16 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

papszOptions	NULL-terminated list of options.

Returns

A GML fragment or NULL in case of error.

References OGR_G_ExportToGMLEx().

12.67.2.17 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().

12.67.2.18 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().

12.67.2.19 OGRErr OGRGeometry::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.

pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGR↩ Geometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Implemented in **OGRGeometryCollection** (p. **??**), **OGRPolygon** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGR↩ CompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), **OGRLinearRing** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryCollection::exportToWkb().

12.67.2.20 OGRErr OGRGeometry::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant =* wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Implemented in **OGRMultiLineString** (p. **??**), **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMulti↩ Polygon** (p. **??**), **OGRMultiSurface** (p. **??**), **OGRGeometryCollection** (p. **??**), **OGRPolygon** (p. **??**), **OGR↩ CurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by dumpReadable().

12.67.2.21 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. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

12.67.2.22 OGRGeometry * OGRGeometry::getBoundary ( ) const [virtual]

Compute boundary (deprecated)

Deprecated

See also

Boundary() (p. **??**)

References Boundary().

12.67.2.23 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.

Reimplemented in **OGRPoint** (p. **??**).

Referenced by OGRGeometryCollection::addGeometryDirectly(), OGRPolygon::CastToCurvePolygon(), OGRCurvePolygon::CastToPolygon(), OGRSimpleCurve::clone(), OGRCircularString::CurveToLine(), OGRSimpleCurve::exportToWkb(), OGRPolygon::exportToWkb(), OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRCircularString::getGeometryType(), OGRCompoundCurve::getGeometryType(), OGRGeometryCollection::getGeometryType(), OGRMultiSurface::getGeometryType(), OGRMultiPolygon::getGeometryType(), OGRMultiPoint::getGeometryType(), OGRMultiCurve::getGeometryType(), OGRMultiLineString::getGeometryType(), getIsoGeometryType(), OGRSimpleCurve::getPoint(), OGRSimpleCurve::getSubLine(), OGRSimpleCurve::importFromWkb(), OGRSimpleCurve::segmentize(), OGRSimpleCurve::setNumPoints(), OGRSimpleCurve::setPoint(), OGRSimpleCurve::setPoints(), OGRSimpleCurve::Value(), OGRCircularString::Value(), OGRSimpleCurve::WkbSize(), and OGRPolygon::WkbSize().

12.67.2.24 OGRGeometry * OGRGeometry::getCurveGeometry (const char *const * *papszOptions* = NULL) const [virtual]

Return curve version of this geometry.

Returns a geometry that has possibly CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by de-approximating curve geometries.

If the geometry has no curve portion, the returned geometry will be a clone of it.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getLinearGeometry()** (p. **??**).

This function is the same as C function **OGR_G_GetCurveGeometry()** (p. **??**).

Parameters

papszOptions	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented in **OGRGeometryCollection** (p. **??**), **OGRPolygon** (p. **??**), and **OGRLineString** (p. **??**).

References clone().

Referenced by OGRPolygon::getCurveGeometry(), and OGRGeometryCollection::getCurveGeometry().

12.67.2.25 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::getCoordinate↩ Dimension()** (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 **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMultiSurface** (p. **??**), **OGRGeometry↩ Collection** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryCollection::getDimension(), OGRLayer::Identity(), OGRLayer::Intersection(), and O↩ GRLayer::Union().

12.67.2.26 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

psEnvelope	the structure in which to place the results.

Implemented in **OGRGeometryCollection** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryCollection::getEnvelope(), OGRLayer::Intersection(), Intersects(), OGRGeometry↩ Factory::organizePolygons(), and OGRWarpedLayer::SetSpatialFilter().

12.67.2.27 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

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implemented in **OGRGeometryCollection** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

12.67.2.28 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. **??**), **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMultiPolygon** (p. **??**), **OGRMultiSurface** (p. **??**), **OGRGeometryCollection** (p. **??**), **OGRPolygon** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), **OGRLinearRing** (p. **??**), **OGRLineString** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by dumpReadable(), OGRSimpleCurve::exportToWkt(), and OGRFeature::GetFieldAsString().

12.67.2.29 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()** (p. **??**) 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. **??**), **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMultiPolygon** (p. **??**), **OGRMultiSurface** (p. **??**), **OGRGeometryCollection** (p. **??**), **OGRPolygon** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), **OGRLineString** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryCollection::addGeometryDirectly(), OGRCurve::CastToCompoundCurve(), Centroid(), OGRSimpleCurve::clone(), OGRCurvePolygon::Contains(), dumpReadable(), OGRPoint::Equals(), OGRSimpleCurve::Equals(), OGRCompoundCurve::Equals(), OGRCurvePolygon::Equals(), OGRGeometryCollection::Equals(), OGRSimpleCurve::exportToWkb(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRGeometryCollection::get_Area(), OGRGeometryCollection::get_Length(), OGRPolygon::getCurveGeometry(), getIsoGeometryType(), OGRPoint::Intersects(), OGRCurvePolygon::Intersects(), OGR_F_GetGeometryRef(), OGR_F_GetGeomFieldRef(), OGR_G_PointOnSurface(), OGRBuildPolygonFromEdges(), Polygonize(), OGRFeature::Validate(), and OGRPoint::Within().

12.67.2.30 OGRwkbGeometryType OGRGeometry::getIsoGeometryType ( ) const

Get the geometry type that conforms with ISO SQL/MM Part3.

Returns

the geometry type that conforms with ISO SQL/MM Part3

References getCoordinateDimension(), getGeometryType(), and wkbFlatten.

Referenced by OGRPoint::exportToWkb(), OGRSimpleCurve::exportToWkb(), OGRPolygon::exportToWkb(), and OGRGeometryCollection::exportToWkb().

12.67.2.31 OGRGeometry * OGRGeometry::getLinearGeometry (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return, possibly approximate, non-curve version of this geometry.

Returns a geometry that has no CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by approximating curve geometries.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getCurveGeometry()** (p. **??**).

This method is the same as the C function **OGR_G_GetLinearGeometry()** (p. **??**).

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings. See **OGRGeometryFactory::curveToLine↩ String()** (p. **??**) for valid options.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented in **OGRGeometryCollection** (p. **??**), **OGRPolygon** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGR↩ CompoundCurve** (p. **??**), and **OGRCircularString** (p. **??**).

References clone().

Referenced by OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGR↩ GeometryFactory::forceToMultiLineString(), OGRPolygon::getLinearGeometry(), and OGRGeometryCollection↩ ::getLinearGeometry().

12.67.2.32 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(), OGRCurve::CastToCompoundCurve(), OGRPolygon::CastToCurvePolygon(), OGRCompoundCurve::CastToLinearRing(), OGRCurvePolygon::CastToPolygon(), Centroid(), OGRPoint::clone(), OGRSimpleCurve::clone(), OGRLinearRing::clone(), OGRCompoundCurve::clone(), OGRCurvePolygon::clone(), OGRGeometryCollection::clone(), ConvexHull(), OGRGeometryFactory::curveFromLineString(), OGRCurve↩ Polygon::CurvePolyToPoly(), OGRCircularString::CurveToLine(), Difference(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometry↩ Factory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceTo↩ Polygon(), OGRPolygon::getCurveGeometry(), OGRGeometryCollection::getCurveGeometry(), OGRGeometry↩ Collection::getLinearGeometry(), OGRSimpleCurve::getSubLine(), Intersection(), OGR_G_PointOnSurface(), Polygonize(), Simplify(), SimplifyPreserveTopology(), SymDifference(), transformTo(), Union(), and UnionCascaded().

12.67.2.33 OGRBoolean OGRGeometry::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNonLinear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented in **OGRMultiLineString** (p. **??**), **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMultiPolygon** (p. **??**), **OGRMultiSurface** (p. **??**), **OGRGeometryCollection** (p. **??**), **OGRPolygon** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), and **OGRCircularString** (p. **??**).

Referenced by OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), and OGRGeometryCollection::getCurveGeometry().

12.67.2.34 OGRErr OGRGeometry::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

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. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), **OGRLinearRing** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryFactory::createFromWkb().

12.67.2.35 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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implemented in **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMultiSurface** (p. **??**), **OGRGeometry↩ Collection** (p. **??**), **OGRPolygon** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGR↩ CircularString** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryFactory::createFromWkt().

12.67.2.36 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

poOtherGeom	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().

12.67.2.37 OGRBoolean OGRGeometry::Intersects (const OGRGeometry * *poOtherGeom*) const [virtual]

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigerous 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

poOtherGeom	the other geometry to test against.

Returns

TRUE if the geometries intersect, otherwise FALSE.

Reimplemented in **OGRCurvePolygon** (p. **??**), and **OGRPoint** (p. **??**).

References getEnvelope().

Referenced by OGRPoint::Intersects(), and OGRCurvePolygon::Intersects().

12.67.2.38 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. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRCurve::CastToCompoundCurve(), OGRLayer::Clip(), OGRPoint::Equals(), OGRSimple↩ Curve::Equals(), OGRCurvePolygon::Equals(), OGRGeometryCollection::Equals(), OGRLayer::Erase(), OGR↩ GeometryFactory::forceTo(), OGRLayer::Identity(), OGRLayer::Intersection(), OGRLayer::SymDifference(), OGR↩ Layer::Union(), and OGRLayer::Update().

12.67.2.39 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.

12.67.2.40 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.

12.67.2.41 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.

Reimplemented in **OGRCircularString** (p. **??**).

Referenced by OGRCircularString::IsValid().

12.67.2.42 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

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if they are overlapping, otherwise FALSE.

References CPLError().

Referenced by OGRGeometryFactory::organizePolygons().

12.67.2.43 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 topogical 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(), wkbFlatten, wkbGeometryCollection, wkbLineString, and wkbMultiLineString.

12.67.2.44 void OGRGeometry::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Add intermediate vertices to a geometry.

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

dfMaxLength	the maximum distance between 2 points after segmentization

This method modifies the geometry to add intermediate vertices if necessary so that the maximum length between 2 consecutives vertices is lower than dfMaxLength.

Parameters

dfMaxLength	maximum length between 2 consecutives vertices.

Reimplemented in **OGRGeometryCollection** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRCircularString** (p. **??**), and **OGRSimpleCurve** (p. **??**).

12.67.2.45 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

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented in **OGRGeometryCollection** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryCollection::addGeometryDirectly(), and OGRGeometryCollection::setCoordinate↩ Dimension().

12.67.2.46 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 CPLE_NotSupported error.

Parameters

dTolerance	the distance tolerance for the simplification.

Returns

the simplified geometry or NULL if an error occurs.

Since

OGR 1.8.0

References assignSpatialReference(), CPLError(), and getSpatialReference().

12.67.2.47 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 CPLE_NotSupported error.

Parameters

dTolerance	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().

12.67.2.48 void OGRGeometry::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented in **OGRGeometryCollection** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

12.67.2.49 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 CPLE_NotSupported error.

Parameters

poOtherGeom	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().

12.67.2.50 OGRGeometry * OGRGeometry::SymmetricDifference (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute symmetric difference (deprecated)

Deprecated

See also

OGRGeometry::SymDifference() (p. **??**)

References SymDifference().

12.67.2.51 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

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if they are touching, otherwise FALSE.

References CPLError().

12.67.2.52 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 **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

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implemented in **OGRGeometryCollection** (p. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryCollection::transform(), and transformTo().

12.67.2.53 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

poSR	spatial reference system to transform to.

Returns

OGRERR_NONE on success, or an error code.

References getSpatialReference(), OGRCreateCoordinateTransformation(), and transform().

12.67.2.54 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

poOtherGeom	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().

12.67.2.55 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().

12.67.2.56 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

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if poOtherGeom is within this geometry, otherwise FALSE.

Reimplemented in **OGRPoint** (p. **??**).

References CPLError().

Referenced by OGRPoint::Within().

12.67.2.57 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. **??**), **OGRCurvePolygon** (p. **??**), **OGRCompoundCurve** (p. **??**), **OGRLinearRing** (p. **??**), **OGRSimpleCurve** (p. **??**), and **OGRPoint** (p. **??**).

Referenced by OGRGeometryCollection::exportToWkb(), and OGRGeometryCollection::WkbSize().

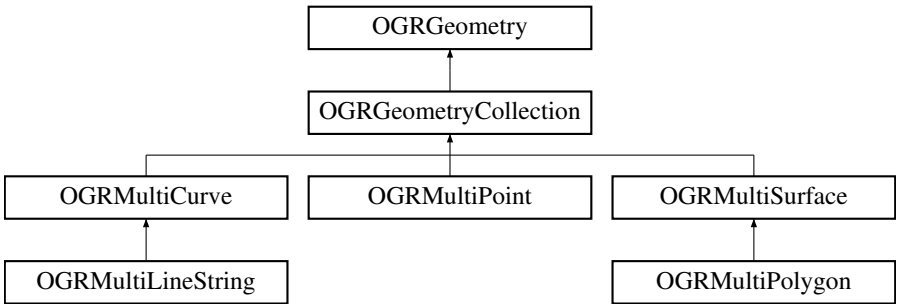
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrgeometry.cpp

12.68 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 OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.
- virtual **OGRGeometry** ∗ **getCurveGeometry** (const char ∗const ∗papszOptions=NULL) const

  Return curve version of this geometry.
- virtual **OGRGeometry** ∗ **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char ∗const ∗papszOptions=NULL) const

  Return, possibly approximate, non-curve version of this geometry.
- virtual int **WkbSize** () const

  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char ∗, int=-1, **OGRwkbVariant**=**wkbVariantOldOgc**)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char ∗, **OGRwkbVariant**=**wkbVariantOldOgc**) 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, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- virtual double **get_Length** () const

  Compute the length of a multicurve.

- 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.

12.68.1 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.

12.68.2 Member Function Documentation

12.68.2.1 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

poNewGeom	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().

12.68.2.2 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

poNewGeom	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 OGRGeometry::getCoordinateDimension(), OGRGeometry::getGeometryType(), OGRGeometry↩ ::setCoordinateDimension(), and setCoordinateDimension().

Referenced by addGeometry(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToMultiLine↩ String(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), getCurve↩ Geometry(), getLinearGeometry(), OGRMultiSurface::importFromWkt(), OGRMultiPoint::importFromWkt(), and OGRGeometryFactory::organizePolygons().

12.68.2.3 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. **??**).

References addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryFactory::createGeometry(), getGeometryType(), and OGRGeometry::getSpatialReference().

Referenced by getCurveGeometry().

12.68.2.4 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(), wkbFlatten, and wkbPolygon.

12.68.2.5 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. **??**).

12.68.2.6 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(), getNumGeometries(), OGRGeometry::IsEmpty(), and IsEmpty().

12.68.2.7 OGRErr OGRGeometryCollection::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGR↩ Geometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

References OGRGeometry::exportToWkb(), getGeometryType(), OGRGeometry::getIsoGeometryType(), wkb↩ Flatten, wkbHasZ, wkbMultiCurve, wkbMultiSurface, OGRGeometry::WkbSize(), wkbVariantIso, wkbVariantOld↩ Ogc, and wkbVariantPostGIS1.

12.68.2.8 OGRErr OGRGeometryCollection::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRMultiLineString** (p. **??**), **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMultiPolygon** (p. **??**), and **OGRMultiSurface** (p. **??**).

12.68.2.9 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. **??**).

12.68.2.10 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.

References OGRGeometry::getGeometryType(), OGR_GT_IsCurve(), OGR_GT_IsSubClassOf(), OGR_GT_IsSurface(), wkbFlatten, wkbGeometryCollection, and wkbMultiSurface.

12.68.2.11 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(), OGR_GT_IsCurve(), OGR_GT_IsSubClassOf(), wkbFlatten, wkbGeometryCollection, and wkbMultiCurve.

12.68.2.12 OGRGeometry * OGRGeometryCollection::getCurveGeometry (const char *const * *papszOptions* = NULL) const [virtual]

Return curve version of this geometry.

Returns a geometry that has possibly CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by de-approximating curve geometries.

If the geometry has no curve portion, the returned geometry will be a clone of it.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getLinearGeometry()** (p. **??**).

This function is the same as C function **OGR_G_GetCurveGeometry()** (p. **??**).

Parameters

papszOptions	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

References addGeometryDirectly(), OGRGeometry::assignSpatialReference(), clone(), OGRGeometryFactory::createGeometry(), OGRGeometry::getCurveGeometry(), getGeometryType(), OGRGeometry::getSpatialReference(), OGRGeometry::hasCurveGeometry(), and OGR_GT_GetCurve().

12.68.2.13 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 **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), and **OGRMultiSurface** (p. **??**).

References OGRGeometry::getDimension().

12.68.2.14 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

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. **??**).

12.68.2.15 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

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. **??**).

References OGRGeometry::getEnvelope(), and IsEmpty().

12.68.2.16 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. **??**), **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMultiPolygon** (p. **??**), and **OGRMultiSurface** (p. **??**).

12.68.2.17 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	the index of the geometry to fetch, between 0 and **getNumGeometries()** (p. **??**) - 1.

Returns

pointer to requested geometry.

Referenced by OGRGeometry::dumpReadable(), Equals(), OGRMultiPoint::exportToWkt(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRBuildPolygonFromEdges(), and OGRGeometry::Polygonize().

12.68.2.18 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()** (p. **??**) 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. **??**), **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMultiPolygon** (p. **??**), and **OGRMultiSurface** (p. **??**).

References OGRGeometry::getCoordinateDimension(), wkbGeometryCollection, and wkbGeometryCollection25D.

Referenced by clone(), closeRings(), Equals(), exportToWkb(), getCurveGeometry(), and getLinearGeometry().

12.68.2.19 OGRGeometry * OGRGeometryCollection::getLinearGeometry (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return, possibly approximate, non-curve version of this geometry.

Returns a geometry that has no CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by approximating curve geometries.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getCurveGeometry()** (p. **??**).

This method is the same as the C function **OGR_G_GetLinearGeometry()** (p. **??**).

Parameters

dfMaxAngleStepSizeDegrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings. See **OGRGeometryFactory::curveToLineString()** (p. **??**) for valid options.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

References addGeometryDirectly(), OGRGeometry::assignSpatialReference(), OGRGeometryFactory::create↩ Geometry(), getGeometryType(), OGRGeometry::getLinearGeometry(), OGRGeometry::getSpatialReference(), and OGR_GT_GetLinear().

Referenced by OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), O↩ GRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), and OGRMultiSurface::↩ PointOnSurface().

12.68.2.20 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 OGRGeometry::dumpReadable(), Equals(), OGRMultiPoint::exportToWkt(), OGRGeometry↩ Factory::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometry↩ Factory::forceToPolygon(), OGRBuildPolygonFromEdges(), and OGRGeometry::Polygonize().

12.68.2.21 OGRBoolean OGRGeometryCollection::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

Reimplemented in **OGRMultiLineString** (p. **??**), **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), **OGRMulti↩ Polygon** (p. **??**), and **OGRMultiSurface** (p. **??**).

Referenced by OGRMultiSurface::hasCurveGeometry(), and OGRMultiCurve::hasCurveGeometry().

12.68.2.22 OGRErr OGRGeometryCollection::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

12.68.2.23 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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRMultiCurve** (p. **??**), **OGRMultiPoint** (p. **??**), and **OGRMultiSurface** (p. **??**).

12.68.2.24 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 Equals(), OGRMultiPoint::exportToWkt(), and getEnvelope().

12.68.2.25 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

iGeom	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
bDelete	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::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), and OGRGeometryFactory::forceToMultiPolygon().

12.68.2.26 void OGRGeometryCollection::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Add intermediate vertices to a geometry.

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

dfMaxLength	the maximum distance between 2 points after segmentization

This method modifies the geometry to add intermediate vertices if necessary so that the maximum length between 2 consecutives vertices is lower than dfMaxLength.

Parameters

dfMaxLength	maximum length between 2 consecutives vertices.

Reimplemented from **OGRGeometry** (p. **??**).

12.68.2.27 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

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. **??**).

References OGRGeometry::setCoordinateDimension().

Referenced by addGeometryDirectly(), and OGRMultiSurface::importFromWkt().

12.68.2.28 void OGRGeometryCollection::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. **??**).

12.68.2.29 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 **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

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. **??**).

References OGRGeometry::assignSpatialReference(), CPLDebug(), OGRCoordinateTransformation::GetTarget↩ CS(), and OGRGeometry::transform().

12.68.2.30 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

12.69 OGRGeometryFactory Class Reference

`#include <ogr_geometry.h>`

Static Public Member Functions

- static OGRErr **createFromWkb** (unsigned char *, **OGRSpatialReference** *, **OGRGeometry** **, int=-1, **OGRwkbVariant**=**wkbVariantOldOgc**)

  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** * **forceTo** (**OGRGeometry** *poGeom, **OGRwkbGeometryType** eTargetType, const char *const *papszOptions=NULL)

  Convert to another geometry type.
- static **OGRGeometry** * **organizePolygons** (**OGRGeometry** **papoPolygons, int nPolygonCount, int *pbResultValidGeometry, 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 dfPrimary↩ Radius, double dfSecondaryAxis, double dfRotation, double dfStartAngle, double dfEndAngle, double df↩ MaxAngleStepSizeDegrees)
- static **OGRLineString** * **curveToLineString** (double x0, double y0, double z0, double x1, double y1, double z1, double x2, double y2, double z2, int bHasZ, double dfMaxAngleStepSizeDegrees, const char *const *papszOptions=NULL)

  Converts an arc circle into an approximate line string.
- static **OGRCurve** * **curveFromLineString** (const **OGRLineString** *poLS, const char *const *papsz↩ Options=NULL)

  Try to convert a linestring approximating curves into a curve.

12.69.1 Detailed Description

Create geometry objects from well known text/binary.

12.69.2 Member Function Documentation

12.69.2.1 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

dfCenterX	center X
dfCenterY	center Y
dfZ	center Z
dfPrimaryRadius	X radius of ellipse.
dfSecondary↩ Radius	Y radius of ellipse.
dfRotation	rotation of the ellipse clockwise.
dfStartAngle	angle to first point on arc (clockwise of X-positive)
dfEndAngle	angle to last point on arc (clockwise of X-positive)
dfMaxAngle↩ StepSize↩ Degrees	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 OGRSimpleCurve::setPoint().

Referenced by OGR_G_ApproximateArcAngles().

12.69.2.2 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

pabyData	pointer to the input BLOB data.
poSR	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
ppoReturn	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure.
nBytes	the number of bytes available in pabyData.
pnBytesConsumed	if not NULL, it will be set to the number of bytes consumed (at most nBytes).

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.

12.69.2.3 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

pszData	The GML fragment for the geometry.

Returns

a geometry on succes, or NULL on error.

References OGR_G_CreateFromGML().

12.69.2.4 OGRErr OGRGeometryFactory::createFromWkb (unsigned char * *pabyData*, **OGRSpatialReference** * *poSR*, **OGRGeometry** ** *ppoReturn*, int *nBytes* = -1, **OGRwkbVariant** *eWkbVariant* = **wkbVariantOldOgc**) [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

pabyData	pointer to the input BLOB data.
poSR	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
ppoReturn	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 **OGRGeometry↩ Factory::destroyGeometry()** (p. **??**) after use.
nBytes	the number of bytes available in pabyData, or -1 if it isn't known.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGRGeometry::assignSpatialReference(), CPLDebug(), CPLGetConfigOption(), createGeometry(), CSLTestBoolean(), OGRGeometry::getLinearGeometry(), OGRGeometry::hasCurveGeometry(), and OGR↩ Geometry::importFromWkb().

Referenced by OGR_G_CreateFromWkb().

12.69.2.5 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

ppszData	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.
poSR	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
ppoReturn	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 **OGRGeometry↩ Factory::destroyGeometry()** (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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGRGeometry::assignSpatialReference(), CPLGetConfigOption(), CSLTestBoolean(), OGR↩ Geometry::getLinearGeometry(), OGRGeometry::hasCurveGeometry(), and OGRGeometry::importFromWkt().

Referenced by OGRMultiSurface::importFromWkt(), and OGR_G_CreateFromWkt().

12.69.2.6 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

eGeometryType	the type code of the geometry class to be instantiated.

Returns

the newly create geometry or NULL on failure. Should be freed with **OGRGeometryFactory::destroy↩ Geometry()** (p. **??**) after use.

References wkbCircularString, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLinearRing, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMulti↩ Surface, wkbPoint, and wkbPolygon.

Referenced by OGRSimpleCurve::clone(), OGRCurvePolygon::clone(), OGRGeometryCollection::clone(), create↩ FromWkb(), forceTo(), OGRGeometryCollection::getCurveGeometry(), OGRGeometryCollection::getLinear↩ Geometry(), and OGR_G_CreateGeometry().

12.69.2.7 OGRCurve * OGRGeometryFactory::curveFromLineString (const OGRLineString * *poLS*, const char *const * *papszOptions* = NULL) [static]

Try to convert a linestring approximating curves into a curve.

This method can return a COMPOUNDCURVE, a CIRCULARSTRING or a LINESTRING.

This method is the reverse of **curveFromLineString()** (p. **??**).

Parameters

poLS	handle to the geometry to convert.
papszOptions	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

Returns

the converted geometry (ownership to caller).

Since

GDAL 2.0

References OGRCompoundCurve::addCurveDirectly(), OGRSimpleCurve::addPoint(), OGRGeometry::assign↩ SpatialReference(), OGRSimpleCurve::clone(), OGRCompoundCurve::getNumCurves(), OGRSimpleCurve::get↩ NumPoints(), OGRSimpleCurve::getPoint(), OGRGeometry::getSpatialReference(), OGRPoint::getX(), OGR↩ SimpleCurve::getX(), OGRPoint::getY(), and OGRSimpleCurve::getY().

Referenced by OGRLineString::getCurveGeometry().

12.69.2.8 OGRLineString * OGRGeometryFactory::curveToLineString (double *x0*, double *y0*, double *z0*, double *x1*, double *y1*, double *z1*, double *x2*, double *y2*, double *z2*, int *bHasZ*, double *dfMaxAngleStepSizeDegrees*, const char *const * *papszOptions* = NULL) [static]

Converts an arc circle into an approximate line string.

The arc circle is defined by a first point, an intermediate point and a final point.

The provided dfMaxAngleStepSizeDegrees is a hint. The discretization algorithm may pick a slightly different value.

So as to avoid gaps when rendering curve polygons that share common arcs, this method is guaranteed to return a line with reversed vertex if called with inverted first and final point, and identical intermediate point.

Parameters

x0	x of first point
y0	y of first point
z0	z of first point
x1	x of intermediate point
y1	y of intermediate point
z1	z of intermediate point
x2	x of final point
y2	y of final point
z2	z of final point
bHasZ	TRUE if z must be taken into account
dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings or NULL. Recognized options: • ADD_INTERMEDIATE_POINT=STEALTH/YES/NO (Default to STEALTH). Determine if and how the intermediate point must be output in the linestring. If set to STE↩ ALTH, no explicit intermediate point is added but its properties are encoded in low significant bits of intermediate points and **OGRGeometryFactory::curveFromLine↩ String()** (p. **??**) can decode them. This is the best compromise for round-tripping in OGR and better results with PostGIS `ST_LineToCurve()` If set to YES, the intermediate point is explicitly added to the linestring. If set to NO, the intermediate point is not explicitly added.

Returns

the converted geometry (ownership to caller).

Since

GDAL 2.0

References OGRSimpleCurve::addPoint(), CPLDebug(), CPLError(), CPLParseNameValue(), OGRSimpleCurve↩ ::getNumPoints(), OGRSimpleCurve::getX(), OGRSimpleCurve::getY(), OGRSimpleCurve::reversePoints(), and OGRSimpleCurve::setPoint().

Referenced by OGRCircularString::CurveToLine().

12.69.2.9 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

poGeom	the geometry to deallocate.

Referenced by OGR_G_DestroyGeometry().

12.69.2.10 OGRGeometry * OGRGeometryFactory::forceTo (OGRGeometry * *poGeom,* OGRwkbGeometryType *eTargetType,* const char *const * *papszOptions* = NULL) [static]

Convert to another geometry type.

Tries to force the provided geometry to the specified geometry type.

It can promote 'single' geometry type to their corresponding collection type (see **OGR_GT_GetCollection()** (p. **??**)) or the reverse. non-linear geometry type to their corresponding linear geometry type (see **OGR_GT_GetLinear()** (p. **??**)), by possibly approximating circular arcs they may contain. Regarding conversion from linear geometry types to curve geometry types, only "wraping" will be done. No attempt to retrieve potential circular arcs by de-approximating stroking will be done. For that, **OGRGeometry::getCurveGeometry()** (p. **??**) can be used.

The passed in geometry is consumed and a new one returned (or potentially the same one).

Parameters

poGeom	the input geometry - ownership is passed to the method.
eTargetType	target output geometry type.
papszOptions	options as a null-terminated list of strings or NULL.

Returns

new geometry.

Since

GDAL 2.0

References OGRGeometryCollection::addGeometryDirectly(), OGRCurvePolygon::addRingDirectly(), OGRGeometry::assignSpatialReference(), OGRCurve::CastToCompoundCurve(), OGRSurface::CastToCurvePolygon(), OGRCurve::CastToLineString(), OGRMultiLineString::CastToMultiCurve(), OGRMultiPolygon::CastToMultiSurface(), OGRGeometry::clone(), createGeometry(), forceToLineString(), forceToMultiLineString(), forceToMultiPoint(), forceToMultiPolygon(), forceToPolygon(), OGRCurve::get_IsClosed(), OGRCurvePolygon::getExteriorRingCurve(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRCurvePolygon::getNumInteriorRings(), OGRCurve::getNumPoints(), OGRGeometry::getSpatialReference(), OGRGeometry::IsEmpty(), OGR_GT_GetCollection(), OGR_GT_IsCurve(), OGR_GT_IsSubClassOf(), OGRGeometryCollection::removeGeometry(), OGRCurvePolygon::stealExteriorRingCurve(), wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, and wkbUnknown.

Referenced by OGR_F_GetGeometryRef(), OGR_F_GetGeomFieldRef(), and OGR_G_ForceTo().

12.69.2.11 OGRGeometry * OGRGeometryFactory::forceToLineString (OGRGeometry * *poGeom,* bool *bOnlyInOrder* = true) [static]

Convert to line string.

Tries to force the provided geometry to be a line string. This nominally effects a change on multilinestrings. In GDAL 2.0, for polygons or curvepolygons that have a single exterior ring, it will return the ring. For circular strings or compound curves, it will return an approximated line string.

The passed in geometry is consumed and a new one returned (or potentially the same one).

Parameters

poGeom	the input geometry - ownership is passed to the method.
bOnlyInOrder	flag that, if set to FALSE, indicate that the order of points in a linestring might be reversed if it enables to match the extremity of another linestring. If set to TRUE, the start of a linestring must match the end of another linestring.

Returns

new geometry.

References OGRSimpleCurve::addSubLineString(), OGRGeometry::assignSpatialReference(), OGRCurve::CastToLineString(), OGRSimpleCurve::EndPoint(), OGRPoint::Equals(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getLinearGeometry(), OGRGeometryCollection::getNumGeometries(), OGRCurvePolygon::getNumInteriorRings(), OGRSimpleCurve::getNumPoints(), OGRGeometry::getSpatialReference(), OGRGeometry::hasCurveGeometry(), OGRGeometryCollection::removeGeometry(), OGRSimpleCurve::reversePoints(), OGRSimpleCurve::StartPoint(), OGRCurvePolygon::stealExteriorRingCurve(), wkbCircularString, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, and wkbPolygon.

Referenced by forceTo(), and OGR_G_ForceToLineString().

12.69.2.12 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.
- circularstrings and compoundcurves will be approximated and 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).
- curvepolygons will be approximated and 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 OGRGeometryCollection::addGeometryDirectly(), OGRSimpleCurve::addSubLineString(), OGRGeometry::assignSpatialReference(), OGRMultiCurve::CastToMultiLineString(), OGRPolygon::CurvePolyToPoly(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometry::getLinearGeometry(), OGRGeometryCollection::getLinearGeometry(), OGRGeometryCollection::getNumGeometries(), OGRCurvePolygon::getNumInteriorRings(), OGRSimpleCurve::getNumPoints(), OGRGeometry::getSpatialReference(), OGRGeometry::hasCurveGeometry(), OGRGeometryCollection::removeGeometry(), wkbCircularString, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPolygon, wkbMultiSurface, and wkbPolygon.

Referenced by forceTo(), and OGR_G_ForceToMultiLineString().

12.69.2.13 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 or collection of points. The passed in geometry is consumed and a new one returned (or potentially the same one).

Returns

new geometry.

References OGRGeometryCollection::addGeometryDirectly(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRGeometry::getSpatialReference(), OGRGeometryCollection::removeGeometry(), wkbFlatten, wkbGeometryCollection, wkbMultiPoint, and wkbPoint.

Referenced by forceTo(), and OGR_G_ForceToMultiPoint().

12.69.2.14 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 OGRGeometryCollection::addGeometryDirectly(), OGRGeometry::assignSpatialReference(), OGRMultiSurface::CastToMultiPolygon(), OGRPolygon::CurvePolyToPoly(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getLinearGeometry(), OGRGeometryCollection::getNumGeometries(), OGRGeometry::getSpatialReference(), OGRGeometry::hasCurveGeometry(), OGRGeometryCollection::removeGeometry(), wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbMultiPolygon, wkbMultiSurface, and wkbPolygon.

Referenced by forceTo(), and OGR_G_ForceToMultiPolygon().

12.69.2.15 OGRGeometry * OGRGeometryFactory::forceToPolygon (OGRGeometry * *poGeom*) [static]

Convert to polygon.

Tries to force the provided geometry to be a polygon. This effects a change on multipolygons. Starting with GDAL 2.0, curve polygons or closed curves will be changed to polygons. The passed in geometry is consumed and a new one returned (or potentially the same one).

Parameters

poGeom	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRCurvePolygon::addRingDirectly(), OGRGeometry::assignSpatialReference(), OGRCurve::CastToLinearRing(), OGRSurface::CastToPolygon(), OGRPolygon::CurvePolyToPoly(), OGRLineString::CurveToLine(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getLinearGeometry(), OGRGeometryCollection::getNumGeometries(), OGRCurvePolygon::getNumInteriorRings(), OGRGeometry::getSpatialReference(), OGRGeometry::hasCurveGeometry(), OGR_GT_IsCurve(), OGRPolygon::stealExteriorRing(), OGRPolygon::stealInteriorRing(), wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbMultiPolygon, wkbMultiSurface, and wkbPolygon.

Referenced by forceTo(), and OGR_G_ForceToPolygon().

12.69.2.16 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().

12.69.2.17 OGRGeometry * OGRGeometryFactory::organizePolygons (OGRGeometry ** *papoPolygons*, int *nPolygonCount*, int * *pbIsValidGeometry*, 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) that follow the OGC Simple Feature specification.

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 example in shapefiles, Personal Geodatabases or File Geodatabases).

For FileGDB, in most cases, but not always, a faster method than ONLY_CCW can be used. It is CCW_INNER_JUST_AFTER_CW_OUTER. When using it, inner rings are assumed to be counterclockwise oriented, and following immediately the outer ring (clockwise oriented) that they belong to. If that assumption is not met, an inner ring could be attached to the wrong outer ring, so this method must be used with care.

If the OGR_ORGANIZE_POLYGONS configuration option is defined, its value will override the value of the METHOD option of papszOptions (useful to modify the behaviour of the shapefile driver)

Parameters

papoPolygons	array of geometry pointers - should all be OGRPolygons. Ownership of the geometries is passed, but not of the array itself.
nPolygonCount	number of items in papoPolygons
pbIsValidGeometry	value will be set TRUE if result is valid or FALSE otherwise.
papszOptions	a list of strings for passing options

Returns

a single resulting geometry (either **OGRPolygon** (p. **??**) or **OGRMultiPolygon** (p. **??**)).

References OGRGeometryCollection::addGeometryDirectly(), OGRCurvePolygon::addRingDirectly(), CPLDebug(), CPLError(), CPLGetConfigOption(), CSLTestBoolean(), OGRPolygon::exportToWkt(), OGRCurvePolygon::get_Area(), OGRGeometry::getEnvelope(), OGRPolygon::getExteriorRing(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getPoint(), OGRPoint::getX(), OGRPoint::getY(), haveGEOS(), OGRLinearRing::isClockwise(), OGRGeometry::Overlaps(), OGRPoint::setX(), OGRPoint::setY(), wkbFlatten, and wkbPolygon.

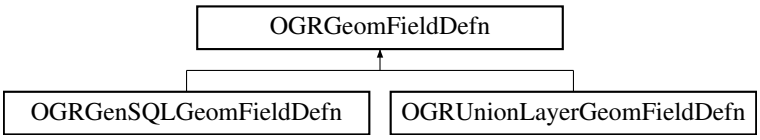
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrgeometryfactory.cpp

12.70 OGRGeomFieldDefn Class Reference

`#include <ogr_feature.h>`

Inheritance diagram for OGRGeomFieldDefn:



Public Member Functions

- **OGRGeomFieldDefn** (const char ∗pszNameIn, **OGRwkbGeometryType** eGeomTypeIn)

  Constructor.
- **OGRGeomFieldDefn** (**OGRGeomFieldDefn** ∗)

  Constructor.
- void **SetName** (const char ∗)

  Reset the name of this field.
- const char ∗ **GetNameRef** ()

  Fetch name of this field.
- **OGRwkbGeometryType GetType** ()

  Fetch geometry type of this field.
- void **SetType** (**OGRwkbGeometryType** eTypeIn)

  *Set the geometry type of this field. This should never be done to an **OGRGeomFieldDefn** (p. **??**) that is already part of an **OGRFeatureDefn** (p. **??**).*
- virtual **OGRSpatialReference** ∗ **GetSpatialRef** ()

  Fetch spatial reference system of this field.
- void **SetSpatialRef** (**OGRSpatialReference** ∗poSRSIn)

  Set the spatial reference of this field.
- int **IsIgnored** ()

  Return whether this field should be omitted when fetching features.
- void **SetIgnored** (int bIgnoreIn)

  Set whether this field should be omitted when fetching features.
- int **IsNullable** () const

  Return whether this geometry field can receive null values.
- void **SetNullable** (int bNullableIn)

  Set whether this geometry field can receive null values.
- int **IsSame** (**OGRGeomFieldDefn** ∗)

  Test if the geometry field definition is identical to the other one.

12.70.1 Detailed Description

Definition of a geometry field of an **OGRFeatureDefn** (p. **??**). A geometry field is described by :

- a name. See **SetName()** (p. **??**) / **GetNameRef()** (p. **??**)
- a type: wkbPoint, wkbLineString, ... See **SetType()** (p. **??**) / **GetType()** (p. **??**)
- a spatial reference system (optional). See **SetSpatialRef()** (p. **??**) / **GetSpatialRef()** (p. **??**)
- a NOT NULL constraint (optional). See **SetNullable()** (p. **??**) / **IsNullable()** (p. **??**)

- a boolean to indicate whether it should be ignored when retrieving features. See **SetIgnored()** (p. **??**) / **IsIgnored()** (p. **??**)

Since

OGR 1.11

12.70.2 Constructor & Destructor Documentation

12.70.2.1 OGRGeomFieldDefn::OGRGeomFieldDefn (const char * *pszNameIn*, OGRwkbGeometryType *eGeomTypeIn*)

Constructor.

Parameters

pszNameIn	the name of the new field.
eGeomTypeIn	the type of the new field.

Since

GDAL 1.11

12.70.2.2 OGRGeomFieldDefn::OGRGeomFieldDefn (OGRGeomFieldDefn * *poPrototype*)

Constructor.

Create by cloning an existing geometry field definition.

Parameters

poPrototype	the geometry field definition to clone.

Since

GDAL 1.11

References GetNameRef(), GetSpatialRef(), GetType(), IsNullable(), SetNullable(), and SetSpatialRef().

12.70.3 Member Function Documentation

12.70.3.1 const char * OGRGeomFieldDefn::GetNameRef ( ) [inline]

Fetch name of this field.

This method is the same as the C function **OGR_GFld_GetNameRef()** (p. **??**).

Returns

pointer to an internal name string that should not be freed or modified.

Since

GDAL 1.11

Referenced by OGRFeature::DumpReadable(), OGRLayer::GetGeometryColumn(), OGRUnionLayer::GetLayer↩Defn(), IsSame(), OGRGeomFieldDefn(), OGRFeature::SetFrom(), and OGRFeature::Validate().

12.70.3.2 OGRSpatialReference * OGRGeomFieldDefn::GetSpatialRef ( ) [virtual]

Fetch spatial reference system of this field.

This method is the same as the C function **OGR_GFld_GetSpatialRef()** (p. **??**).

Returns

field spatial reference system.

Since

GDAL 1.11

Referenced by OGRUnionLayer::GetLayerDefn(), OGRLayer::GetSpatialRef(), OGRUnionLayer::GetSpatialRef(), IsSame(), and OGRGeomFieldDefn().

12.70.3.3 OGRwkbGeometryType OGRGeomFieldDefn::GetType ( ) [inline]

Fetch geometry type of this field.

This method is the same as the C function **OGR_GFld_GetType()** (p. **??**).

Returns

field geometry type.

Since

GDAL 1.11

Referenced by OGRGenSQLResultsLayer::GetExtent(), OGRUnionLayer::GetGeomType(), OGRFeatureDefn::GetGeomType(), OGRUnionLayer::GetLayerDefn(), IsSame(), OGRGeomFieldDefn(), and OGRFeature::Validate().

12.70.3.4 int OGRGeomFieldDefn::IsIgnored ( ) [inline]

Return whether this field should be omitted when fetching features.

This method is the same as the C function **OGR_GFld_IsIgnored()** (p. **??**).

Returns

ignore state

Since

GDAL 1.11

Referenced by OGRFeatureDefn::IsGeometryIgnored().

12.70.3.5 int OGRGeomFieldDefn::IsNullable ( ) const [inline]

Return whether this geometry field can receive null values.

By default, fields are nullable.

Even if this method returns FALSE (i.e not-nullable field), it doesn't mean that **OGRFeature::IsFieldSet()** (p. **??**) will necessary return TRUE, as fields can be temporary unset and null/not-null validation is usually done when **OGRLayer::CreateFeature()** (p. **??**)/SetFeature() is called.

Note that not-nullable geometry fields might also contain 'empty' geometries.

This method is the same as the C function **OGR_GFld_IsNullable()** (p. **??**).

Returns

TRUE if the field is authorized to be null.

Since

GDAL 2.0

Referenced by IsSame(), OGRGeomFieldDefn(), and OGRFeature::Validate().

12.70.3.6 int OGRGeomFieldDefn::IsSame (OGRGeomFieldDefn * *poOtherFieldDefn*)

Test if the geometry field definition is identical to the other one.

Parameters

poOtherField↩ Defn	the other field definition to compare to.

Returns

TRUE if the geometry field definition is identical to the other one.

Since

GDAL 1.11

References GetNameRef(), GetSpatialRef(), GetType(), IsNullable(), and OGRSpatialReference::IsSame().

Referenced by OGRFeatureDefn::IsSame().

12.70.3.7 void OGRGeomFieldDefn::SetIgnored (int *ignore*) [inline]

Set whether this field should be omitted when fetching features.

This method is the same as the C function **OGR_GFld_SetIgnored()** (p. **??**).

Parameters

ignore	ignore state

Since

GDAL 1.11

Referenced by OGRFeatureDefn::SetGeometryIgnored(), and OGRLayer::SetIgnoredFields().

12.70.3.8 void OGRGeomFieldDefn::SetName (const char * *pszNameIn*)

Reset the name of this field.

This method is the same as the C function **OGR_GFld_SetName()** (p. **??**).

Parameters

pszNameIn	the new name to apply.

Since

GDAL 1.11

References CPLStrdup().

12.70.3.9 void OGRGeomFieldDefn::SetNullable (int *bNullableIn*) [inline]

Set whether this geometry field can receive null values.

By default, fields are nullable, so this method is generally called with FALSE to set a not-null constraint.

Drivers that support writing not-null constraint will advertize the GDAL_DCAP_NOTNULL_GEOMFIELDS driver metadata item.

This method is the same as the C function **OGR_GFld_SetNullable()** (p. **??**).

Parameters

bNullableIn	FALSE if the field must have a not-null constraint.

Since

GDAL 2.0

Referenced by OGRGeomFieldDefn().

12.70.3.10 void OGRGeomFieldDefn::SetSpatialRef (OGRSpatialReference * *poSRSIn*)

Set the spatial reference of this field.

This method is the same as the C function **OGR_GFld_SetSpatialRef()** (p. **??**).

This method drops the reference of the previously set SRS object and acquires a new reference on the passed object (if non-NULL).

Parameters

poSRSIn	the new SRS to apply.

Since

GDAL 1.11

References OGRSpatialReference::Reference(), and OGRSpatialReference::Release().

Referenced by OGRUnionLayer::GetLayerDefn(), and OGRGeomFieldDefn().

12.70.3.11 void OGRGeomFieldDefn::SetType (OGRwkbGeometryType *eType*)

Set the geometry type of this field. This should never be done to an **OGRGeomFieldDefn** (p. **??**) that is already part of an **OGRFeatureDefn** (p. **??**).

This method is the same as the C function **OGR_GFld_SetType()** (p. **??**).

Parameters

eType	the new field geometry type.

Since

GDAL 1.11

Referenced by OGRUnionLayer::GetLayerDefn(), and OGRFeatureDefn::SetGeomType().

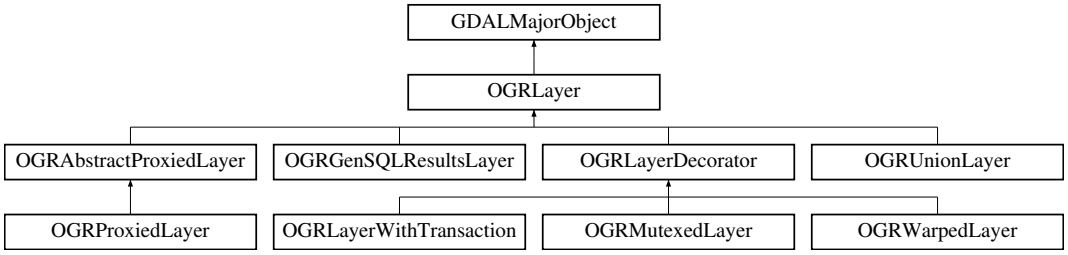
The documentation for this class was generated from the following files:

- **ogr_feature.h**
- ogrgeomfielddefn.cpp

12.71 OGRLayer Class Reference

`#include <ogrsf_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 void **SetSpatialFilter** (int iGeomField, **OGRGeometry** *)

  Set a new spatial filter.
- virtual void **SetSpatialFilterRect** (int iGeomField, double dfMinX, double dfMinY, double dfMaxX, double df↩ MaxY)

  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** (GIntBig nIndex)

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

  Fetch a feature by its identifier.

- OGRErr **SetFeature** (**OGRFeature** ∗poFeature)

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

  Create and write a new feature within a layer.
- virtual OGRErr **DeleteFeature** (GIntBig nFID)

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

  Return the layer name.
- virtual **OGRwkbGeometryType GetGeomType** ()

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

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

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

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

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

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

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

  Test if this layer supported the named capability.
- virtual 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 nFlagsIn)

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

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

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

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

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

  Set layer style table.
- virtual 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, GD↩ ALProgressFunc pfnProgress=NULL, void *pProgressArg=NULL)

  Union of two layers.
- OGRErr **SymDifference** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions, G↩ DALProgressFunc pfnProgress, void *pProgressArg)

  Symmetrical difference of two layers.
- OGRErr **Identity** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions=NULL, G↩ DALProgressFunc 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, G↩ DALProgressFunc pfnProgress=NULL, void *pProgressArg=NULL)

  Update this layer with features from the update layer.
- OGRErr **Clip** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions=NULL, GDA↩ LProgressFunc 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, GD↩ ALProgressFunc 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.

Protected Member Functions

- virtual OGRErr **ISetFeature** (**OGRFeature** *poFeature)

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

  Create and write a new feature within a layer.

12.71.1 Detailed Description

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

12.71.2 Member Function Documentation

12.71.2.1 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 existance that were obtained or created with the previous layer definition.

Not all drivers support this method. You can query a layer to check if it supports it with the OLCAlterFieldDefn capability. Some drivers may only support this method while there are still no features in the layer. When it is

supported, the existings features of the backing file/database should be updated accordingly. Some drivers might also not support all update flags.

This function is the same as the C function **OGR_L_AlterFieldDefn()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerDecorator** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References CPLError().

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), OGRLayerDecorator::AlterFieldDefn(), and OGR↩ ProxiedLayer::AlterFieldDefn().

12.71.2.2 OGRErr OGRLayer::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.

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

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

Returns

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

Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CPLErrorReset(), CreateFeature(), CSLTestBoolean(), GetFeatureCount(), OGRFeature::GetGeometryRef(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::Intersection(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), SetSpatialFilter(), and OGRGeometry::Union().

12.71.2.3 OGRErr OGRLayer::CreateFeature (**OGRFeature** * *poFeature*)

Create and write a new feature within a layer.

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

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

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

References ICreateFeature().

Referenced by Clip(), Erase(), OGRLayerDecorator::ICreateFeature(), OGRLayerWithTransaction::ICreateFeature(), OGRWarpedLayer::ICreateFeature(), OGRProxiedLayer::ICreateFeature(), OGRUnionLayer::ICreateFeature(), Identity(), Intersection(), SymDifference(), Union(), and Update().

12.71.2.4 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 existance that were obtained or created with the previous layer definition.

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

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

This function is the same as the C function **OGR_L_CreateField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerDecorator** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References CPLError().

Referenced by OGRLayerWithTransaction::CreateField(), OGRLayerDecorator::CreateField(), and OGRProxied↩ Layer::CreateField().

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

Create a new geometry field on a layer.

You must use this to create new geometry fields on a real layer. Internally the **OGRFeatureDefn** (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 existance that were obtained or created with the previous layer definition.

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

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

This function is the same as the C function **OGR_L_CreateGeomField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.11

Reimplemented in **OGRLayerWithTransaction** (p. **??**).

References CPLError().

Referenced by OGRLayerWithTransaction::CreateGeomField().

12.71.2.6 OGRErr OGRLayer::DeleteFeature (GIntBig *nFID*) [virtual]

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The **TestCapability()** (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

nFID	the feature id to be deleted from the layer

Returns

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

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by OGRLayerDecorator::DeleteFeature(), and OGRProxiedLayer::DeleteFeature().

12.71.2.7 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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_DeleteField()** (p. **??**).

Parameters

iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerDecorator** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References CPLError().

Referenced by OGRLayerWithTransaction::DeleteField(), OGRLayerDecorator::DeleteField(), and OGRProxied↩ Layer::DeleteField().

12.71.2.8 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.

12.71.2.9 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

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

Returns

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

Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CPLErrorReset(), CreateFeature(), CSLTestBoolean(), OGR↩ Geometry::Difference(), GetFeatureCount(), OGRFeature::GetGeometryRef(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), SetSpatialFilter(), and OGRGeometry::Union().

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

Find the index of field in the layer.

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

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

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

Returns

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

References OGRFeatureDefn::GetFieldIndex(), and GetLayerDefn().

12.71.2.11 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

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRGenSQLResultsLayer** (p. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by OGRLayerDecorator::GetExtent(), OGRWarpedLayer::GetExtent(), OGRGenSQLResultsLayer::GetExtent(), GetExtent(), OGRProxiedLayer::GetExtent(), and Intersection().

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

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

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

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (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.

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

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

Parameters

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRGenSQLResultsLayer** (p. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References GetExtent().

12.71.2.13 OGRFeature ∗ OGRLayer::GetFeature (GIntBig *nFID*) [virtual]

Fetch a feature by its identifier.

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

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (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 (with **GetNextFeature()** (p. **??**)) 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

nFID	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. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerWithTransaction** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References OGRGeometry::clone(), CPLStrdup(), OGRFeature::GetFID(), GetNextFeature(), ResetReading(), SetAttributeFilter(), and SetSpatialFilter().

Referenced by OGRLayerDecorator::GetFeature(), OGRLayerWithTransaction::GetFeature(), OGRWarpedLayer::GetFeature(), OGRGenSQLResultsLayer::GetFeature(), OGRProxiedLayer::GetFeature(), and OGRUnionLayer::GetFeature().

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

Fetch the feature count in this layer.

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

The returned count takes the spatial filter into account.

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

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

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

Parameters

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

Returns

feature count, -1 if count not known.

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRGenSQLResultsLayer** (p. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (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().

12.71.2.15 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. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by OGRLayerDecorator::GetFIDColumn(), and OGRProxiedLayer::GetFIDColumn().

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

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

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

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

Returns

geometry column name.

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References OGRFeatureDefn::GetGeomFieldDefn(), GetLayerDefn(), and OGRGeomFieldDefn::GetNameRef().

Referenced by OGRLayerDecorator::GetGeometryColumn(), and OGRProxiedLayer::GetGeometryColumn().

12.71.2.17 OGRwkbGeometryType OGRLayer::GetGeomType ( ) [virtual]

Return the layer geometry type.

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

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

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

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

Returns

the geometry type

Since

OGR 1.8.0

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRUnionLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References CPLDebug(), OGRFeatureDefn::GetGeomType(), GetLayerDefn(), and wkbUnknown.

Referenced by OGRLayerDecorator::GetGeomType(), OGRUnionLayer::GetGeomType(), and OGRProxiedLayer::GetGeomType().

12.71.2.18 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. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerDecorator** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

Referenced by OGRLayerWithTransaction::AlterFieldDefn(), Clip(), OGRLayerWithTransaction::CreateField(), OGRLayerWithTransaction::CreateGeomField(), Erase(), FindFieldIndex(), OGRUnionLayer::GetExtent(), GetGeometryColumn(), GetGeomType(), OGRLayerWithTransaction::GetLayerDefn(), OGRLayerDecorator::GetLayerDefn(), OGRWarpedLayer::GetLayerDefn(), OGRProxiedLayer::GetLayerDefn(), OGRUnionLayer::GetLayerDefn(), GetName(), GetSpatialRef(), OGRLayerWithTransaction::ICreateFeature(), Identity(), Intersection(), OGRLayerWithTransaction::ISetFeature(), OGR_Dr_CopyDataSource(), ReorderField(), SetAttributeFilter(), SetIgnoredFields(), SetSpatialFilter(), SymDifference(), Union(), and Update().

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

Return the layer name.

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

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRUnionLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerDecorator** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References GetLayerDefn(), and OGRFeatureDefn::GetName().

Referenced by OGRLayerDecorator::GetName(), and OGRProxiedLayer::GetName().

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

Fetch the next available feature from this layer.

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

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

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. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerWithTransaction** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by Clip(), Erase(), GetFeature(), GetFeatureCount(), OGRLayerDecorator::GetNextFeature(), OGRLayerWithTransaction::GetNextFeature(), OGRWarpedLayer::GetNextFeature(), OGRGenSQLResultsLayer::GetNextFeature(), OGRProxiedLayer::GetNextFeature(), OGRUnionLayer::GetNextFeature(), Identity(), Intersection(), SetNextByIndex(), SymDifference(), Union(), and Update().

12.71.2.21 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.

12.71.2.22 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. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by OGRLayerDecorator::GetSpatialFilter(), and OGRProxiedLayer::GetSpatialFilter().

12.71.2.23 OGRSpatialReference * OGRLayer::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.

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

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. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References OGRFeatureDefn::GetGeomFieldDefn(), GetLayerDefn(), and OGRGeomFieldDefn::GetSpatialRef().

Referenced by OGRLayerDecorator::GetSpatialRef(), OGRWarpedLayer::GetSpatialRef(), OGRProxiedLayer::GetSpatialRef(), and OGRUnionLayer::GetSpatialRef().

12.71.2.24 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. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by OGRLayerDecorator::GetStyleTable(), and OGRProxiedLayer::GetStyleTable().

12.71.2.25 OGRErr OGRLayer::ICreateFeature (OGRFeature * *poFeature*) [protected], [virtual]

Create and write a new feature within a layer.

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

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerWithTransaction** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by CreateFeature().

12.71.2.26 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

pLayerMethod	the method layer. Should not be NULL.

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

Returns

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

Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CPLErrorReset(), CreateFeature(), CSLTestBoolean(), OGR↩Geometry::Difference(), OGRGeometry::getDimension(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::Intersection(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

12.71.2.27 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

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

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

Returns

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

Since

OGR 1.10

References CPLError(), CPLErrorReset(), CreateFeature(), CSLTestBoolean(), OGRGeometry::getDimension(), OGRGeometry::getEnvelope(), GetExtent(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGR↩GeometryFactory::haveGEOS(), OGRGeometry::Intersection(), OGRGeometry::IsEmpty(), ResetReading(), OG↩RFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

12.71.2.28 OGRErr OGRLayer::ISetFeature (OGRFeature * *poFeature*) [protected], [virtual]

Rewrite an existing feature.

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

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

Parameters

poFeature	the feature to write.

Returns

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

Since

GDAL 2.0

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRWarpedLayer** (p. **??**), **OGR↩MutexedLayer** (p. **??**), **OGRLayerWithTransaction** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by SetFeature().

12.71.2.29 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.

12.71.2.30 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 existance that were obtained or created with the previous layer definition.

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

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

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

This function is the same as the C function **OGR_L_ReorderField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

References CPLError(), CPLMalloc(), OGRFeatureDefn::GetFieldCount(), GetLayerDefn(), and ReorderFields().

12.71.2.31 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 existance that were obtained or created with the previous layer definition.

panMap is such that,for each field definition at position i after reordering, its position before reordering was pan↩ Map[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

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), **OGRLayerDecorator** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References CPLError().

Referenced by ReorderField(), OGRLayerWithTransaction::ReorderFields(), OGRLayerDecorator::ReorderFields(), and OGRProxiedLayer::ReorderFields().

12.71.2.32 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. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by Clip(), Erase(), GetFeature(), GetFeatureCount(), Identity(), Intersection(), OGRLayerDecorator::ResetReading(), OGRProxiedLayer::ResetReading(), SetAttributeFilter(), SetNextByIndex(), SetSpatialFilter(), SymDifference(), Union(), and Update().

12.71.2.33 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 interprete the WHERE clause in which case the capabilities will be broader than those of OGR SQL.

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

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

Parameters

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

Returns

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

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRGenSQLResultsLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References CPLStrdup(), GetLayerDefn(), and ResetReading().

Referenced by GetFeature(), OGRLayerDecorator::SetAttributeFilter(), OGRProxiedLayer::SetAttributeFilter(), OGRGenSQLResultsLayer::SetAttributeFilter(), and OGRUnionLayer::SetAttributeFilter().

12.71.2.34 OGRErr OGRLayer::SetFeature (OGRFeature * *poFeature*)

Rewrite an existing feature.

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

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

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

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

Parameters

poFeature	the feature to write.

Returns

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

References ISetFeature().

Referenced by OGRLayerDecorator::ISetFeature(), OGRLayerWithTransaction::ISetFeature(), OGRWarpedLayer::ISetFeature(), OGRProxiedLayer::ISetFeature(), and OGRUnionLayer::ISetFeature().

12.71.2.35 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

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

Returns

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

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRUnionLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetFieldIndex(), OGRFeatureDefn::GetGeomFieldDefn(), OGRFeatureDefn::GetGeomFieldIndex(), GetLayerDefn(), OGRFeatureDefn::SetGeometryIgnored(), OGRFieldDefn::SetIgnored(), OGRGeomFieldDefn::SetIgnored(), and OGRFeatureDefn::SetStyleIgnored().

Referenced by OGRLayerDecorator::SetIgnoredFields(), OGRUnionLayer::SetIgnoredFields(), and OGRProxiedLayer::SetIgnoredFields().

12.71.2.36 OGRErr OGRLayer::SetNextByIndex (GIntBig *nIndex*) [virtual]

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

This method allows positioning of a layer such that the **GetNextFeature()** (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

nIndex	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. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References GetNextFeature(), and ResetReading().

Referenced by OGRLayerDecorator::SetNextByIndex(), OGRGenSQLResultsLayer::SetNextByIndex(), and OGRProxiedLayer::SetNextByIndex().

12.71.2.37 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

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

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRGenSQLResultsLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References ResetReading().

Referenced by Clip(), Erase(), GetFeature(), Identity(), Intersection(), OGRLayerDecorator::SetSpatialFilter(), OGRWarpedLayer::SetSpatialFilter(), SetSpatialFilter(), OGRProxiedLayer::SetSpatialFilter(), OGRGenSQLResultsLayer::SetSpatialFilter(), SetSpatialFilterRect(), SymDifference(), Union(), and Update().

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

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**GetSpatialRef()** (p. **??**)). In the future this may be generalized.

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

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

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

Parameters

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

Since

GDAL 1.11

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRGenSQLResultsLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References CPLError(), GetLayerDefn(), ResetReading(), and SetSpatialFilter().

12.71.2.39 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

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

Reimplemented in **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by OGRWarpedLayer::SetSpatialFilter(), OGRLayerDecorator::SetSpatialFilterRect(), and OGRWarpedLayer::SetSpatialFilterRect().

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

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 as the geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**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_SetSpatialFilterRectEx()** (p. **??**).

Parameters

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

Since

GDAL 1.11

Reimplemented in **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References OGRSimpleCurve::addPoint(), OGRCurvePolygon::addRing(), and SetSpatialFilter().

12.71.2.41 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

poStyleTable	pointer to style table to set

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

References OGRStyleTable::Clone().

Referenced by OGRLayerDecorator::SetStyleTable(), and OGRProxiedLayer::SetStyleTable().

12.71.2.42 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

poStyleTable	pointer to style table to set

Reimplemented in **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by OGRLayerDecorator::SetStyleTableDirectly(), and OGRProxiedLayer::SetStyleTableDirectly().

12.71.2.43 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

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

Returns

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

Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CPLErrorReset(), CreateFeature(), CSLTestBoolean(), OGR↩ Geometry::Difference(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveG↩ EOS(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometry↩ Directly(), and SetSpatialFilter().

12.71.2.44 OGRErr OGRLayer::SyncToDisk ( ) [virtual]

Flush pending changes to disk.

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

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

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

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

Returns

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

Reimplemented in **OGRUnionLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGR↩ LayerDecorator** (p. **??**).

Referenced by OGRLayerDecorator::SyncToDisk(), OGRProxiedLayer::SyncToDisk(), and OGRUnionLayer::↩ SyncToDisk().

12.71.2.45 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 **Get↩ NextFeature()** (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.

- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using **CreateGeomField()** (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.

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

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

Parameters

pszCap	the name of the capability to test.

Returns

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

Implemented in **OGRUnionLayer** (p. **??**), **OGRProxiedLayer** (p. **??**), **OGRGenSQLResultsLayer** (p. **??**), **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerDecorator** (p. **??**).

Referenced by OGRLayerDecorator::TestCapability(), OGRWarpedLayer::TestCapability(), OGRGenSQLResultsLayer::TestCapability(), and OGRProxiedLayer::TestCapability().

12.71.2.46 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

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

Returns

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

Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CPLErrorReset(), CreateFeature(), CSLTestBoolean(), OGR↩ Geometry::Difference(), OGRGeometry::getDimension(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::Intersection(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

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

Note

If the schema of the result is set by user and contains fields that have the same name as a field in the method layer, then the attribute in the result feature the originates from the method layer will get the value from the feature of the method layer.
For best performance use the minimum amount of features in the method layer and copy it into a memory layer.
This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted.
- 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

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

Returns

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

Since

OGR 1.10

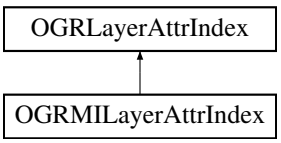
References OGRGeometry::clone(), CPLError(), CPLErrorReset(), CreateFeature(), CSLTestBoolean(), OGR↩ Geometry::Difference(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveG↩ EOS(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometry(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

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

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

12.72 OGRLayerAttrIndex Class Reference

Inheritance diagram for OGRLayerAttrIndex:

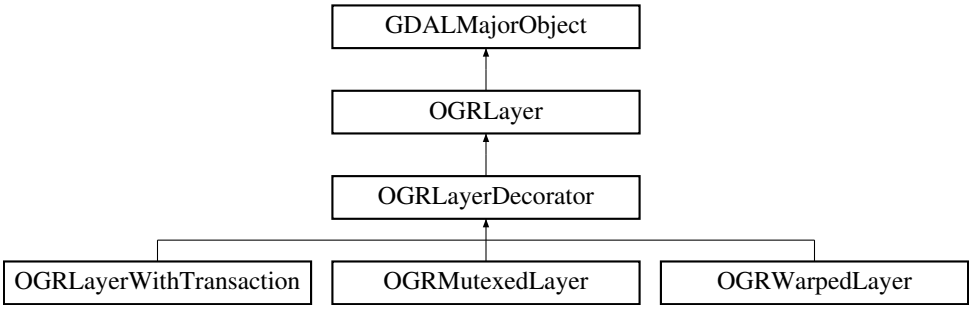


The documentation for this class was generated from the following files:

- ogr_attrind.h
- ogr_attrind.cpp

12.73 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 void **SetSpatialFilter** (int iGeomField, **OGRGeometry** *)

  Set a new spatial filter.

- virtual void **SetSpatialFilterRect** (int iGeomField, double dfMinX, double dfMinY, double dfMaxX, double dfMaxY)

  Set a new rectangular spatial filter.

- virtual OGRErr **SetAttributeFilter** (const char *)

  Set a new attribute query.

- virtual void **ResetReading** ()

  Reset feature reading to start on the first feature.

- virtual **OGRFeature** * **GetNextFeature** ()

  Fetch the next available feature from this layer.

- virtual OGRErr **SetNextByIndex** (GIntBig nIndex)

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

- virtual **OGRFeature** * **GetFeature** (GIntBig nFID)

  Fetch a feature by its identifier.

- virtual OGRErr **ISetFeature** (**OGRFeature** *poFeature)

  Rewrite an existing feature.

- virtual OGRErr **ICreateFeature** (**OGRFeature** *poFeature)

  Create and write a new feature within a layer.

- virtual OGRErr **DeleteFeature** (GIntBig 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 GIntBig **GetFeatureCount** (int bForce=TRUE)

  Fetch the feature count in this layer.

- virtual OGRErr **GetExtent** (int iGeomField, **OGREnvelope** *psExtent, int bForce=TRUE)

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

- 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.

Additional Inherited Members

12.73.1 Member Function Documentation

12.73.1.1 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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_AlterFieldDefn()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::AlterFieldDefn().

Referenced by OGRMutexedLayer::AlterFieldDefn().

12.73.1.2 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 existance that were obtained or created with the previous layer definition.

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

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

This function is the same as the C function **OGR_L_CreateField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::CreateField().

Referenced by OGRMutexedLayer::CreateField().

12.73.1.3 OGRErr OGRLayerDecorator::DeleteFeature (GIntBig *nFID*) [virtual]

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The **TestCapability()** (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

nFID	the feature id to be deleted from the layer

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::DeleteFeature().

Referenced by OGRMutexedLayer::DeleteFeature().

12.73.1.4 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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_DeleteField()** (p. **??**).

Parameters

iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::DeleteField().

Referenced by OGRMutexedLayer::DeleteField().

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

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

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

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (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.

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

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

Parameters

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetExtent().

Referenced by OGRMutexedLayer::GetExtent().

12.73.1.6 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 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

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetExtent().

12.73.1.7 OGRFeature * OGRLayerDecorator::GetFeature (GIntBig *nFID*) [virtual]

Fetch a feature by its identifier.

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

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (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 (with **GetNextFeature()** (p. **??**)) 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

nFID	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. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::GetFeature().

Referenced by OGRMutexedLayer::GetFeature().

12.73.1.8 GIntBig 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. **??**).

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

Parameters

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

Returns

feature count, -1 if count not known.

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetFeatureCount().

Referenced by OGRMutexedLayer::GetFeatureCount().

12.73.1.9 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. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetFIDColumn().

Referenced by OGRMutexedLayer::GetFIDColumn().

12.73.1.10 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.

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

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

Returns

geometry column name.

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetGeometryColumn().

Referenced by OGRMutexedLayer::GetGeometryColumn().

12.73.1.11 OGRwkbGeometryType OGRLayerDecorator::GetGeomType ( ) [virtual]

Return the layer geometry type.

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

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

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

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

Returns

the geometry type

Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetGeomType(), and wkbNone.

Referenced by OGRMutexedLayer::GetGeomType().

12.73.1.12 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. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::GetLayerDefn().

Referenced by OGRMutexedLayer::GetLayerDefn().

12.73.1.13 const char * OGRLayerDecorator::GetName ( ) [virtual]

Return the layer name.

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

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::GetName().

Referenced by OGRMutexedLayer::GetName().

12.73.1.14 OGRFeature * OGRLayerDecorator::GetNextFeature ( ) [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity 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.

Features returned by **GetNextFeature()** (p. **??**) may or may not be affected by concurrent modifications depending on drivers. A guaranteed way of seing modifications in effect is to call **ResetReading()** (p. **??**) on layers where **Get↩ NextFeature()** (p. **??**) has been called, before reading again. Structural changes in layers (field addition, deletion, ...)

when a read is in progress may or may not be possible depending on drivers. If a transaction is committed/aborted, the current sequential reading may or may not be valid after that operation and a call to **ResetReading()** (p. **??**) might be needed.

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. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::GetNextFeature().

Referenced by OGRMutexedLayer::GetNextFeature().

12.73.1.15 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. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetSpatialFilter().

Referenced by OGRMutexedLayer::GetSpatialFilter().

12.73.1.16 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.

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

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. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetSpatialRef().

Referenced by OGRMutexedLayer::GetSpatialRef().

12.73.1.17 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. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::GetStyleTable().

Referenced by OGRMutexedLayer::GetStyleTable().

12.73.1.18 OGRErr OGRLayerDecorator::ICreateFeature (OGRFeature * *poFeature*) [virtual]

Create and write a new feature within a layer.

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

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::CreateFeature().

Referenced by OGRMutexedLayer::ICreateFeature().

12.73.1.19 OGRErr OGRLayerDecorator::ISetFeature (OGRFeature * *poFeature*) [virtual]

Rewrite an existing feature.

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

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

Parameters

poFeature	the feature to write.

Returns

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

Since

GDAL 2.0

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::SetFeature().

Referenced by OGRMutexedLayer::ISetFeature().

12.73.1.20 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 existance that were obtained or created with the previous layer definition.

panMap is such that,for each field definition at position i after reordering, its position before reordering was pan↩ Map[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

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**), and **OGRLayerWithTransaction** (p. **??**).

References OGRLayer::ReorderFields().

Referenced by OGRMutexedLayer::ReorderFields().

12.73.1.21 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. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::ResetReading().

Referenced by OGRMutexedLayer::ResetReading(), and OGRWarpedLayer::SetSpatialFilter().

12.73.1.22 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 interprete the WHERE clause in which case the capabilities will be broader than those of OGR SQL.

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

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

Parameters

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

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetAttributeFilter().

Referenced by OGRMutexedLayer::SetAttributeFilter().

12.73.1.23 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

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

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetIgnoredFields().

Referenced by OGRMutexedLayer::SetIgnoredFields().

12.73.1.24 OGRErr OGRLayerDecorator::SetNextByIndex (GIntBig *nIndex*) [virtual]

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

This method allows positioning of a layer such that the **GetNextFeature()** (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

nIndex	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. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetNextByIndex().

Referenced by OGRMutexedLayer::SetNextByIndex().

12.73.1.25 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

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

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetSpatialFilter().

Referenced by OGRMutexedLayer::SetSpatialFilter().

12.73.1.26 void OGRLayerDecorator::SetSpatialFilter (int *iGeomField,* OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**Get↩SpatialRef()** (p. **??**)). In the future this may be generalized.

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

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

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

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetSpatialFilter().

12.73.1.27 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

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

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetSpatialFilterRect().

Referenced by OGRMutexedLayer::SetSpatialFilterRect().

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

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 as the geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**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_SetSpatialFilterRectEx()** (p. **??**).

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetSpatialFilterRect().

12.73.1.29 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

poStyleTable	pointer to style table to set

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetStyleTable().

Referenced by OGRMutexedLayer::SetStyleTable().

12.73.1.30 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

poStyleTable	pointer to style table to set

Reimplemented from **OGRLayer** (p. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::SetStyleTableDirectly().

Referenced by OGRMutexedLayer::SetStyleTableDirectly().

12.73.1.31 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() 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. **??**).

Reimplemented in **OGRMutexedLayer** (p. **??**).

References OGRLayer::SyncToDisk().

Referenced by OGRMutexedLayer::SyncToDisk().

12.73.1.32 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.
- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using **CreateGeomField()** (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.
- **OLCCurveGeometries** / "CurveGeometries": TRUE if this layer supports writing curve geometries or may return such geometries. (GDAL 2.0).

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

Parameters

pszCap	the name of the capability to test.

Returns

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

Implements **OGRLayer** (p. **??**).

Reimplemented in **OGRWarpedLayer** (p. **??**), and **OGRMutexedLayer** (p. **??**).

References OGRLayer::TestCapability().

Referenced by OGRMutexedLayer::TestCapability().

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

- ogrlayerdecorator.h
- ogrlayerdecorator.cpp

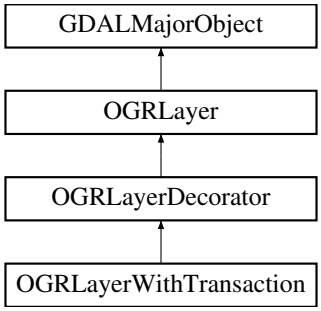
12.74 OGRLayerPool Class Reference

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

- ogrlayerpool.h
- ogrlayerpool.cpp

12.75 OGRLayerWithTransaction Class Reference

Inheritance diagram for OGRLayerWithTransaction:



Public Member Functions

- virtual const char ∗ **GetName** ()

  Return the layer name.
- virtual **OGRFeatureDefn** ∗ **GetLayerDefn** ()

  Fetch the schema information for this layer.
- virtual OGRErr **CreateField** (**OGRFieldDefn** ∗poField, int bApproxOK=TRUE)

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

  Delete an existing field on a layer.

- virtual OGRErr **ReorderFields** (int *panMap)

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

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

  Create a new geometry field on a layer.
- virtual **OGRFeature** * **GetNextFeature** ()

  Fetch the next available feature from this layer.
- virtual **OGRFeature** * **GetFeature** (GIntBig nFID)

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

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

  Create and write a new feature within a layer.

Friends

- class **OGRDataSourceWithTransaction**

Additional Inherited Members

12.75.1 Member Function Documentation

12.75.1.1 OGRErr OGRLayerWithTransaction::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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_AlterFieldDefn()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::AlterFieldDefn(), OGRFieldDefn::GetDefault(), OGRFeatureDefn::GetFieldDefn(), OGRLayer::GetLayerDefn(), OGRFieldDefn::GetNameRef(), OGRFieldDefn::GetPrecision(), OGRFieldDefn::GetSubType(), OGRFieldDefn::GetType(), OGRFieldDefn::GetWidth(), OGRFieldDefn::IsNullable(), OGRFieldDefn::SetDefault(), OGRFieldDefn::SetName(), OGRFieldDefn::SetNullable(), OGRFieldDefn::SetPrecision(), OGRFieldDefn::SetSubType(), OGRFieldDefn::SetType(), and OGRFieldDefn::SetWidth().

12.75.1.2 OGRErr OGRLayerWithTransaction::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 existance that were obtained or created with the previous layer definition.

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

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

This function is the same as the C function **OGR_L_CreateField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRFeatureDefn::AddFieldDefn(), OGRLayer::CreateField(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), and OGRLayer::GetLayerDefn().

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

Create a new geometry field on a layer.

You must use this to create new geometry fields on a real layer. Internally the **OGRFeatureDefn** (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 existance that were obtained or created with the previous layer definition.

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

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

This function is the same as the C function **OGR_L_CreateGeomField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.11

Reimplemented from **OGRLayer** (p. **??**).

References OGRFeatureDefn::AddGeomFieldDefn(), OGRLayer::CreateGeomField(), OGRFeatureDefn::GetGeomFieldCount(), OGRFeatureDefn::GetGeomFieldDefn(), and OGRLayer::GetLayerDefn().

12.75.1.4 OGRErr OGRLayerWithTransaction::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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_DeleteField()** (p. **??**).

Parameters

iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::DeleteField(), and OGRFeatureDefn::DeleteFieldDefn().

12.75.1.5 OGRFeature * OGRLayerWithTransaction::GetFeature (GIntBig *nFID*) [virtual]

Fetch a feature by its identifier.

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

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (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 (with **GetNextFeature()** (p. **??**)) 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

nFID	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::GetFID(), GetLayerDefn(), OGRFeature::SetFID(), and OGRFeature::SetFrom().

12.75.1.6 OGRFeatureDefn * OGRLayerWithTransaction::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.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRFeatureDefn::Clone(), OGRLayer::GetLayerDefn(), and OGRFeatureDefn::Reference().

Referenced by GetFeature(), and GetNextFeature().

12.75.1.7 virtual const char* OGRLayerWithTransaction::GetName ( ) [inline], [virtual]

Return the layer name.

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

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

12.75.1.8 OGRFeature * OGRLayerWithTransaction::GetNextFeature ( ) [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity 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.

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

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::GetFID(), GetLayerDefn(), OGRLayer::GetNextFeature(), OGRFeature::SetFID(), and OGRFeature::SetFrom().

12.75.1.9 OGRErr OGRLayerWithTransaction::ICreateFeature (OGRFeature * *poFeature*) [virtual]

Create and write a new feature within a layer.

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

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::CreateFeature(), OGRFeature::GetFID(), OGRLayer::GetLayerDefn(), OGRFeature::SetFID(), and OGRFeature::SetFrom().

12.75.1.10 OGRErr OGRLayerWithTransaction::ISetFeature (OGRFeature * *poFeature*) [virtual]

Rewrite an existing feature.

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

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

Parameters

poFeature	the feature to write.

Returns

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

Since

GDAL 2.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRFeature::GetFID(), OGRLayer::GetLayerDefn(), OGRLayer::SetFeature(), OGRFeature::SetFID(), and OGRFeature::SetFrom().

12.75.1.11 OGRErr OGRLayerWithTransaction::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 existance that were obtained or created with the previous layer definition.

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

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

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

This function is the same as the C function **OGR_L_ReorderFields()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRFeatureDefn::ReorderFieldDefns(), and OGRLayer::ReorderFields().

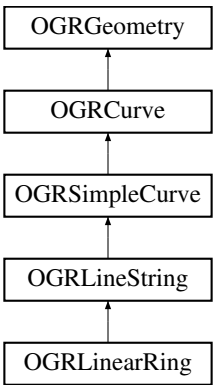
The documentation for this class was generated from the following file:

- ogremulatedtransaction.cpp

12.76 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 int **WkbSize** () const
  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1, **OGRwkbVariant**=**wkbVariantOldOgc**)
  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *, **OGRwkbVariant**=**wkbVariantOldOgc**) const
  Convert a geometry into well known binary format.

Static Protected Member Functions

- static **OGRLineString** * **CastToLineString** (**OGRLinearRing** *poLR)

  Cast to line string.

Friends

- class **OGRPolygon**

Additional Inherited Members

12.76.1 Detailed Description

Concrete representation of a closed ring.

This class is functionally equivelent 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 genearally 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.

Note: this class exists in SFSQL 1.2, but not in ISO SQL/MM Part 3.

12.76.2 Member Function Documentation

12.76.2.1 OGRLineString * OGRLinearRing::CastToLineString (OGRLinearRing * *poLR*) [static], [protected]

Cast to line string.

The passed in geometry is consumed and a new one returned .

Parameters

poLR	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRLineString::OGRLineString().

Referenced by OGRPolygon::CastToCurvePolygon().

12.76.2.2 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 **OGRSimpleCurve** (p. **??**).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), and OGRSimple↩Curve::setPoints().

12.76.2.3 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 OGRSimpleCurve::addPoint(), OGRSimpleCurve::getPoint(), OGRSimpleCurve::getX(), OGRSimple↩Curve::getY(), and OGRSimpleCurve::getZ().

12.76.2.4 OGRErr OGRLinearRing::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGR↩Geometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRSimpleCurve** (p. **??**).

12.76.2.5 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. **??**).

12.76.2.6 OGRErr OGRLinearRing::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRSimpleCurve** (p. **??**).

12.76.2.7 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().

12.76.2.8 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 **OGRSimpleCurve** (p. **??**).

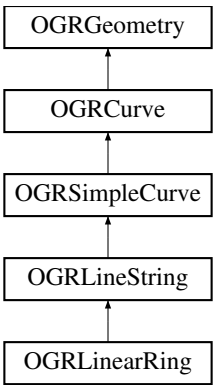
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrlinearring.cpp

12.77 OGRLineString Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRLineString:



Public Member Functions

- **OGRLineString** ()

  Create an empty line string.
- virtual **OGRLineString** ∗ **CurveToLine** (double dfMaxAngleStepSizeDegrees=0, const char ∗const ∗papsz↩ Options=NULL) const

  Return a linestring from a curve geometry.
- virtual **OGRGeometry** ∗ **getCurveGeometry** (const char ∗const ∗papszOptions=NULL) const

  Return curve version of this geometry.
- virtual double **get_Area** () const

  Get the area of the (closed) curve.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual const char ∗ **getGeometryName** () const

  Fetch WKT name for geometry type.

Protected Member Functions

- virtual double **get_AreaOfCurveSegments** () const

  Get the area of the purely curve portions of a (closed) curve.

Static Protected Member Functions

- static **OGRLinearRing** ∗ **CastToLinearRing** (**OGRLineString** ∗poLS)

  Cast to linear ring.

Additional Inherited Members

12.77.1 Detailed Description

Concrete representation of a multi-vertex line.

Note: for implementation convenience, we make it inherit from **OGRSimpleCurve** (p. **??**) whereas SFSQL and SQL/MM only make it inherits from **OGRCurve** (p. **??**).

12.77.2 Member Function Documentation

12.77.2.1 OGRLinearRing * OGRLineString::CastToLinearRing (OGRLineString * *poLS*) [static], [protected]

Cast to linear ring.

The passed in geometry is consumed and a new one returned (or NULL in case of failure)

Parameters

poLS	the input geometry - ownership is passed to the method.

Returns

new geometry.

References CPLError(), and OGRCurve::get_IsClosed().

12.77.2.2 OGRLineString * OGRLineString::CurveToLine (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return a linestring from a curve geometry.

The returned geometry is a new instance whose ownership belongs to the caller.

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.

This method relates to the ISO SQL/MM Part 3 ICurve::CurveToLine() method.

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

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings or NULL. See **OGRGeometryFactory::curveTo↩ LineString()** (p. **??**) for valid options.

Returns

a line string approximating the curve

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

References OGRSimpleCurve::clone().

Referenced by OGRGeometryFactory::forceToPolygon().

12.77.2.3 double OGRLineString::get_Area ( ) const [virtual]

Get the area of the (closed) curve.

This method is designed to be used by **OGRCurvePolygon::get_Area()** (p. **??**).

Returns

the area of the feature in square units of the spatial reference system in use.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

References OGRSimpleCurve::get_LinearArea().

Referenced by OGRCircularString::get_Area(), and OGRCompoundCurve::get_Area().

12.77.2.4 double OGRLineString::get_AreaOfCurveSegments ( ) const [protected], [virtual]

Get the area of the purely curve portions of a (closed) curve.

This method is designed to be used on a closed convex curve.

Returns

the area of the feature in square units of the spatial reference system in use.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

12.77.2.5 OGRGeometry * OGRLineString::getCurveGeometry (const char *const * *papszOptions* = NULL) const [virtual]

Return curve version of this geometry.

Returns a geometry that has possibly CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICU↩RVE or MULTISURFACE in it, by de-approximating curve geometries.

If the geometry has no curve portion, the returned geometry will be a clone of it.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getLinearGeometry()** (p. **??**).

This function is the same as C function **OGR_G_GetCurveGeometry()** (p. **??**).

Parameters

papszOptions	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

References OGRGeometryFactory::curveFromLineString().

12.77.2.6 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. **??**).

12.77.2.7 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()** (p. **??**) 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. **??**).

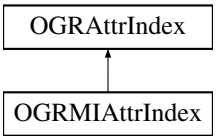
References wkbLineString, and wkbLineString25D.

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

- **ogr_geometry.h**
- ogrlinestring.cpp

12.78 OGRMIAttrIndex Class Reference

Inheritance diagram for OGRMIAttrIndex:

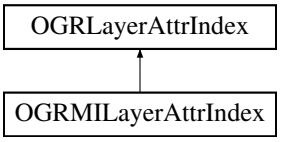


The documentation for this class was generated from the following file:

- ogr_miattrind.cpp

12.79 OGRMILayerAttrIndex Class Reference

Inheritance diagram for OGRMILayerAttrIndex:



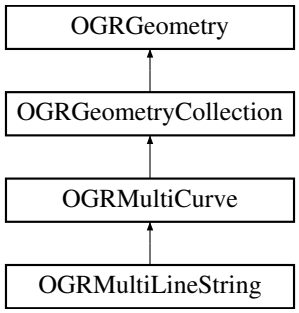
The documentation for this class was generated from the following file:

- ogr_miattrind.cpp

12.80 OGRMultiCurve Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiCurve:



Public Member Functions

- **OGRMultiCurve** ()

  Create an empty multi curve collection.
- virtual const char ∗ **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual OGRErr **importFromWkt** (char ∗∗)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char ∗∗, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- virtual int **getDimension** () const

  Get the dimension of this object.
- virtual OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.

Static Public Member Functions

- static **OGRMultiLineString** ∗ **CastToMultiLineString** (**OGRMultiCurve** ∗poMC)

  Cast to multi line string.

12.80.1 Detailed Description

A collection of **OGRCurve** (p. **??**).

Since

GDAL 2.0

12.80.2 Member Function Documentation

12.80.2.1 OGRMultiLineString * OGRMultiCurve::CastToMultiLineString (OGRMultiCurve * *poMC*) [static]

Cast to multi line string.

This method should only be called if the multicurve actually only contains instances of **OGRLineString** (p. **??**). This can be verified if hasCurveGeometry(TRUE) returns FALSE. It is not intended to approximate circular curves. For that use **getLinearGeometry()** (p. **??**).

The passed in geometry is consumed and a new one returned (or NULL in case of failure).

Parameters

poMS	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRCurve::CastToLineString().

Referenced by OGRGeometryFactory::forceToMultiLineString().

12.80.2.2 OGRErr OGRMultiCurve::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. **??**).

Reimplemented in **OGRMultiLineString** (p. **??**).

References wkbVariantIso.

12.80.2.3 int OGRMultiCurve::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::getCoordinate↩ Dimension()** (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. **??**).

12.80.2.4 const char * OGRMultiCurve::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. **??**).

Reimplemented in **OGRMultiLineString** (p. **??**).

12.80.2.5 OGRwkbGeometryType OGRMultiCurve::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()** (p. **??**) 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. **??**).

Reimplemented in **OGRMultiLineString** (p. **??**).

References OGRGeometry::getCoordinateDimension(), wkbMultiCurve, and wkbMultiCurveZ.

Referenced by importFromWkt().

12.80.2.6 OGRBoolean OGRMultiCurve::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()**

(p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometryCollection** (p. **??**).

Reimplemented in **OGRMultiLineString** (p. **??**).

References OGRGeometryCollection::hasCurveGeometry().

12.80.2.7 OGRErr OGRMultiCurve::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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRGeometryCollection** (p. **??**).

References getGeometryType(), wkbFlatten, and wkbMultiCurve.

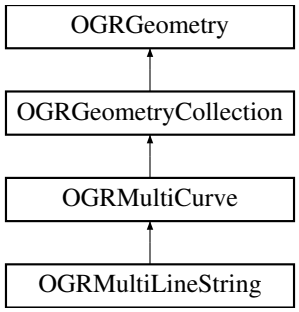
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmulticurve.cpp

12.81 OGRMultiLineString Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiLineString:



Public Member Functions

- **OGRMultiLineString** ()

  Create an empty multi line string collection.
- virtual const char ∗ **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual OGRErr **exportToWkt** (char ∗∗, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- virtual OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.

Static Public Member Functions

- static **OGRMultiCurve** ∗ **CastToMultiCurve** (**OGRMultiLineString** ∗poMLS)

  Cast to multicurve.

12.81.1 Detailed Description

A collection of **OGRLineString** (p. **??**).

12.81.2 Member Function Documentation

12.81.2.1 OGRMultiCurve ∗ OGRMultiLineString::CastToMultiCurve (OGRMultiLineString ∗ *poMLS*) [static]

Cast to multicurve.

The passed in geometry is consumed and a new one returned .

Parameters

poMLS	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRMultiCurve::OGRMultiCurve().

Referenced by OGRGeometryFactory::forceTo().

12.81.2.2 OGRErr OGRMultiLineString::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant =* wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRMultiCurve** (p. **??**).

12.81.2.3 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 **OGRMultiCurve** (p. **??**).

12.81.2.4 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()** (p. **??**) 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 **OGRMultiCurve** (p. **??**).

References OGRGeometry::getCoordinateDimension(), wkbMultiLineString, and wkbMultiLineString25D.

12.81.2.5 OGRBoolean OGRMultiLineString::hasCurveGeometry (int *bLookForNonLinear =* FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRMultiCurve** (p. **??**).

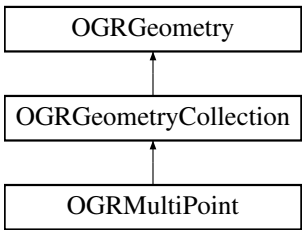
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultilinestring.cpp

12.82 OGRMultiPoint Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiPoint:



Public Member Functions

- **OGRMultiPoint** ()

  Create an empty multi point collection.
- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual OGRErr **importFromWkt** (char **)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char **, **OGRwkbVariant**=**wkbVariantOldOgc**) const

Convert a geometry into well known text format.

- virtual int **getDimension** () const

  Get the dimension of this object.

- virtual OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.

12.82.1 Detailed Description

A collection of **OGRPoint** (p. **??**).

12.82.2 Member Function Documentation

12.82.2.1 OGRErr OGRMultiPoint::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. **??**).

References CPLDebug(), CPLRealloc(), CPLStrdup(), OGRGeometry::getCoordinateDimension(), OGRPoint::getCoordinateDimension(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), OGRPoint::IsEmpty(), OGRGeometryCollection::IsEmpty(), and wkbVariantIso.

12.82.2.2 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. **??**).

12.82.2.3 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().

12.82.2.4 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()** (p. **??**) 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.

12.82.2.5 OGRBoolean OGRMultiPoint::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometryCollection** (p. **??**).

12.82.2.6 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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRGeometryCollection** (p. **??**).

References OGRGeometryCollection::addGeometryDirectly().

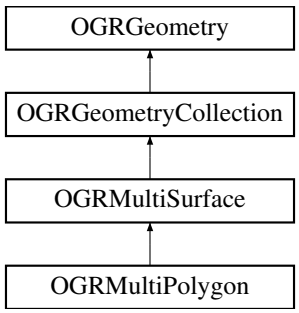
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultipoint.cpp

12.83 OGRMultiPolygon Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiPolygon:



Public Member Functions

- **OGRMultiPolygon** ()

  Create an empty multi polygon collection.
- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual OGRErr **exportToWkt** (char **, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- virtual OGRErr **PointOnSurface** (**OGRPoint** *poPoint) const

This method relates to the SFCOM IMultiSurface::get_PointOnSurface() method.

- virtual OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.

Static Public Member Functions

- static **OGRMultiSurface** * **CastToMultiSurface** (**OGRMultiPolygon** *poMP)

  Cast to multisurface.

12.83.1 Detailed Description

A collection of non-overlapping **OGRPolygon** (p. **??**).

12.83.2 Member Function Documentation

12.83.2.1 OGRMultiSurface * OGRMultiPolygon::CastToMultiSurface (OGRMultiPolygon * *poMP*) [static]

Cast to multisurface.

The passed in geometry is consumed and a new one returned .

Parameters

poMP	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRMultiSurface::OGRMultiSurface().

Referenced by OGRGeometryFactory::forceTo().

12.83.2.2 OGRErr OGRMultiPolygon::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRMultiSurface** (p. **??**).

12.83.2.3 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 **OGRMultiSurface** (p. **??**).

12.83.2.4 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()** (p. **??**) 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 **OGRMultiSurface** (p. **??**).

References OGRGeometry::getCoordinateDimension(), wkbMultiPolygon, and wkbMultiPolygon25D.

12.83.2.5 OGRBoolean OGRMultiPolygon::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRMultiSurface** (p. **??**).

12.83.2.6 OGRErr OGRMultiPolygon::PointOnSurface (OGRPoint * *poPoint*) const [virtual]

This method relates to the SFCOM IMultiSurface::get_PointOnSurface() method.

NOTE: Only implemented when GEOS included in build.

Parameters

poPoint	point to be set with an internal point.

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Reimplemented from **OGRMultiSurface** (p. **??**).

References OGRPoint::empty(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::IsEmpty(), OGR_G_PointOn↩ Surface(), OGRPoint::setX(), and OGRPoint::setY().

Referenced by OGRMultiSurface::PointOnSurface().

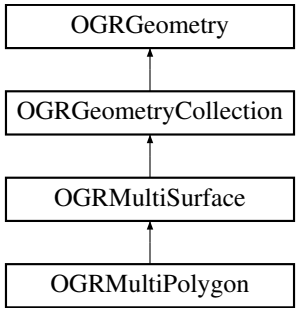
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultipolygon.cpp

12.84 OGRMultiSurface Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiSurface:



Public Member Functions

- **OGRMultiSurface** ()

  Create an empty multi surface collection.
- virtual const char * **getGeometryName** () const

  Fetch WKT name for geometry type.
- virtual **OGRwkbGeometryType getGeometryType** () const

  Fetch geometry type.
- virtual OGRErr **importFromWkt** (char **)

  Assign geometry from well known text data.
- virtual OGRErr **exportToWkt** (char **, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- virtual OGRErr **PointOnSurface** (**OGRPoint** *poPoint) const

  This method relates to the SFCOM IMultiSurface::get_PointOnSurface() method.
- virtual int **getDimension** () const

  Get the dimension of this object.
- virtual OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.

Static Public Member Functions

- static **OGRMultiPolygon** * **CastToMultiPolygon** (**OGRMultiSurface** *poMS)

  Cast to multipolygon.

12.84.1 Detailed Description

A collection of non-overlapping **OGRSurface** (p. **??**).

Since

GDAL 2.0

12.84.2 Member Function Documentation

12.84.2.1 OGRMultiPolygon * OGRMultiSurface::CastToMultiPolygon (OGRMultiSurface * *poMS*) [static]

Cast to multipolygon.

This method should only be called if the multisurface actually only contains instances of **OGRPolygon** (p. **??**). This can be verified if hasCurveGeometry(TRUE) returns FALSE. It is not intended to approximate curve polygons. For that use **getLinearGeometry()** (p. **??**).

The passed in geometry is consumed and a new one returned (or NULL in case of failure).

Parameters

poMS	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRSurface::CastToPolygon().

Referenced by OGRGeometryFactory::forceToMultiPolygon().

12.84.2.2 OGRErr OGRMultiSurface::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. **??**).

Reimplemented in **OGRMultiPolygon** (p. **??**).

References wkbVariantIso.

12.84.2.3 int OGRMultiSurface::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::getCoordinate↩ Dimension()** (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. **??**).

12.84.2.4 const char * OGRMultiSurface::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. **??**).

Reimplemented in **OGRMultiPolygon** (p. **??**).

12.84.2.5 OGRwkbGeometryType OGRMultiSurface::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()** (p. **??**) 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. **??**).

Reimplemented in **OGRMultiPolygon** (p. **??**).

References OGRGeometry::getCoordinateDimension(), wkbMultiSurface, and wkbMultiSurfaceZ.

Referenced by importFromWkt().

12.84.2.6 OGRBoolean OGRMultiSurface::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNonLinear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometryCollection** (p. **??**).

Reimplemented in **OGRMultiPolygon** (p. **??**).

References OGRGeometryCollection::hasCurveGeometry().

12.84.2.7 OGRErr OGRMultiSurface::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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRGeometryCollection** (p. **??**).

References OGRGeometryCollection::addGeometryDirectly(), CPLError(), OGRGeometryFactory::createFrom↩ Wkt(), getGeometryType(), OGRGeometryCollection::setCoordinateDimension(), wkbFlatten, and wkbMultiSurface.

12.84.2.8 OGRErr OGRMultiSurface::PointOnSurface (OGRPoint * *poPoint*) const [virtual]

This method relates to the SFCOM IMultiSurface::get_PointOnSurface() method.

NOTE: Only implemented when GEOS included in build.

Parameters

poPoint	point to be set with an internal point.

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Reimplemented in **OGRMultiPolygon** (p. **??**).

References OGRGeometryCollection::getLinearGeometry(), and OGRMultiPolygon::PointOnSurface().

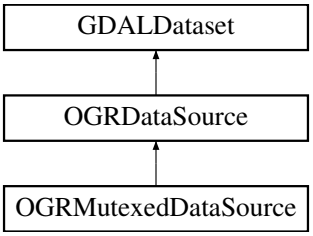
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultisurface.cpp

12.85 OGRMutexedDataSource Class Reference

`#include <ogrmutexeddatasource.h>`

Inheritance diagram for OGRMutexedDataSource:



12.85.1 Detailed Description

OGRMutexedDataSource (p. **??**) class protects all virtual methods of **OGRDataSource** (p. **??**) with a mutex. If the passed mutex is NULL, then no locking will be done.

Note that the constructors and destructors are not explictely protected by the mutex*

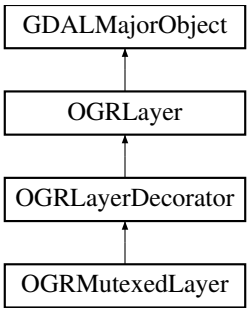
The documentation for this class was generated from the following files:

- ogrmutexeddatasource.h
- ogrmutexeddatasource.cpp

12.86 OGRMutexedLayer Class Reference

`#include <ogrmutexedlayer.h>`

Inheritance diagram for OGRMutexedLayer:



Public Member Functions

- virtual **OGRGeometry** ∗ **GetSpatialFilter** ()

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

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

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

  Set a new spatial filter.
- virtual void **SetSpatialFilterRect** (int iGeomField, double dfMinX, double dfMinY, double dfMaxX, double df↩ MaxY)

  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** (GIntBig nIndex)

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

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

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

  Create and write a new feature within a layer.
- virtual OGRErr **DeleteFeature** (GIntBig 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 GIntBig **GetFeatureCount** (int bForce=TRUE)

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

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

Additional Inherited Members

12.86.1 Detailed Description

OGRMutexedLayer (p. **??**) class protects all virtual methods of **OGRLayer** (p. **??**) with a mutex.

If the passed mutex is NULL, then no locking will be done.

Note that the constructors and destructors are not explictely protected by the mutex.

12.86.2 Member Function Documentation

12.86.2.1 OGRErr OGRMutexedLayer::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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_AlterFieldDefn()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::AlterFieldDefn().

12.86.2.2 OGRErr OGRMutexedLayer::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 existance that were obtained or created with the previous layer definition.

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

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

This function is the same as the C function **OGR_L_CreateField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::CreateField().

12.86.2.3 OGRErr OGRMutexedLayer::DeleteFeature (GIntBig *nFID*) [virtual]

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The **TestCapability()** (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

nFID	the feature id to be deleted from the layer

Returns

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::DeleteFeature().

12.86.2.4 OGRErr OGRMutexedLayer::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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_DeleteField()** (p. **??**).

Parameters

iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::DeleteField().

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

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

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

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (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.

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

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

Parameters

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetExtent().

12.86.2.6 OGRErr OGRMutexedLayer::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

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetExtent().

12.86.2.7 OGRFeature * OGRMutexedLayer::GetFeature (GIntBig *nFID*) [virtual]

Fetch a feature by its identifier.

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

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (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 (with **GetNextFeature()** (p. **??**)) 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

nFID	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 OGRLayerDecorator::GetFeature().

12.86.2.8 GIntBig OGRMutexedLayer::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. **??**).

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

Parameters

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

Returns

feature count, -1 if count not known.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetFeatureCount().

12.86.2.9 const char * OGRMutexedLayer::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 **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetFIDColumn().

12.86.2.10 const char * OGRMutexedLayer::GetGeometryColumn ( ) [virtual]

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

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

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

Returns

geometry column name.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetGeometryColumn().

12.86.2.11 OGRwkbGeometryType OGRMutexedLayer::GetGeomType ( ) [virtual]

Return the layer geometry type.

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

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

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

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

Returns

the geometry type

Since

OGR 1.8.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetGeomType().

12.86.2.12 OGRFeatureDefn * OGRMutexedLayer::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.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetLayerDefn().

12.86.2.13 const char * OGRMutexedLayer::GetName ( ) [virtual]

Return the layer name.

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

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetName().

12.86.2.14 OGRFeature * OGRMutexedLayer::GetNextFeature ( ) [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity 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.

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

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 OGRLayerDecorator::GetNextFeature().

12.86.2.15 OGRGeometry * OGRMutexedLayer::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 **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetSpatialFilter().

12.86.2.16 OGRSpatialReference * OGRMutexedLayer::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.

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

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. **??**).

References OGRLayerDecorator::GetSpatialRef().

12.86.2.17 OGRStyleTable * OGRMutexedLayer::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 **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::GetStyleTable().

12.86.2.18 OGRErr OGRMutexedLayer::ICreateFeature (OGRFeature * *poFeature*) [virtual]

Create and write a new feature within a layer.

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

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::ICreateFeature().

12.86.2.19 OGRErr OGRMutexedLayer::ISetFeature (OGRFeature * *poFeature*) [virtual]

Rewrite an existing feature.

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

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

Parameters

poFeature	the feature to write.

Returns

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

Since

GDAL 2.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::ISetFeature().

12.86.2.20 OGRErr OGRMutexedLayer::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 existance that were obtained or created with the previous layer definition.

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

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

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

This function is the same as the C function **OGR_L_ReorderFields()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::ReorderFields().

12.86.2.21 void OGRMutexedLayer::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. **??**).

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::ResetReading().

12.86.2.22 OGRErr OGRMutexedLayer::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 interprete the WHERE clause in which case the capabilities will be broader than those of OGR SQL.

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

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

Parameters

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

Returns

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetAttributeFilter().

12.86.2.23 OGRErr OGRMutexedLayer::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

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

Returns

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetIgnoredFields().

12.86.2.24 OGRErr OGRMutexedLayer::SetNextByIndex (GIntBig *nIndex*) [virtual]

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

This method allows positioning of a layer such that the **GetNextFeature()** (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

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

Returns

OGRERR_NONE on success or an error code.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetNextByIndex().

12.86.2.25 void OGRMutexedLayer::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

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetSpatialFilter().

12.86.2.26 void OGRMutexedLayer::SetSpatialFilter (int *iGeomField,* OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**GetSpatialRef()** (p. **??**)). In the future this may be generalized.

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

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

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

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetSpatialFilter().

12.86.2.27 void OGRMutexedLayer::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

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetSpatialFilterRect().

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

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 as the geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**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_SetSpatialFilterRectEx()** (p. **??**).

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetSpatialFilterRect().

12.86.2.29 void OGRMutexedLayer::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

poStyleTable	pointer to style table to set

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetStyleTable().

12.86.2.30 void OGRMutexedLayer::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

poStyleTable	pointer to style table to set

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SetStyleTableDirectly().

12.86.2.31 OGRErr OGRMutexedLayer::SyncToDisk ( ) [virtual]

Flush pending changes to disk.

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

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

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

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

Returns

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::SyncToDisk().

12.86.2.32 int OGRMutexedLayer::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.
- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using **CreateGeomField()** (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 Rollback↩ Transaction() 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.
- **OLCCurveGeometries** / "CurveGeometries": TRUE if this layer supports writing curve geometries or may return such geometries. (GDAL 2.0).

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

Parameters

pszCap	the name of the capability to test.

Returns

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayerDecorator::TestCapability().

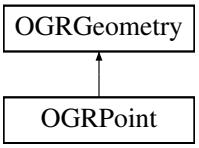
The documentation for this class was generated from the following files:

- ogrmutexedlayer.h
- ogrmutexedlayer.cpp

12.87 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, **OGRwkbVariant**=**wkbVariantOldOgc**)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char ∗, **OGRwkbVariant**=**wkbVariantOldOgc**) 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, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.

- virtual int **getDimension** () const

  Get the dimension of this object.
- virtual int **getCoordinateDimension** () const

  Get the dimension of the coordinates in 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 OGRBoolean **Intersects** (const **OGRGeometry** ∗) const

  Do these features intersect?
- virtual OGRBoolean **Within** (const **OGRGeometry** ∗) const

  Test for containment.
- 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.

12.87.1 Detailed Description

Point class.

Implements SFCOM IPoint methods.

12.87.2 Member Function Documentation

12.87.2.1 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().

12.87.2.2 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(), OGRPoint(), OGRPolygon::PointOnSurface(), and OGRMultiPolygon↩ ::PointOnSurface().

12.87.2.3 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(), getZ(), OGRGeometry::Is↩ Empty(), and IsEmpty().

Referenced by OGRGeometryFactory::forceToLineString().

12.87.2.4 OGRErr OGRPoint::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

References getCoordinateDimension(), getGeometryType(), OGRGeometry::getIsoGeometryType(), IsEmpty(), and wkbVariantIso.

12.87.2.5 OGRErr OGRPoint::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

References CPLStrdup(), getCoordinateDimension(), IsEmpty(), and wkbVariantIso.

12.87.2.6 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. **??**).

12.87.2.7 int OGRPoint::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.

Reimplemented from **OGRGeometry** (p. **??**).

Referenced by OGRSimpleCurve::addPoint(), exportToWkb(), exportToWkt(), OGRMultiPoint::exportToWkt(), getGeometryType(), OGRSimpleCurve::setPoint(), and WkbSize().

12.87.2.8 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. **??**).

12.87.2.9 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

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. **??**).

References getX(), and getY().

12.87.2.10 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

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. **??**).

References getX(), getY(), and getZ().

12.87.2.11 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. **??**).

12.87.2.12 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()** (p. **??**) 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 getCoordinateDimension(), wkbPoint, and wkbPoint25D.

Referenced by Equals(), exportToWkb(), and OGR_G_Centroid().

12.87.2.13 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 OGRSimpleCurve::addPoint(), OGRGeometry::Centroid(), OGRCircularString::ContainsPoint(), OGRGeometryFactory::curveFromLineString(), Equals(), OGRMultiPoint::exportToWkt(), OGRCompound↩Curve::get_Area(), OGRCurve::get_IsClosed(), getEnvelope(), OGRCurve::IsConvex(), OGRGeometryFactory↩::organizePolygons(), OGRPolygon::PointOnSurface(), OGRMultiPolygon::PointOnSurface(), and OGRSimple↩Curve::setPoint().

12.87.2.14 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 OGRSimpleCurve::addPoint(), OGRGeometry::Centroid(), OGRCircularString::ContainsPoint(), OGRGeometryFactory::curveFromLineString(), Equals(), OGRMultiPoint::exportToWkt(), OGRCompound↩Curve::get_Area(), OGRCurve::get_IsClosed(), getEnvelope(), OGRCurve::IsConvex(), OGRGeometryFactory↩::organizePolygons(), OGRPolygon::PointOnSurface(), OGRMultiPolygon::PointOnSurface(), and OGRSimple↩Curve::setPoint().

12.87.2.15 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 OGRSimpleCurve::addPoint(), Equals(), OGRMultiPoint::exportToWkt(), getEnvelope(), and OGR↩SimpleCurve::setPoint().

12.87.2.16 OGRErr OGRPoint::importFromWkb (unsigned char * *pabyData,* int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

12.87.2.17 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

ppszInput	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_UNSUP↩PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

12.87.2.18 OGRBoolean OGRPoint::Intersects (const OGRGeometry * *poOtherGeom*) const [virtual]

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigerous 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

poOtherGeom	the other geometry to test against.

Returns

TRUE if the geometries intersect, otherwise FALSE.

Reimplemented from **OGRGeometry** (p. **??**).

References OGRGeometry::getGeometryType(), OGRGeometry::Intersects(), IsEmpty(), wkbCurvePolygon, and wkbFlatten.

12.87.2.19 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(), Equals(), exportToWkb(), exportToWkt(), OGRMultiPoint::exportToWkt(), Intersects(), OGRPolygon::PointOnSurface(), OGRMultiPolygon::PointOnSurface(), and Within().

12.87.2.20 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

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. **??**).

Referenced by clone().

12.87.2.21 void OGRPoint::setX (double *xIn*) [inline]

Assign point X coordinate.

There is no corresponding SFCOM method.

Referenced by OGRGeometry::Centroid(), OGRSimpleCurve::getPoint(), OGRCurve::IsConvex(), OGR↩ GeometryFactory::organizePolygons(), OGRPolygon::PointOnSurface(), OGRMultiPolygon::PointOnSurface(), OGRSimpleCurve::Value(), and OGRCircularString::Value().

12.87.2.22 void OGRPoint::setY (double *yIn*) [inline]

Assign point Y coordinate.

There is no corresponding SFCOM method.

Referenced by OGRGeometry::Centroid(), OGRSimpleCurve::getPoint(), OGRCurve::IsConvex(), OGR↩ GeometryFactory::organizePolygons(), OGRPolygon::PointOnSurface(), OGRMultiPolygon::PointOnSurface(), OGRSimpleCurve::Value(), and OGRCircularString::Value().

12.87.2.23 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 OGRSimpleCurve::getPoint(), OGRSimpleCurve::Value(), and OGRCircularString::Value().

12.87.2.24 void OGRPoint::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. **??**).

12.87.2.25 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 **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

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. **??**).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), and OG↩ RCoordinateTransformation::Transform().

12.87.2.26 OGRBoolean OGRPoint::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

poOtherGeom	the geometry to compare to this geometry.

Returns

TRUE if poOtherGeom is within this geometry, otherwise FALSE.

Reimplemented from **OGRGeometry** (p. **??**).

References OGRGeometry::Contains(), OGRGeometry::getGeometryType(), IsEmpty(), OGRGeometry::Within(), wkbCurvePolygon, and wkbFlatten.

12.87.2.27 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. **??**).

References getCoordinateDimension().

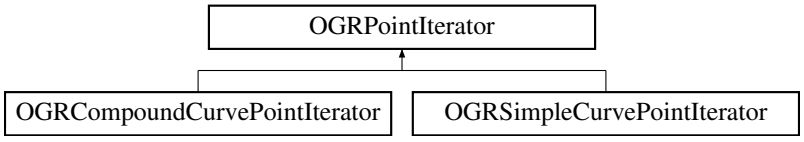
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrpoint.cpp

12.88 OGRPointIterator Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRPointIterator:



Public Member Functions

- virtual OGRBoolean **getNextPoint** (**OGRPoint** *p)=0

  Returns the next point followed by the iterator.

Static Public Member Functions

- static void **destroy** (**OGRPointIterator** *)

  Destroys a point iterator.

12.88.1 Detailed Description

Interface for a point iterator.

Since

GDAL 2.0

12.88.2 Member Function Documentation

12.88.2.1 void OGRPointIterator::destroy (OGRPointIterator * *polter*) [static]

Destroys a point iterator.

Since

GDAL 2.0

12.88.2.2 OGRBoolean OGRPointIterator::getNextPoint (OGRPoint * *p*) [pure virtual]

Returns the next point followed by the iterator.

Parameters

p	point to fill.

Returns

TRUE in case of success, or FALSE if the end of the curve is reached.

Since

GDAL 2.0

Implemented in **OGRSimpleCurvePointIterator** (p. **??**), and **OGRCompoundCurvePointIterator** (p. **??**).

Referenced by OGRCompoundCurve::get_Area(), OGRCompoundCurvePointIterator::getNextPoint(), and OGR↩Curve::IsConvex().

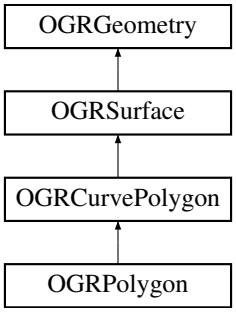
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrcurve.cpp

12.89 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 OGRBoolean **hasCurveGeometry** (int bLookForNonLinear=FALSE) const

  Returns if this geometry is or has curve geometry.
- virtual **OGRGeometry** * **getCurveGeometry** (const char *const *papszOptions=NULL) const

  Return curve version of this geometry.
- virtual **OGRGeometry** * **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char *const *papszOptions=NULL) const

  Return, possibly approximate, non-curve version of this geometry.
- 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, **OGRwkbVariant**=**wkbVariantOldOgc**)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *, **OGRwkbVariant**=**wkbVariantOldOgc**) 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, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- virtual **OGRPolygon** * **CurvePolyToPoly** (double dfMaxAngleStepSizeDegrees=0, const char *const *papszOptions=NULL) const

  Return a polygon from a curve polygon.
- **OGRLinearRing** * **getExteriorRing** ()

  Fetch reference to external polygon ring.
- **OGRLinearRing** * **getInteriorRing** (int)

  Fetch reference to indicated internal ring.

- **OGRLinearRing** * **stealExteriorRing** ()

  "Steal" reference to external polygon ring.
- **OGRLinearRing** * **stealInteriorRing** (int)

  "Steal" reference to indicated interior ring.
- virtual void **closeRings** ()

  Force rings to be closed.

Static Protected Member Functions

- static **OGRCurvePolygon** * **CastToCurvePolygon** (**OGRPolygon** *poPoly)

  Cast to curve polygon.

Friends

- class **OGRMultiSurface**

Additional Inherited Members

12.89.1 Detailed Description

Concrete class representing polygons.

Note that the OpenGIS simple features polygons consist of one outer ring (linearring), and zero or more inner rings. A polygon cannot represent disconnected regions (such as multiple islands in a political body). The **OGRMulti↩ Polygon** (p. **??**) must be used for this.

12.89.2 Member Function Documentation

12.89.2.1 OGRCurvePolygon * OGRPolygon::CastToCurvePolygon (OGRPolygon * *poPoly*) [static], [protected]

Cast to curve polygon.

The passed in geometry is consumed and a new one returned .

Parameters

poPoly	the input geometry - ownership is passed to the method.

Returns

new geometry.

References OGRGeometry::assignSpatialReference(), OGRLinearRing::CastToLineString(), OGRGeometry::get↩ CoordinateDimension(), OGRGeometry::getSpatialReference(), OGRCurvePolygon::OGRCurvePolygon(), and OGRCurvePolygon::setCoordinateDimension().

12.89.2.2 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. **??**).

References OGRGeometry::closeRings().

12.89.2.3 OGRPolygon * OGRPolygon::CurvePolyToPoly (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return a polygon from a curve polygon.

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

The returned geometry is a new instance whose ownership belongs to the caller.

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

Returns

a linestring

Since

OGR 2.0

Reimplemented from **OGRCurvePolygon** (p. **??**).

References OGRCurvePolygon::clone().

Referenced by OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPolygon(), and OGRGeometryFactory::forceToPolygon().

12.89.2.4 OGRErr OGRPolygon::exportToWkb (OGRwkbByteOrder *eByteOrder*, unsigned char * *pabyData*, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGR↩ Geometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRCurvePolygon** (p. **??**).

References OGRGeometry::getCoordinateDimension(), getGeometryType(), OGRGeometry::getIsoGeometry↩ Type(), and wkbVariantIso.

12.89.2.5 OGRErr OGRPolygon::exportToWkt (char ** *ppszDstText*, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRCurvePolygon** (p. **??**).

References CPLCalloc(), CPLDebug(), CPLStrdup(), OGRSimpleCurve::exportToWkt(), OGRGeometry::getCoordinateDimension(), getExteriorRing(), OGRSimpleCurve::getNumPoints(), OGRCurvePolygon::IsEmpty(), OGRSimpleCurve::setCoordinateDimension(), and wkbVariantIso.

Referenced by OGRGeometryFactory::organizePolygons().

12.89.2.6 OGRGeometry * OGRPolygon::getCurveGeometry (const char *const * *papszOptions* = NULL) const [virtual]

Return curve version of this geometry.

Returns a geometry that has possibly CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by de-approximating curve geometries.

If the geometry has no curve portion, the returned geometry will be a clone of it.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getLinearGeometry()** (p. **??**).

This function is the same as C function **OGR_G_GetCurveGeometry()** (p. **??**).

Parameters

papszOptions	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented from **OGRGeometry** (p. **??**).

References OGRCurvePolygon::addRingDirectly(), OGRGeometry::assignSpatialReference(), OGRCurvePolygon::clone(), OGRGeometry::getCurveGeometry(), OGRGeometry::getGeometryType(), OGRGeometry::getSpatialReference(), OGRCurvePolygon::OGRCurvePolygon(), wkbFlatten, and wkbLineString.

12.89.2.7 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 exportToWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceTo↩ Polygon(), and OGRGeometryFactory::organizePolygons().

12.89.2.8 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.

Reimplemented from **OGRCurvePolygon** (p. **??**).

12.89.2.9 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()** (p. **??**) 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 **OGRCurvePolygon** (p. **??**).

References wkbPolygon, and wkbPolygon25D.

Referenced by exportToWkb().

12.89.2.10 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

iRing	internal ring index from 0 to getNumInternalRings() - 1.

Returns

pointer to interior ring. May be NULL.

Referenced by OGRGeometryFactory::forceToMultiLineString().

12.89.2.11 OGRGeometry * OGRPolygon::getLinearGeometry (double *dfMaxAngleStepSizeDegrees* = 0, const char *const * *papszOptions* = NULL) const [virtual]

Return, possibly approximate, non-curve version of this geometry.

Returns a geometry that has no CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by approximating curve geometries.

The ownership of the returned geometry belongs to the caller.

The reverse method is **OGRGeometry::getCurveGeometry()** (p. **??**).

This method is the same as the C function **OGR_G_GetLinearGeometry()** (p. **??**).

Parameters

dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings. See **OGRGeometryFactory::curveToLine↩ String()** (p. **??**) for valid options.

Returns

a new geometry.

Since

GDAL 2.0

Reimplemented from **OGRCurvePolygon** (p. **??**).

References OGRGeometry::getLinearGeometry().

12.89.2.12 OGRBoolean OGRPolygon::hasCurveGeometry (int *bLookForNonLinear* = FALSE) const [virtual]

Returns if this geometry is or has curve geometry.

Returns if a geometry is, contains or may contain a CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **getLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **getLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This method is the same as the C function **OGR_G_HasCurveGeometry()** (p. **??**).

Parameters

bLookForNon↩ Linear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

Reimplemented from **OGRCurvePolygon** (p. **??**).

12.89.2.13 OGRErr OGRPolygon::importFromWkb (unsigned char * *pabyData,* int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRCurvePolygon** (p. **??**).

12.89.2.14 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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRCurvePolygon** (p. **??**).

12.89.2.15 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

poPoint	point to be set with an internal point.

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Reimplemented from **OGRCurvePolygon** (p. **??**).

References OGRPoint::empty(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::IsEmpty(), OGR_G_PointOnSurface(), OGRPoint::setX(), and OGRPoint::setY().

Referenced by OGRCurvePolygon::PointOnSurface().

12.89.2.16 OGRLinearRing * OGRPolygon::stealExteriorRing ( )

"Steal" reference to external polygon ring.

After the call to that function, only call to **stealInteriorRing()** (p. **??**) or destruction of the **OGRPolygon** (p. **??**) is valid. Other operations may crash.

Returns

pointer to external ring. May be NULL if the **OGRPolygon** (p. **??**) is empty.

References OGRCurvePolygon::stealExteriorRingCurve().

Referenced by OGRGeometryFactory::forceToPolygon().

12.89.2.17 OGRLinearRing * OGRPolygon::stealInteriorRing (int *iRing*)

"Steal" reference to indicated interior ring.

After the call to that function, only call to **stealInteriorRing()** (p. **??**) or destruction of the **OGRPolygon** (p. **??**) is valid. Other operations may crash.

Parameters

iRing	internal ring index from 0 to getNumInternalRings() - 1.

Returns

pointer to interior ring. May be NULL.

Referenced by OGRGeometryFactory::forceToPolygon().

12.89.2.18 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.

Reimplemented from **OGRCurvePolygon** (p. **??**).

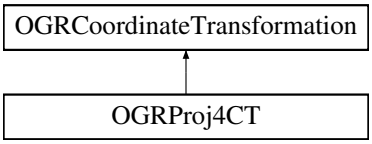
References OGRGeometry::getCoordinateDimension().

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

- **ogr_geometry.h**
- ogrpolygon.cpp

12.90 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

12.90.1 Member Function Documentation

12.90.1.1 OGRSpatialReference ∗ OGRProj4CT::GetSourceCS ( ) [virtual]

Fetch internal source coordinate system.

Implements **OGRCoordinateTransformation** (p. **??**).

12.90.1.2 OGRSpatialReference ∗ OGRProj4CT::GetTargetCS ( ) [virtual]

Fetch internal target coordinate system.

Implements **OGRCoordinateTransformation** (p. **??**).

12.90.1.3 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

nCount	number of points to transform.
x	array of nCount X vertices, modified in place.
y	array of nCount Y vertices, modified in place.
z	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().

12.90.1.4 int OGRProj4CT::TransformEx (int *nCount*, double * *x*, double * *y*, double * *z* = NULL, int * *pabSuccess* = NULL) [virtual]

Transform points from source to destination space.

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

Parameters

nCount	number of points to transform.
x	array of nCount X vertices, modified in place.
y	array of nCount Y vertices, modified in place.
z	array of nCount Z vertices, modified in place.
pabSuccess	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

12.91 OGRProj4Datum Struct Reference

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

- ogr_srs_proj4.cpp

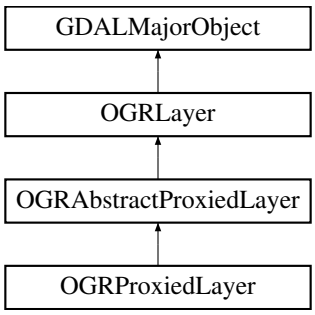
12.92 OGRProj4PM Struct Reference

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

- ogr_srs_proj4.cpp

12.93 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 void **SetSpatialFilter** (int iGeomField, **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** (GIntBig nIndex)

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

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

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

  Create and write a new feature within a layer.
- virtual OGRErr **DeleteFeature** (GIntBig 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 GIntBig **GetFeatureCount** (int bForce=TRUE)

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

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

- 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.

Additional Inherited Members

12.93.1 Member Function Documentation

12.93.1.1 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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_AlterFieldDefn()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::AlterFieldDefn().

12.93.1.2 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 existance that were obtained or created with the previous layer definition.

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

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

This function is the same as the C function **OGR_L_CreateField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::CreateField().

12.93.1.3 OGRErr OGRProxiedLayer::DeleteFeature (GIntBig *nFID*) [virtual]

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The **TestCapability()** (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

nFID	the feature id to be deleted from the layer

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::DeleteFeature().

12.93.1.4 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 existance that were obtained or created with the previous layer definition.

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

This function is the same as the C function **OGR_L_DeleteField()** (p. **??**).

Parameters

iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::DeleteField().

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

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

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

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (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.

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

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

Parameters

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetExtent().

12.93.1.6 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

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetExtent().

12.93.1.7 OGRFeature * OGRProxiedLayer::GetFeature (GIntBig *nFID*) [virtual]

Fetch a feature by its identifier.

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

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (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 (with **GetNextFeature()** (p. **??**)) 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

nFID	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().

12.93.1.8 GIntBig 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. **??**).

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

Parameters

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

Returns

feature count, -1 if count not known.

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetFeatureCount().

12.93.1.9 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().

12.93.1.10 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.

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

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

Returns

geometry column name.

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetGeometryColumn().

12.93.1.11 OGRwkbGeometryType OGRProxiedLayer::GetGeomType ( ) [virtual]

Return the layer geometry type.

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

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

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

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

Returns

the geometry type

Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetGeomType(), and wkbUnknown.

12.93.1.12 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().

12.93.1.13 const char * OGRProxiedLayer::GetName ( ) [virtual]

Return the layer name.

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

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetName().

12.93.1.14 OGRFeature * OGRProxiedLayer::GetNextFeature ( ) [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity 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.

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

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().

12.93.1.15 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().

12.93.1.16 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.

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

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().

12.93.1.17 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().

12.93.1.18 OGRErr OGRProxiedLayer::ICreateFeature (OGRFeature * *poFeature*) [virtual]

Create and write a new feature within a layer.

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

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::CreateFeature().

12.93.1.19 OGRErr OGRProxiedLayer::ISetFeature (OGRFeature * *poFeature*) [virtual]

Rewrite an existing feature.

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

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

Parameters

poFeature	the feature to write.

Returns

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

Since

GDAL 2.0

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetFeature().

12.93.1.20 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 existance that were obtained or created with the previous layer definition.

panMap is such that,for each field definition at position i after reordering, its position before reordering was pan↩ Map[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

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::ReorderFields().

12.93.1.21 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().

12.93.1.22 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 interprete the WHERE clause in which case the capabilities will be broader than those of OGR SQL.

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

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

Parameters

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

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetAttributeFilter().

12.93.1.23 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

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

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetIgnoredFields().

12.93.1.24 OGRErr OGRProxiedLayer::SetNextByIndex (GIntBig *nIndex*) [virtual]

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

This method allows positioning of a layer such that the **GetNextFeature()** (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

nIndex	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().

12.93.1.25 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

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

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetSpatialFilter().

12.93.1.26 void OGRProxiedLayer::SetSpatialFilter (int *iGeomField,* OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**Get↩ SpatialRef()** (p. **??**)). In the future this may be generalized.

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

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

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

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetSpatialFilter().

12.93.1.27 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

poStyleTable	pointer to style table to set

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetStyleTable().

12.93.1.28 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

poStyleTable	pointer to style table to set

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::SetStyleTableDirectly().

12.93.1.29 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() 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().

12.93.1.30 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.
- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using **CreateGeomField()** (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.
- **OLCCurveGeometries** / "CurveGeometries": TRUE if this layer supports writing curve geometries or may return such geometries. (GDAL 2.0).

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

Parameters

pszCap	the name of the capability to test.

Returns

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

Implements **OGRLayer** (p. **??**).

References OGRLayer::TestCapability().

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

- ogrlayerpool.h
- ogrlayerpool.cpp

12.94 OGRRawPoint Class Reference

`#include <ogr_geometry.h>`

12.94.1 Detailed Description

Simple container for a position.

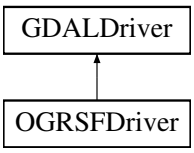
The documentation for this class was generated from the following file:

- **ogr_geometry.h**

12.95 OGRSFDriver Class Reference

`#include <ogrsf_frmts.h>`

Inheritance diagram for OGRSFDriver:



12.95.1 Detailed Description

LEGACY class. Use GDALDriver in your new code ! This class may be removed in a later release.

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. **??**).

NOTE: Starting with GDAL 2.0, it is *NOT* safe to cast the handle of a C function that returns a OGRSFDriverH to a OGRSFDriver∗. If a C++ object is needed, the handle should be cast to GDALDriver∗.

Deprecated

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

- **ogrsf_frmts.h**
- ogrsfdriver.cpp

12.96 OGRSFDriverRegistrar Class Reference

`#include <ogrsf_frmts.h>`

12.96.1 Detailed Description

LEGACY class. Use GDALDriverManager in your new code ! This class may be removed in a later release.

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 **OGR↩ CleanupAll()** (p. **??**).

Deprecated

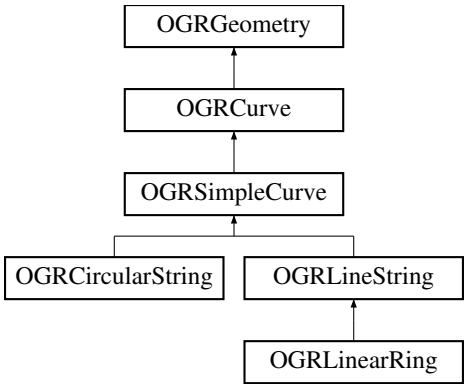
The documentation for this class was generated from the following files:

- **ogrsf_frmts.h**
- ogrsfdriverregistrar.cpp

12.97 OGRSimpleCurve Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRSimpleCurve:



Public Member Functions

- virtual int **WkbSize** () const

  Returns size of related binary representation.
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1, **OGRwkbVariant**=**wkbVariantOldOgc**)

  Assign geometry from well known binary data.
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *, **OGRwkbVariant**=**wkbVariantOldOgc**) 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, **OGRwkbVariant**=**wkbVariantOldOgc**) const

  Convert a geometry into well known text format.
- 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.
- virtual double **Project** (const **OGRPoint** *) const

  Project point on linestring.
- virtual **OGRLineString** * **getSubLine** (double, double, int) const

  Get the portion of linestring.
- virtual 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 nNewPointCount, int bZeroizeNewContent=TRUE)

  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 *padfZIn=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 **OGRPointIterator** * **getPointIterator** () const

  Returns a point iterator over the curve.
- 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.

Protected Member Functions

- virtual double **get_LinearArea** () const

  Compute area of ring / closed linestring.

Friends

- class **OGRGeometry**

Additional Inherited Members

12.97.1 Detailed Description

Abstract curve base class for **OGRLineString** (p. **??**) and **OGRCircularString** (p. **??**)

Note: this class does not exist in SQL/MM standard and exists for implementation convenience.

Since

GDAL 2.0

12.97.2 Member Function Documentation

12.97.2.1 void OGRSimpleCurve::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

poPoint	the point to assign to the new vertex.

References OGRPoint::getCoordinateDimension(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), and setPoint().

Referenced by OGRLinearRing::closeRings(), OGRGeometryFactory::curveFromLineString(), OGRGeometryFactory::curveToLineString(), getSubLine(), OGRBuildPolygonFromEdges(), and OGRLayer::SetSpatialFilterRect().

12.97.2.2 void OGRSimpleCurve::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

x	the X coordinate to assign to the new point.
y	the Y coordinate to assign to the new point.
z	the Z coordinate to assign to the new point (defaults to zero).

References setPoint().

12.97.2.3 void OGRSimpleCurve::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

poOtherLine	the other **OGRLineString** (p. **??**).
nStartVertex	the first vertex to copy, defaults to 0 to start with the first vertex in the other linestring.
nEndVertex	the last vertex to copy, defaults to -1 indicating the last vertex of the other line string.

References getNumPoints(), and setNumPoints().

Referenced by OGRCircularString::CurveToLine(), OGRGeometryFactory::forceToLineString(), and OGR↩ GeometryFactory::forceToMultiLineString().

12.97.2.4 OGRGeometry * OGRSimpleCurve::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(), OGRGeometryFactory::createGeometry(), OGR↩ Geometry::getCoordinateDimension(), OGRGeometry::getGeometryType(), OGRGeometry::getSpatialReference(), setCoordinateDimension(), and setPoints().

Referenced by OGRGeometryFactory::curveFromLineString(), and OGRLineString::CurveToLine().

12.97.2.5 void OGRSimpleCurve::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 OGRCircularString::importFromWkb(), and OGRCircularString::importFromWkt().

12.97.2.6 void OGRSimpleCurve::EndPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve end point.

This method relates to the SF COM ICurve::get_EndPoint() method.

Parameters

poPoint	the point to be assigned the end location.

Implements **OGRCurve** (p. **??**).

References getPoint().

Referenced by OGRGeometryFactory::forceToLineString(), Value(), and OGRCircularString::Value().

12.97.2.7 OGRBoolean OGRSimpleCurve::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(), getNumPoints(), getX(), getY(), getZ(), OGRGeometry::IsEmpty(), and IsEmpty().

12.97.2.8 OGRErr OGRSimpleCurve::exportToWkb (OGRwkbByteOrder *eByteOrder,* unsigned char * *pabyData,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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. **??**) or **OGR_G_ExportToIsoWkb()** (p. **??**), depending on the value of eWkbVariant.

Parameters

eByteOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyData	a buffer into which the binary representation is written. This buffer must be at least **OGR↩ Geometry::WkbSize()** (p. **??**) byte in size.
eWkbVariant	What standard to use when exporting geometries with three dimensions (or more). The default wkbVariantOldOgc is the historical OGR variant. wkbVariantIso is the variant defined in ISO SQL/MM and adopted by OGC for SFSQL 1.2.

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRCircularString** (p. **??**), and **OGRLinearRing** (p. **??**).

References OGRGeometry::getCoordinateDimension(), OGRGeometry::getGeometryType(), OGRGeometry↩ ::getIsoGeometryType(), wkbFlatten, wkbHasZ, wkbVariantIso, and wkbVariantPostGIS1.

Referenced by OGRCircularString::exportToWkb().

12.97.2.9 OGRErr OGRSimpleCurve::exportToWkt (char ** *ppszDstText,* OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) 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

ppszDstText	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().

eWkbVariant	the specification that must be conformed too : • wbkVariantOgc for old-style 99-402 extended dimension (Z) WKB types • wbkVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3

Returns

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRCircularString** (p. **??**).

References CPLDebug(), CPLStrdup(), OGRGeometry::getCoordinateDimension(), OGRGeometry::getGeometryName(), IsEmpty(), and wkbVariantIso.

Referenced by OGRCircularString::exportToWkt(), and OGRPolygon::exportToWkt().

12.97.2.10 void OGRSimpleCurve::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. **??**).

12.97.2.11 double OGRSimpleCurve::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. **??**).

Reimplemented in **OGRCircularString** (p. **??**).

Referenced by getSubLine().

12.97.2.12 double OGRSimpleCurve::get_LinearArea ( ) const [protected], [virtual]

Compute area of ring / closed linestring.

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 OGRLineString::get_Area(), and OGRCircularString::get_Area().

12.97.2.13 void OGRSimpleCurve::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

psEnvelope	the structure in which to place the results.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRCircularString** (p. **??**).

References IsEmpty().

Referenced by getEnvelope(), and OGRCircularString::getEnvelope().

12.97.2.14 void OGRSimpleCurve::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

psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRCircularString** (p. **??**).

References getEnvelope(), and IsEmpty().

12.97.2.15 int OGRSimpleCurve::getNumPoints ( ) const [inline], [virtual]

Fetch vertex count.

Returns the number of vertices in the line string.

Returns

vertex count.

Implements **OGRCurve** (p. **??**).

Referenced by addSubLineString(), OGRGeometryFactory::curveFromLineString(), OGRGeometryFactory::curveToLineString(), OGRGeometry::dumpReadable(), Equals(), OGRPolygon::exportToWkt(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRCircularString::get_AreaOfCurveSegments(), OGRSimpleCurvePointIterator::getNextPoint(), getSubLine(), OGR_G_GetPoint(), OGR_G_GetPoints(), OGR_G_GetX(), OGR_G_GetY(), OGR_G_GetZ(), OGRBuildPolygonFromEdges(), and OGRGeometryFactory::organizePolygons().

12.97.2.16 void OGRSimpleCurve::getPoint (int *i*, OGRPoint * *poPoint*) const

Fetch a point in line string.

This method relates to the SFCOM ILineString::get_Point() method.

Parameters

i	the vertex to fetch, from 0 to **getNumPoints()** (p. **??**)-1.
poPoint	a point to initialize with the fetched point.

References OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGRPoint::setY(), and OGRPoint::setZ().

Referenced by OGRLinearRing::closeRings(), OGRGeometryFactory::curveFromLineString(), EndPoint(), OGRSimpleCurvePointIterator::getNextPoint(), OGRGeometryFactory::organizePolygons(), and StartPoint().

12.97.2.17 OGRPointIterator * OGRSimpleCurve::getPointIterator ( ) const [virtual]

Returns a point iterator over the curve.

The curve must not be modified while an iterator exists on it.

The iterator must be destroyed with **OGRPointIterator::destroy()** (p. **??**).

Returns

a point iterator over the curve.

Since

GDAL 2.0

Implements **OGRCurve** (p. **??**).

12.97.2.18 void OGRSimpleCurve::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

paoPointsOut	a buffer into which the points is written.
padfZ	the Z values that go with the points (optional, may be NULL).

Referenced by getPoints(), and OGR_G_GetPoints().

12.97.2.19 void OGRSimpleCurve::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

pabyX	a buffer of at least (sizeof(double) * nXStride * nPointCount) bytes, may be NULL.

nXStride	the number of bytes between 2 elements of pabyX.
pabyY	a buffer of at least (sizeof(double) ∗ nYStride ∗ nPointCount) bytes, may be NULL.
nYStride	the number of bytes between 2 elements of pabyY.
pabyZ	a buffer of at last size (sizeof(double) ∗ nZStride ∗ nPointCount) bytes, may be NULL.
nZStride	the number of bytes between 2 elements of pabyZ.

Since

OGR 1.9.0

References getPoints().

12.97.2.20 OGRLineString ∗ OGRSimpleCurve::getSubLine (double *dfDistanceFrom,* double *dfDistanceTo,* int *bAsRatio*) const [virtual]

Get the portion of linestring.

The portion of the linestring extracted to new one. The input distances (maybe present as ratio of length of linestring) set begin and end of extracted portion.

Parameters

dfDistanceFrom	The distance from the origin of linestring, where the subline should begins
dfDistanceTo	The distance from the origin of linestring, where the subline should ends
bAsRatio	The flag indicating that distances are the ratio of the linestring length.

Returns

a newly allocated linestring now owned by the caller, or NULL on failure.

Since

OGR 1.11.0

References addPoint(), OGRGeometry::assignSpatialReference(), CPLError(), get_Length(), OGRGeometry::getCoordinateDimension(), getNumPoints(), OGRGeometry::getSpatialReference(), and setCoordinateDimension().

12.97.2.21 double OGRSimpleCurve::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

iVertex	the vertex to return, between 0 and **getNumPoints()** (p. **??**)-1.

Returns

X value.

Referenced by OGRLinearRing::closeRings(), OGRGeometryFactory::curveFromLineString(), OGRGeometryFactory::curveToLineString(), Equals(), OGRCircularString::get_AreaOfCurveSegments(), OGR_G_GetPoint(), OGR_G_GetX(), and OGRBuildPolygonFromEdges().

12.97.2.22 double OGRSimpleCurve::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

iVertex	the vertex to return, between 0 and **getNumPoints()** (p. **??**)-1.

Returns

X value.

Referenced by OGRLinearRing::closeRings(), OGRGeometryFactory::curveFromLineString(), OGRGeometryFactory::curveToLineString(), Equals(), OGRCircularString::get_AreaOfCurveSegments(), OGR_G_GetPoint(), OGR_G_GetY(), and OGRBuildPolygonFromEdges().

12.97.2.23 double OGRSimpleCurve::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

iVertex	the vertex to return, between 0 and **getNumPoints()** (p. **??**)-1.

Returns

Z value.

Referenced by OGRLinearRing::closeRings(), Equals(), OGR_G_GetPoint(), OGR_G_GetZ(), and OGRBuildPolygonFromEdges().

12.97.2.24 OGRErr OGRSimpleCurve::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1, OGRwkbVariant *eWkbVariant* = wkbVariantOldOgc) [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

pabyData	the binary input data.
nSize	the size of pabyData in bytes, or zero if not known.
eWkbVariant	if wkbVariantPostGIS1, special interpretation is done for curve geometries code

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 **OGRCircularString** (p. **??**), and **OGRLinearRing** (p. **??**).

References CPLError(), OGRGeometry::getCoordinateDimension(), and setNumPoints().

Referenced by OGRCircularString::importFromWkb().

12.97.2.25 OGRErr OGRSimpleCurve::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

ppszInput	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. **??**).

Reimplemented in **OGRCircularString** (p. **??**).

Referenced by OGRCircularString::importFromWkt().

12.97.2.26 OGRBoolean OGRSimpleCurve::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 Equals(), exportToWkt(), OGRCircularString::get_Area(), and getEnvelope().

12.97.2.27 double OGRSimpleCurve::Project (const OGRPoint * *poPoint*) const [virtual]

Project point on linestring.

The input point projeted on linestring. This is the shortest distance from point to the linestring. The distance from begin of linestring to the point projection returned.

This method is built on the GEOS library (GEOS >= 3.2.0), check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always return -1, issuing a CPLE_NotSupported error.

Returns

a distance from the begin of the linestring to the projected point.

Since

OGR 1.11.0

References CPLError().

12.97.2.28 void OGRSimpleCurve::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::curveToLineString(), OGRGeometryFactory::forceToLineString(), segmentize(), and OGRCircularString::segmentize().

12.97.2.29 void OGRSimpleCurve::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Add intermediate vertices to a geometry.

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

dfMaxLength	the maximum distance between 2 points after segmentization

This method modifies the geometry to add intermediate vertices if necessary so that the maximum length between 2 consecutives vertices is lower than dfMaxLength.

Parameters

dfMaxLength	maximum length between 2 consecutives vertices.

Reimplemented from **OGRGeometry** (p. **??**).

Reimplemented in **OGRCircularString** (p. **??**).

References CPLError(), OGRGeometry::getCoordinateDimension(), and reversePoints().

12.97.2.30 void OGRSimpleCurve::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

nNewDimension	New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. **??**).

Referenced by clone(), OGRPolygon::exportToWkt(), and getSubLine().

12.97.2.31 void OGRSimpleCurve::setNumPoints (int *nNewPointCount*, int *bZeroizeNewContent* = TRUE)

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

nNewPointCount	the new number of points for geometry.

References CPLError(), and OGRGeometry::getCoordinateDimension().

Referenced by addSubLineString(), empty(), OGRCompoundCurve::get_Area(), importFromWkb(), OGR_G_SetPointCount(), OGR_G_SetPoints(), setPoint(), and setPoints().

12.97.2.32 void OGRSimpleCurve::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 accommodate the request.

There is no SFCOM analog to this method.

Parameters

iPoint	the index of the vertex to assign (zero based).
poPoint	the value to assign to the vertex.

References OGRPoint::getCoordinateDimension(), OGRPoint::getX(), OGRPoint::getY(), and OGRPoint::getZ().

Referenced by addPoint(), OGRGeometryFactory::approximateArcAngles(), OGRGeometryFactory::curveToLineString(), OGRCompoundCurve::get_Area(), and OGR_G_SetPoints().

12.97.2.33 void OGRSimpleCurve::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 accommodate the request.

There is no SFCOM analog to this method.

Parameters

iPoint	the index of the vertex to assign (zero based).
xIn	input X coordinate to assign.
yIn	input Y coordinate to assign.
zIn	input Z coordinate to assign (defaults to zero).

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

12.97.2.34 void OGRSimpleCurve::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

nPointsIn	number of points being passed in paoPointsIn
paoPointsIn	list of points being assigned.
padfZ	the Z values that go with the points (optional, may be NULL).

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

Referenced by clone(), OGRLinearRing::clone(), OGR_G_SetPoints(), and transform().

12.97.2.35 void OGRSimpleCurve::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

nPointsIn	number of points being passed in padfX and padfY.
padfX	list of X coordinates of points being assigned.
padfY	list of Y coordinates of points being assigned.
padfZ	list of Z coordinates of points being assigned (defaults to NULL for 2D objects).

References setNumPoints().

12.97.2.36 void OGRSimpleCurve::StartPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve start point.

This method relates to the SF COM ICurve::get_StartPoint() method.

Parameters

poPoint	the point to be assigned the start location.

Implements **OGRCurve** (p. **??**).

References getPoint().

Referenced by OGRGeometryFactory::forceToLineString(), Value(), and OGRCircularString::Value().

12.97.2.37 void OGRSimpleCurve::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. **??**).

12.97.2.38 OGRErr OGRSimpleCurve::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

poCT	the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. **??**).

References OGRGeometry::assignSpatialReference(), CPLError(), CPLGetConfigOption(), CSLTestBoolean(), O↩ GRCoordinateTransformation::GetTargetCS(), setPoints(), and OGRCoordinateTransformation::TransformEx().

12.97.2.39 void OGRSimpleCurve::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.

This function is the same as the C function **OGR_G_Value()** (p. **??**).

Parameters

dfDistance	distance along the curve at which to sample position. This distance should be between zero and **get_Length()** (p. **??**) for this curve.
poPoint	the point to be assigned the curve position.

Implements **OGRCurve** (p. **??**).

Reimplemented in **OGRCircularString** (p. **??**).

References EndPoint(), OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGRPoint::setY(), OGR↩ Point::setZ(), and StartPoint().

12.97.2.40 int OGRSimpleCurve::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. **??**).

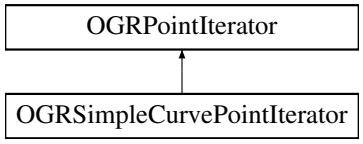
References OGRGeometry::getCoordinateDimension().

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

- **ogr_geometry.h**
- ogrlinestring.cpp

12.98 OGRSimpleCurvePointIterator Class Reference

Inheritance diagram for OGRSimpleCurvePointIterator:



Public Member Functions

- virtual OGRBoolean **getNextPoint** (**OGRPoint** *p)

  Returns the next point followed by the iterator.

Additional Inherited Members

12.98.1 Member Function Documentation

12.98.1.1 OGRBoolean OGRSimpleCurvePointIterator::getNextPoint (OGRPoint * p) [virtual]

Returns the next point followed by the iterator.

Parameters

p	point to fill.

Returns

TRUE in case of success, or FALSE if the end of the curve is reached.

Since

GDAL 2.0

Implements **OGRPointIterator** (p. **??**).

References OGRSimpleCurve::getNumPoints(), and OGRSimpleCurve::getPoint().

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

- ogrlinestring.cpp

12.99 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 **exportToMICoordSys** (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 ∗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 **importFromMICoordSys** (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 eXAxis↩Orientation, 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 ∗pszEllipsoid↩ Name, double dfSemiMajor, double dfInvFlattening, const char ∗pszPMName=NULL, double dfPM↩ Offset=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 nVertDatum↩ Class=2005)

  Set the user visible VERT_CS name.
- OGRErr **SetCompoundCS** (const char ∗pszName, const **OGRSpatialReference** ∗poHorizSRS, const **O**↩ **GRSpatialReference** ∗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 ∗=NULL) const

  Get spheroid semi major axis.

- double **GetSemiMinor** (OGRErr *=NULL) const

  Get spheroid semi minor axis.
- double **GetInvFlattening** (OGRErr *=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 *=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 *=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 dfPseudoStd↩ ParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLAEA** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalse↩ Northing)
- OGRErr **SetLCC** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double df↩ FalseEasting, 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 df↩ FalseEasting, 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 dfFalse↩ Northing)
- OGRErr **SetOS** (double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double df↩ FalseNorthing)
- OGRErr **SetOrthographic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double df↩ FalseNorthing)
- OGRErr **SetPolyconic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalse↩ Northing)
- 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 dfFalse↩ Easting, double dfFalseNorthing)
- OGRErr **SetSOC** (double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double df↩ FalseNorthing)
- OGRErr **SetTM** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMVariant** (const char ∗pszVariantName, double dfCenterLat, double dfCenterLong, double df↩ Scale, 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 **SetQSC** (double dfCenterLat, double dfCenterLong)
- 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.*
- static **OGRSpatialReference** * **GetWGS84SRS** ()

  Returns an instance of a SRS object with WGS84 WKT.

12.99.1 Detailed Description

This class respresents 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.

12.99.2 Constructor & Destructor Documentation

12.99.2.1 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

pszWKT	well known text definition to which the object should be initialized, or NULL (the default).

References importFromWkt().

12.99.2.2 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

12.99.3 Member Function Documentation

12.99.3.1 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 **CPLError()** (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().

12.99.3.2 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().

12.99.3.3 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().

12.99.3.4 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().

12.99.3.5 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

poSrcSRS	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().

12.99.3.6 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().

12.99.3.7 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

poSRS	the object to delete

Since

GDAL 1.7.0

12.99.3.8 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().

12.99.3.9 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().

12.99.3.10 OGRErr OGRSpatialReference::exportToERM (char * *pszProj,* char * *pszDatum,* char * *pszUnits*)

Convert coordinate system to ERMapper format.

Parameters

pszProj	32 character buffer to receive projection name.
pszDatum	32 character buffer to recieve datum name.
pszUnits	32 character buffer to receive units name.

Returns

OGRERR_NONE on success, OGRERR_SRS_UNSUPPORTED if not translation is found, or OGRERR_F↩AILURE on other failures.

References GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), GetLinearUnits(), GetUTMZone(), import↩FromDict(), IsGeographic(), and IsProjected().

12.99.3.11 OGRErr OGRSpatialReference::exportToMICoordSys (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 **OSRExportToMICoordSys()** (p. **??**).

Parameters

ppszResult	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().

12.99.3.12 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

piProjSys	Pointer to variable, where the projection system code will be returned.
piDatum	Pointer to variable, where the coordinate system code will be returned.
piEllips	Pointer to variable, where the spheroid code will be returned.
piZone	Pointer to variable, where the zone for UTM projection system will be returned.
padfPrjParams	an existing 7 double buffer into which the projection parameters will be placed. See **importFromPanorama()** (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().

12.99.3.13 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 equivelent of the C function **OSRExportToPCI()** (p. **??**).

Parameters

ppszProj	pointer to which dynamically allocated PCI projection definition will be assigned.
ppszUnits	pointer to which dynamically allocated units definition will be assigned.
ppadfPrjParams	pointer to which dynamically allocated array of 17 projection parameters will be assigned. See **importFromPCI()** (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(), IsLocal(), and OSRCalcSemiMinorFromInvFlattening().

12.99.3.14 OGRErr OGRSpatialReference::exportToPrettyWkt (char ** *ppszResult,* int *bSimplify* = FALSE) const

Convert this SRS into a 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

ppszResult	the resulting string is returned in this pointer.
bSimplify	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().

12.99.3.15 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 equivelent of the C function **OSRExportToProj4()** (p. **??**).

Parameters

ppszProj4	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(), CPLsprintf(), 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().

12.99.3.16 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

piProjSys	Pointer to variable, where the projection system code will be returned.
piZone	Pointer to variable, where the zone for UTM and State Plane projection systems will be returned.
ppadfPrjParams	Pointer to which dynamically allocated array of 15 projection parameters will be assigned. See **importFromUSGS()** (p. **??**) for the list of parameters. Caller responsible to free this array.
piDatum	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().

12.99.3.17 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

ppszResult	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(), and Validate().

12.99.3.18 OGRErr OGRSpatialReference::exportToXML (char **, const char * = 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 equivelent of the C function **OSRExportToXML()** (p. **??**).

Parameters

ppszRawXML	pointer to which dynamically allocated XML definition will be assigned.
pszDialect	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().

12.99.3.19 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

pszParameter	projection parameter to look for
poPROJCS	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_SRS↩ Node::GetValue().

Referenced by GetProjParm(), morphFromESRI(), and morphToESRI().

12.99.3.20 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 amoung 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 Set↩ LinearUnits().

Referenced by morphToESRI().

12.99.3.21 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(), importFromProj4(), importFromUSGS(), morphFromESRI(), and morphToESRI().

12.99.3.22 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

ppszName	a pointer to be updated with the pointer to the units name. The returned value remains internal to the **OGRSpatialReference** (p. **??**) and shouldn't be freed, or modified. It may be invalidated on the next **OGRSpatialReference** (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().

12.99.3.23 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

pszNodePath	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(), importFromEP↩SG(), importFromESRI(), importFromProj4(), IsGeographic(), IsProjected(), IsSame(), IsVertical(), morphFromES↩RI(), morphToESRI(), SetAngularUnits(), SetAuthority(), SetGeocCS(), SetGeogCS(), SetLinearUnitsAndUpdate↩Parameters(), SetLocalCS(), SetProjCS(), SetProjection(), SetProjParm(), SetStatePlane(), SetTargetLinearUnits(), SetTOWGS84(), and SetVertCS().

12.99.3.24 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

pszNodeName	the tree node to look for (case insensitive).
iAttr	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_SRS↩Node::GetValue().

Referenced by exportToERM(), exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetUTM↩Zone(), IsSame(), IsSameGeogCS(), IsSameVertCS(), morphFromESRI(), morphToESRI(), and SetUTM().

12.99.3.25 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

pszTargetKey	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().

12.99.3.26 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

pszTargetKey	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().

12.99.3.27 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

pszTargetKey	the coordinate system part to query ("PROJCS" or "GEOGCS").
iAxis	the axis to query (0 for first, 1 for second).
peOrientation	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().

12.99.3.28 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

pszTargetKey	the name or path to the parent node of the EXTENSION.
pszName	the name of the extension being fetched.
pszDefault	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().

12.99.3.29 double OGRSpatialReference::GetInvFlattening (OGRErr * *pnErr* = NULL) const

Get spheroid inverse flattening.

This method does the same thing as the C function **OSRGetInvFlattening()** (p. **??**).

Parameters

pnErr	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(), morphFromESRI(), and morphToESRI().

12.99.3.30 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

ppszName	a pointer to be updated with the pointer to the units name. The returned value remains internal to the **OGRSpatialReference** (p. **??**) and shouldn't be freed, or modified. It may be invalidated on the next **OGRSpatialReference** (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().

12.99.3.31 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

pszName	the name of the parameter to fetch, from the set of SRS_PP codes in **ogr_srs_api.h** (p. **??**).
dfDefaultValue	the value to return if this parameter doesn't exist.

pnErr	place to put error code on failure. Ignored if NULL.

Returns

value of parameter.

References GetProjParm().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetUTMZone(), morphTo↩ESRI(), and SetStatePlane().

12.99.3.32 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

ppszName	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 OG↩R_SRSNode::GetValue().

Referenced by morphFromESRI().

12.99.3.33 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

pszName	the name of the parameter to fetch, from the set of SRS_PP codes in **ogr_srs_api.h** (p. **??**).
dfDefaultValue	the value to return if this parameter doesn't exist.
pnErr	place to put error code on failure. Ignored if NULL.

Returns

value of parameter.

References CPLAtof(), FindProjParm(), GetAttrNode(), OGR_SRSNode::GetChild(), and OGR_SRSNode::Get↩Value().

Referenced by GetNormProjParm(), GetUTMZone(), importFromProj4(), IsSame(), morphFromESRI(), morphTo↩ESRI(), and SetLinearUnitsAndUpdateParameters().

12.99.3.34 int OGRSpatialReference::GetReferenceCount ( ) const [inline]

Fetch current reference count.

Returns

the current reference count.

12.99.3.35 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

pnErr	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().

12.99.3.36 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

pnErr	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(), GetSemiMajor(), and OSRCalcSemiMinorFromInvFlattening().

Referenced by exportToProj4().

12.99.3.37 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

pszTargetKey	the key to look on. ie. "PROJCS" or "VERT_CS".
ppszName	a pointer to be updated with the pointer to the units name. The returned value remains internal to the **OGRSpatialReference** (p. **??**) and shouldn't be freed, or modified. It may be invalidated on the next **OGRSpatialReference** (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_S↩ RSNode::GetValue(), and IsVertical().

Referenced by GetLinearUnits().

12.99.3.38 OGRErr OGRSpatialReference::GetTOWGS84 (double * *padfCoeff,* int *nCoeffCount* = 7) const

Fetch TOWGS84 parameters, if available.

Parameters

padfCoeff	array into which up to 7 coefficients are placed.
nCoeffCount	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 OG↩ R_SRSNode::GetValue().

Referenced by exportToPCI(), and IsSameGeogCS().

12.99.3.39 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

pbNorth	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(), export↩ ToUSGS(), and morphToESRI().

12.99.3.40 OGRSpatialReference * OGRSpatialReference::GetWGS84SRS ( ) [static]

Returns an instance of a SRS object with WGS84 WKT.

The reference counter of the returned object is not increased by this operation.

Returns

instance.

Since

GDAL 2.0

12.99.3.41 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

pszURL	the URL string.

Returns

OGRERR_NONE on success or an error code.

References OGR_SRSNode::AddChild(), Clear(), OGR_SRSNode::Clone(), CPLError(), CPLMalloc(), CPL↩ Strdup(), OGR_SRSNode::GetValue(), importFromCRSURL(), and SetNode().

Referenced by importFromCRSURL(), and SetFromUserInput().

12.99.3.42 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

pszDictFile	the name of the dictionary file to load.
pszCode	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().

12.99.3.43 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

nCode	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::Strip↩ Nodes().

Referenced by importFromERM(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), morphFromESRI(), SetFromUserInput(), SetStatePlane(), and SetWellKnownGeogCS().

12.99.3.44 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

nCode	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(), Is↩ Geographic(), IsProjected(), and SetAuthority().

Referenced by importFromEPSG(), SetFromUserInput(), and SetWellKnownGeogCS().

12.99.3.45 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

pszProj	the projection name, such as "NUTM11" or "GEOGRAPHIC".
pszDatum	the datum name, such as "NAD83".
pszUnits	the linear units "FEET" or "METERS".

Returns

OGRERR_NONE on success or OGRERR_UNSUPPORTED_SRS if not found.

References Clear(), CopyGeogCSFrom(), CPLAtof(), importFromDict(), importFromEPSG(), IsLocal(), and SetLinearUnits().

12.99.3.46 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 equivelent 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 equilvelent of the C function **OSRImportFromESRI()** (p. **??**).

Parameters

papszPrj	NULL terminated list of strings containing the definition.

Returns

OGRERR_NONE on success or an error code in case of failure.

References CopyGeogCSFrom(), CPLAtof(), 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().

12.99.3.47 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

pszCoordSys	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().

12.99.3.48 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

papszLines	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(), CSL↩ Destroy(), CSLTokenizeString2(), FixupOrdering(), importFromEPSG(), IsLocal(), IsProjected(), SetACEA(), Set↩ GeogCS(), SetLCC(), SetLCC1SP(), SetLinearUnits(), SetLocalCS(), SetMercator(), SetSinusoidal(), SetTM(), SetTOWGS84(), and SetUTM().

12.99.3.49 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

iProjSys	Input projection system code, used in GIS "Panorama". Supported Projections 1: Gauss-Kruger (Transverse Mercator) 2: Lambert Conformal Conic 2SP 5: Stereographic 6: Azimuthal Equidistant (Postel) 8: Mercator 10: Polyconic 13: Polar Stereographic 15: Gnomonic 17: Universal Transverse Mercator (UTM) 18: Wagner I (Kavraisky VI) 19: Mollweide 20: Equidistant Conic 24: Lambert Azimuthal Equal Area 27: Equirectangular 28: Cylindrical Equal Area (Lambert) 29: International Map of the World Polyconic
iDatum	Input coordinate system. Supported Datums 1: Pulkovo, 1942 2: WGS, 1984 3: OSGB 1936 (British National Grid) 9: Pulkovo, 1995
iEllips	Input spheroid. Supported Spheroids 1: Krassovsky, 1940 2: WGS, 1972 3: International, 1924 (Hayford, 1909) 4: Clarke, 1880 5: Clarke, 1866 (NAD1927) 6: Everest, 1830 7: Bessel, 1841 8: Airy, 1830 9: WGS, 1984 (GPS)

padfPrjParams	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(), importFrom↩EPSG(), IsLocal(), IsProjected(), SetAE(), SetAuthority(), SetCEA(), SetEC(), SetEquirectangular(), SetGeogC↩S(), SetGnomonic(), SetIWMPolyconic(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMC(), Set↩Mercator(), SetMollweide(), SetPolyconic(), SetPS(), SetStereographic(), SetTM(), SetUTM(), SetWagner(), and SetWellKnownGeogCS().

12.99.3.50 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 **OSRImportFromPCI()** (p. **??**).

Parameters

pszProj	NULL terminated string containing the definition. Looks like "ppppppppppp Ennn" or "ppppppppppp Dnnn", where "ppppppppppp" is a projection code, "Ennn" is an ellipsoid code, "Dnnn" — a datum code.
pszUnits	Grid units code ("DEGREE" or "METRE"). If NULL "METRE" will be used.
padfPrjParams	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 suppliet 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(), CPL↩Strnlen(), CSLCount(), CSLDestroy(), FixupOrdering(), importFromEPSG(), IsGeographic(), IsLocal(), Is↩Projected(), OSRCalcInvFlattening(), SetACEA(), SetAE(), SetAngularUnits(), SetAuthority(), SetCS(), SetE↩C(), SetEquirectangular2(), SetGeogCS(), SetGnomonic(), SetHOM(), SetHOM2PNO(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLinearUnits(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetMC(), SetMercator(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStatePlane(), Set↩Stereographic(), SetTM(), SetTOWGS84(), SetUTM(), and SetVDG().

12.99.3.51 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 equivelents, 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 aso 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 +nodefs"]
      ]
```

This method is the equivalent of the C function **OSRImportFromProj4()** (p. **??**).

Parameters

pszProj4	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(), FixupOrdering(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), OGR_SRSNode::GetValue(), importFromEPSG(), IsGeocentric(), IsLocal(), IsProjected(), OSRCalcInvFlattening(), 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(), SetQSC(), SetRobinson(), SetSinusoidal(), SetStereographic(), SetTM(), SetTMSO(), SetTOWGS84(), SetTPED(), SetUTM(), SetVDG(), SetWagner(), and SetWellKnownGeogCS().

Referenced by importFromEPSGA(), and SetFromUserInput().

12.99.3.52 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

pszUrl	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 dowload data.

References CPLError(), CPLErrorReset(), CPLGetLastErrorNo(), CPLHTTPDestroyResult(), CPLHTTPFetch(), CPLHTTPResult::nDataLen, CPLHTTPResult::nStatus, CPLHTTPResult::pabyData, CPLHTTPResult::pszErrBuf, and SetFromUserInput().

Referenced by SetFromUserInput().

12.99.3.53 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

pszURN	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().

12.99.3.54 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

iProjSys	Input projection system code, used in GCTP.
iZone	Input zone for UTM and State Plane projection systems. For Southern Hemisphere UTM use a negative zone code. iZone ignored for all other projections.
padfPrjParams	Array of 15 coordinate system parameters. These parameters differs for different projections.

<h4>Projection Transformation Package Projection Parameters</h4>

Code & Projection Id	Array Element							
	0	1	2	3	4	5	6	7
0 Geographic								
1 U T M	Lon/Z	Lat/Z						
2 State Plane								
3 Albers Equal Area	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
4 Lambert Conformal C	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
5 Mercator	SMajor	SMinor			CentMer	TrueScale	FE	FN
6 Polar Stereographic	SMajor	SMinor			LongPol	TrueScale	FE	FN
7 Polyconic	SMajor	SMinor			CentMer	OriginLat	FE	FN
8 Equid. Conic A	SMajor	SMinor	STDPAR		CentMer	OriginLat	FE	FN
Equid. Conic B	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
9 Transverse Mercator	SMajor	SMinor	Factor		CentMer	OriginLat	FE	FN
10 Stereographic	Sphere				CentLon	CenterLat	FE	FN
11 Lambert Azimuthal	Sphere				CentLon	CenterLat	FE	FN
12 Azimuthal	Sphere				CentLon	CenterLat	FE	FN
13 Gnomonic	Sphere				CentLon	CenterLat	FE	FN
14 Orthographic	Sphere				CentLon	CenterLat	FE	FN
15 Gen. Vert. Near Per	Sphere		Height		CentLon	CenterLat	FE	FN
16 Sinusoidal	Sphere				CentMer		FE	FN
17 Equirectangular	Sphere				CentMer	TrueScale	FE	FN
18 Miller Cylindrical	Sphere				CentMer		FE	FN
19 Van der Grinten	Sphere				CentMer	OriginLat	FE	FN
20 Hotin Oblique Merc A	SMajor	SMinor	Factor			OriginLat	FE	FN
Hotin Oblique Merc B	SMajor	SMinor	Factor	AziAng	AzmthPt	OriginLat	FE	FN
21 Robinson	Sphere				CentMer		FE	FN
22 Space Oblique Merc A	SMajor	SMinor		IncAng	AscLong		FE	FN
Space Oblique Merc B	SMajor	SMinor	Satnum	Path			FE	FN
23 Alaska Conformal	SMajor	SMinor					FE	FN
24 Interrupted Goode	Sphere							
25 Mollweide	Sphere				CentMer		FE	FN
26 Interrupt Mollweide	Sphere							
27 Hammer	Sphere				CentMer		FE	FN
28 Wagner IV	Sphere				CentMer		FE	FN
29 Wagner VII	Sphere				CentMer		FE	FN
30 Oblated Equal Area	Sphere		Shapem	Shapen	CentLon	CenterLat	FE	FN

---------- | 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

iDatum	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]:

  semiminor = sqrt(1.0 - ES) ∗ 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

nUSGSAngle↩ Format	one of USGS_ANGLE_DECIMALDEGREES, USGS_ANGLE_PACKEDDMS, or USGS_A↩ NGLE_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(), OS↩ RCalcInvFlattening(), SetACEA(), SetAE(), SetAuthority(), SetEC(), SetEquirectangular2(), SetGeogCS(), Set↩ Gnomonic(), SetHOM(), SetHOM2PNO(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMC(), Set↩ Mercator(), SetMollweide(), SetOrthographic(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), Set↩ StatePlane(), SetStereographic(), SetTM(), SetUTM(), SetVDG(), SetWagner(), and SetWellKnownGeogCS().

12.99.3.55 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

ppszInput	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().

12.99.3.56 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

pszDefinition	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().

12.99.3.57 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

pszXML	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().

12.99.3.58 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().

12.99.3.59 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().

12.99.3.60 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().

12.99.3.61 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().

12.99.3.62 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().

12.99.3.63 int OGRSpatialReference::IsSame (const OGRSpatialReference * *poOtherSRS*) const

Do these two spatial references describe the same system ?

Parameters

poOtherSRS	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(), OGRGeomFieldDefn::IsSame(), OGRGeometry::SymDifference(), and OGRGeometry::Union().

12.99.3.64 int OGRSpatialReference::IsSameGeogCS (const OGRSpatialReference * *poOther*) const

Do the GeogCS'es match?

This method is the same as the C function **OSRIsSameGeogCS()** (p. **??**).

Parameters

poOther	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().

12.99.3.65 int OGRSpatialReference::IsSameVertCS (const OGRSpatialReference * *poOther*) const

Do the VertCS'es match?

This method is the same as the C function **OSRIsSameVertCS()** (p. **??**).

Parameters

poOther	the SRS being compared against.

Returns

TRUE if they are the same or FALSE otherwise.

References CPLAtof(), and GetAttrValue().

Referenced by IsSame().

12.99.3.66 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().

12.99.3.67 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 defintion (altough 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

	TOWGS84		Adds missing TOWGS84 parameters (necessary for datum transformations), based on named datum and spheroid values.

	DATUM		Adds EPSG AUTHORITY nodes and sets SPHEROID name to OGR spec.
	GEOGCS		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(), FindProjParm(), 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().

12.99.3.68 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 note 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(), FixupOrdering(), GetAngularUnits(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetInvFlattening(), GetLinearUnits(), GetNormProjParm(), GetProjParm(), GetUTMZone(), OGR_SRSNode::GetValue(), SetNode(), SetProjParm(), OGR_SRSNode::SetValue(), and StripCTParms().

12.99.3.69 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(), OGRUnionLayer::GetLayerDefn(), OGRProxiedLayer::↩ GetSpatialRef(), OGRUnionLayer::GetSpatialRef(), and OGRGeomFieldDefn::SetSpatialRef().

12.99.3.70 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(), and OGRGeomFieldDefn::SetSpatialRef().

12.99.3.71 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().

12.99.3.72 OGRErr OGRSpatialReference::SetAE (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Azimuthal Equidistant

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

12.99.3.73 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

pszUnitsName	the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. **??**) such as SRS_UA_DEGREE.
dfInRadians	the value to multiple by an angle in the indicated units to transform to radians. Some standard conversion factors can be found in **ogr_srs_api.h** (p. **??**).

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::Get↩ Child(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::SetValue().

Referenced by CloneGeogCS(), Fixup(), and importFromPCI().

12.99.3.74 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

pszTargetKey	the partial or complete path to the node to set an authority on. ie. "PROJCS", "GEOGCS" or "GEOGCS|UNIT".
pszAuthority	authority name, such as "EPSG".
nCode	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().

12.99.3.75 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

pszTargetKey	either "PROJCS" or "GEOGCS", must already exist in SRS.
pszXAxisName	name of first axis, normally "Long" or "Easting".
eXAxisOrientation	normally OAO_East.
pszYAxisName	name of second axis, normally "Lat" or "Northing".
eYAxisOrientation	normally OAO_North.

Returns

OGRERR_NONE on success or an error code.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), and OSRAxisEnumToName().

12.99.3.76 OGRErr OGRSpatialReference::SetBonne (double *dfStdP1,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Bonne

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.77 OGRErr OGRSpatialReference::SetCEA (double *dfStdP1,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Cylindrical Equal Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), and importFromProj4().

12.99.3.78 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

pszName	the name of the compound coordinate system.
poHorizSRS	the horizontal SRS (PROJCS or GEOGCS).
poVertSRS	the vertical SRS (VERT_CS).

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), Clear(), OGR_SRSNode::Clone(), CPLError(), IsGeographic(), IsProjected(), and IsVertical().

12.99.3.79 OGRErr OGRSpatialReference::SetCS (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Cassini-Soldner

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), and importFromProj4().

12.99.3.80 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().

12.99.3.81 OGRErr OGRSpatialReference::SetEckert (int *nVariation,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Eckert I-VI

References CPLError(), SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.82 OGRErr OGRSpatialReference::SetEquirectangular (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equirectangular

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), and importFromWMSAUTO().

12.99.3.83 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().

12.99.3.84 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

pszTargetKey	the name or path to the parent node of the EXTENSION.
pszName	the name of the extension being fetched.
pszValue	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().

12.99.3.85 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 WG↩S72.
9. WKT (directly or in a file) in ESRI format should be prefixed with ESRI:: to trigger an automatic **morph↩FromESRI()** (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

pszDefinition	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().

12.99.3.86 OGRErr OGRSpatialReference::SetGaussSchreiberTMercator (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gauss Schreiber Transverse Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.87 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 tarnsferred over to the GEOGCS.

Parameters

pszName	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().

12.99.3.88 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

pszGeogName	user visible name for the geographic coordinate system (not to serve as a key).
pszDatumName	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.
pszSpheroidName	user visible spheroid name (not to serve as a key)
dfSemiMajor	the semi major axis of the spheroid.
dfInvFlattening	the inverse flattening for the spheroid. This can be computed from the semi minor axis as 1/f = 1.0 / (1.0 - semiminor/semimajor).
pszPMName	the name of the prime merdidian (not to serve as a key) If this is NULL a default value of "Greenwich" will be used.
dfPMOffset	the longitude of greenwich relative to this prime meridian.
pszAngularUnits	the angular units name (see **ogr_srs_api.h** (p. **??**) for some standard names). If NULL a value of "degrees" will be assumed.
dfConvertToRadians	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().

12.99.3.89 OGRErr OGRSpatialReference::SetGEOS (double *dfCentralMeridian,* double *dfSatelliteHeight,* double *dfFalseEasting,* double *dfFalseNorthing*)

Geostationary Satellite

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.90 OGRErr OGRSpatialReference::SetGH (double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Goode Homolosine

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.91 OGRErr OGRSpatialReference::SetGnomonic (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gnomonic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

12.99.3.92 OGRErr OGRSpatialReference::SetGS (double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gall Stereograpic

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.93 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

dfCenterLat	Latitude of the projection origin.
dfCenterLong	Longitude of the projection origin.
dfAzimuth	Azimuth, measured clockwise from North, of the projection centerline.
dfRectToSkew	?.
dfScale	Scale factor applies to the projection origin.
dfFalseEasting	False easting.
dfFalseNorthing	False northing.

Returns

OGRERR_NONE on success.

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

12.99.3.94 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

dfCenterLat	Latitude of the projection origin.
dfLat1	Latitude of the first point on center line.
dfLong1	Longitude of the first point on center line.
dfLat2	Latitude of the second point on center line.
dfLong2	Longitude of the second point on center line.
dfScale	Scale factor applies to the projection origin.
dfFalseEasting	False easting.
dfFalseNorthing	False northing.

Returns

OGRERR_NONE on success.

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), and importFromUSGS().

12.99.3.95 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

dfCenterLat	Latitude of the projection origin.
dfCenterLong	Longitude of the projection origin.
dfAzimuth	Azimuth, measured clockwise from North, of the projection centerline.
dfRectToSkew	?.
dfScale	Scale factor applies to the projection origin.
dfFalseEasting	False easting.
dfFalseNorthing	False northing.

Returns

OGRERR_NONE on success.

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.96 OGRErr OGRSpatialReference::SetIGH ( )

Interrupted Goode Homolosine

References SetProjection().

Referenced by importFromProj4().

12.99.3.97 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().

12.99.3.98 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().

12.99.3.99 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().

12.99.3.100 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().

12.99.3.101 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().

12.99.3.102 OGRErr OGRSpatialReference::SetLCCB (double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic (Belgium)

References SetNormProjParm(), and SetProjection().

12.99.3.103 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

pszUnitsName	the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. **??**) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
dfInMeters	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in **ogr_srs_api.h** (p. **??**).

Returns

OGRERR_NONE on success.

References SetTargetLinearUnits().

Referenced by Fixup(), importFromERM(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), importFromWMSAUTO(), SetLinearUnitsAndUpdateParameters(), SetStatePlane(), and SetUTM().

12.99.3.104 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

pszName	the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. **??**) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
dfInMeters	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in **ogr_srs_api.h** (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().

12.99.3.105 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

pszName	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().

12.99.3.106 OGRErr OGRSpatialReference::SetMC (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Miller Cylindrical

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

12.99.3.107 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().

12.99.3.108 OGRErr OGRSpatialReference::SetMollweide (double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Mollweide

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

12.99.3.109 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 existance. 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

pszNodePath	full path to attribute to be set. For instance "PROJCS|GEOGCS|UNIT".
pszNewNode↩ Value	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_SRS↩ Node::GetChildCount(), OGR_SRSNode::GetValue(), SetRoot(), and OGR_SRSNode::SetValue().

Referenced by importFromCRSURL(), importFromProj4(), importFromURN(), morphFromESRI(), morphToESRI(), SetFromUserInput(), SetGeocCS(), SetLocalCS(), SetProjCS(), SetProjection(), and SetUTM().

12.99.3.110 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

pszName	the parameter name, which should be selected from the macros in **ogr_srs_api.h** (p. **??**), such as SRS_PP_CENTRAL_MERIDIAN.
dfValue	value to assign.

Returns

OGRERR_NONE on success.

References SetProjParm().

Referenced by importFromProj4(), SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), Set↩ Eckert(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetHOMAC(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetQSC(), SetRobinson(), SetSinusoidal(), SetSOC(), SetStatePlane(), Set↩ Stereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetUTM(), SetVDG(), and Set↩ Wagner().

12.99.3.111 OGRErr OGRSpatialReference::SetNZMG (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

New Zealand Map Grid

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.112 OGRErr OGRSpatialReference::SetOrthographic (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Orthographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

12.99.3.113 OGRErr OGRSpatialReference::SetOS (double *dfOriginLat,* double *dfCMeridian,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Oblique Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), and importFromProj4().

12.99.3.114 OGRErr OGRSpatialReference::SetPolyconic (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Polyconic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFrom↩USGS().

12.99.3.115 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

pszName	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 Set↩Node().

12.99.3.116 OGRErr OGRSpatialReference::SetProjection (const char * *pszProjection*)

Set a projection name.

This method is the same as the C function **OSRSetProjection()** (p. **??**).

Parameters

pszProjection	the projection name, which should be selected from the macros in **ogr_srs_api.h** (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(), SetQSC(), SetRobinson(), SetSinusoidal(), SetSOC(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetUTM(), SetVDG(), and SetWagner().

12.99.3.117 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` pages for legal parameter names for specific projections.

Parameters

pszParmName	the parameter name, which should be selected from the macros in **ogr_srs_api.h** (p. **??**), such as SRS_PP_CENTRAL_MERIDIAN.
dfValue	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(), morphToESRI(), SetLinearUnitsAndUpdateParameters(), and SetNormProjParm().

12.99.3.118 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().

12.99.3.119 OGRErr OGRSpatialReference::SetQSC (double *dfCenterLat,* double *dfCenterLong*)

Quadrilateralized Spherical Cube

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.120 OGRErr OGRSpatialReference::SetRobinson (double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Robinson

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

12.99.3.121 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 is assumed by the **OGRSpatialReference** (p. **??**).

Parameters

poNewRoot	object to assign as root.

Referenced by CloneGeogCS(), CopyGeogCSFrom(), SetGeogCS(), SetNode(), SetVertCS(), and StripVertical().

12.99.3.122 OGRErr OGRSpatialReference::SetSinusoidal (double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Sinusoidal

References SetNormProjParm(), and SetProjection().

Referenced by importFromOzi(), importFromPCI(), importFromProj4(), and importFromUSGS().

12.99.3.123 OGRErr OGRSpatialReference::SetSOC (double *dfLatitudeOfOrigin,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Swiss Oblique Cylindrical

References SetNormProjParm(), and SetProjection().

12.99.3.124 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

nZone	State plane zone number, in the USGS numbering scheme (as dinstinct from the Arc/Info and Erdas numbering scheme.
bNAD83	TRUE if the NAD83 zone definition should be used or FALSE if the NAD27 zone definition should be used.

pszOverride↩ UnitName	Linear unit name to apply overriding the legal definition for this zone.
dfOverrideUnit	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 accessable.

References Clear(), CPLAtof(), CPLError(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), GetLinearUnits(), GetNormProjParm(), importFromEPSG(), SetLinearUnits(), SetLocalCS(), and SetNormProjParm().

Referenced by importFromESRI(), importFromPCI(), and importFromUSGS().

12.99.3.125 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().

12.99.3.126 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

pszTargetKey	the keyword to set the linear units for. ie. "PROJCS" or "VERT_CS"
pszUnitsName	the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. **??**) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
dfInMeters	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in **ogr_srs_api.h** (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().

12.99.3.127 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().

12.99.3.128 OGRErr OGRSpatialReference::SetTMG (double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Tunesia Mining Grid

References SetNormProjParm(), and SetProjection().

12.99.3.129 OGRErr OGRSpatialReference::SetTMSO (double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Transverse Mercator (South Oriented)

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.99.3.130 OGRErr OGRSpatialReference::SetTMVariant (const char * *pszVariantName,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Transverse Mercator variants.

References SetNormProjParm(), and SetProjection().

12.99.3.131 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

dfDX	X child in meters.
dfDY	Y child in meters.
dfDZ	Z child in meters.
dfEX	X rotation in arc seconds (optional, defaults to zero).
dfEY	Y rotation in arc seconds (optional, defaults to zero).
dfEZ	Z rotation in arc seconds (optional, defaults to zero).
dfPPM	scaling factor (parts per million).

Returns

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttr↩Node(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::InsertChild().

Referenced by importFromOzi(), importFromPCI(), and importFromProj4().

12.99.3.132 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().

12.99.3.133 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

nZone	UTM zone.
bNorth	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().

12.99.3.134 OGRErr OGRSpatialReference::SetVDG (double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

VanDerGrinten

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

12.99.3.135 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

pszVertCSName	the user visible name of the vertical coordinate system. Not used as a key.
pszVertDatum↩ Name	the user visible name of the vertical datum. It is helpful if this matches the EPSG name.

nVertDatumType	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().

12.99.3.136 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().

12.99.3.137 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

pszName	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(), importFrom↩WMSAUTO(), and SetFromUserInput().

12.99.3.138 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

poCurrent	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().

12.99.3.139 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().

12.99.3.140 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_↩ UNSUPPORTED_SRS if the SRS is well formed, but contains non-standard PROJECTION[] values.

References CPLDebug(), CPLGetConfigOption(), CSLTestBoolean(), and exportToWkt().

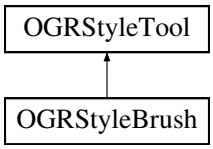
The documentation for this class was generated from the following files:

- **ogr_spatialref.h**
- ogr_fromepsg.cpp
- ogr_srs_dict.cpp
- ogr_srs_erm.cpp
- ogr_srs_esri.cpp
- ogr_srs_ozi.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

12.100 OGRStyleBrush Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStyleBrush:



12.100.1 Detailed Description

This class represents a style brush

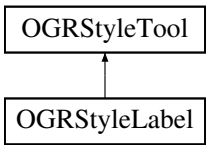
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

12.101 OGRStyleLabel Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStyleLabel:



12.101.1 Detailed Description

This class represents a style label

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

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

12.102 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** *=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.

12.102.1 Detailed Description

This class represents a style manager

12.102.2 Constructor & Destructor Documentation

12.102.2.1 OGRStyleMgr::OGRStyleMgr (**OGRStyleTable** * *poDataSetStyleTable* = NULL)

Constructor.

This method is the same as the C function **OGR_SM_Create()** (p. **??**)

Parameters

poDataSet↩ StyleTable	(currently unused, reserved for future use), pointer to **OGRStyleTable** (p. **??**). Pass NULL for now.

12.102.2.2 OGRStyleMgr::~OGRStyleMgr ( )

Destructor.

This method is the same as the C function **OGR_SM_Destroy()** (p. **??**)

12.102.3 Member Function Documentation

12.102.3.1 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

poStyleTool	the style tool defining the part to add.

Returns

TRUE on success, FALSE on errors.

References CPLStrdup().

12.102.3.2 GBool OGRStyleMgr::AddPart (const char * *pszPart*)

Add a part (style string) to the current style.

Parameters

pszPart	the style string defining the part to add.

Returns

TRUE on success, FALSE on errors.

References CPLStrdup().

12.102.3.3 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

pszStyleName	the name of the style to add.
pszStyleString	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().

12.102.3.4 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 instanciates a new object that should be freed with **OGR_ST_Destroy()** (p. **??**).

Parameters

nPartId	the part number (0-based index).
pszStyleString	(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().

12.102.3.5 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

pszStyleString	(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.

12.102.3.6 const char * OGRStyleMgr::GetStyleByName (const char * *pszStyleName*)

find a style in the current style table.

Parameters

pszStyleName	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().

12.102.3.7 const char * OGRStyleMgr::GetStyleName (const char * *pszStyleString* = NULL)

Get the name of a style from the style table.

Parameters

pszStyleString	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().

12.102.3.8 const char * OGRStyleMgr::GetStyleString (OGRFeature * *poFeature* = NULL)

Get the style string from the style manager.

Parameters

poFeature	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().

12.102.3.9 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

poFeature	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().

12.102.3.10 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

pszStyleString	the style string to use (can be NULL).

Returns

TRUE on success, FALSE on errors.

References CPLStrdup(), and GetStyleByName().

Referenced by InitFromFeature().

12.102.3.11 GBool OGRStyleMgr::SetFeatureStyleString (OGRFeature * *poFeature*, const char * *pszStyleString* = NULL, GBool *bNoMatching* = FALSE)

Set a style in a feature.

Parameters

poFeature	the feature object to store the style in
pszStyleString	the style to store
bNoMatching	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().

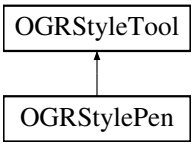
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

12.103 OGRStylePen Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStylePen:



12.103.1 Detailed Description

This class represents a style pen

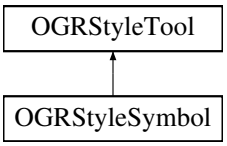
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

12.104 OGRStyleSymbol Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStyleSymbol:



12.104.1 Detailed Description

This class represents a style symbol

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

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

12.105 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.

12.105.1 Detailed Description

This class represents a style table

12.105.2 Member Function Documentation

12.105.2.1 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

pszName	the name the style to add.
pszStyleString	the style string to add.

Returns

TRUE on success, FALSE on error

References IsExist().

Referenced by OGRStyleMgr::AddStyle(), and ModifyStyle().

12.105.2.2 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().

12.105.2.3 const char * OGRStyleTable::Find (const char * *pszName*)

Get a style string by name.

Parameters

pszName	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().

12.105.2.4 const char * OGRStyleTable::GetStyleName (const char * *pszStyleString*)

Get style name by style string.

Parameters

pszStyleString	the style string to look up.

Returns

the Name of the matching style string or NULL on error.

References CSLCount().

Referenced by OGRStyleMgr::GetStyleName().

12.105.2.5 int OGRStyleTable::IsExist (const char * *pszName*)

Get the index of a style in the table by its name.

Parameters

pszName	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().

12.105.2.6 GBool OGRStyleTable::LoadStyleTable (const char * *pszFilename*)

Load a style table from a file.

Parameters

pszFilename	the name of the file to load from.

Returns

TRUE on success, FALSE on error

References CSLDestroy(), and CSLLoad().

12.105.2.7 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

pszName	the name of the style to modify.
pszStyleString	the style string.

Returns

TRUE on success, FALSE on error

References AddStyle(), and RemoveStyle().

12.105.2.8 void OGRStyleTable::Print (FILE * *fpOut*)

Print a style table to a FILE pointer.

Parameters

fpOut	the FILE pointer to print to.

12.105.2.9 GBool OGRStyleTable::RemoveStyle (const char * *pszName*)

Remove a style in the table by its name.

Parameters

pszName	the name of the style to remove.

Returns

TRUE on success, FALSE on error

References IsExist().

Referenced by ModifyStyle().

12.105.2.10 GBool OGRStyleTable::SaveStyleTable (const char * *pszFilename*)

Save a style table to a file.

Parameters

pszFilename	the name of the file to save to.

Returns

TRUE on success, FALSE on error

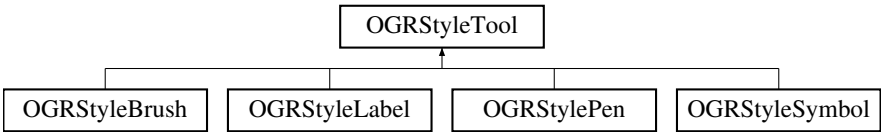
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

12.106 OGRStyleTool Class Reference

`#include <ogr_featurestyle.h>`

Inheritance diagram for OGRStyleTool:



12.106.1 Detailed Description

This class represents a style tool

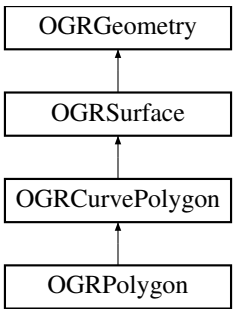
The documentation for this class was generated from the following files:

- **ogr_featurestyle.h**
- ogrfeaturestyle.cpp

12.107 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.

Static Public Member Functions

- static **OGRPolygon** * **CastToPolygon** (**OGRSurface** *poSurface)

  Cast to polygon.

- static **OGRCurvePolygon** * **CastToCurvePolygon** (**OGRSurface** *poSurface)

  Cast to curve polygon.

12.107.1 Detailed Description

Abstract base class for 2 dimensional objects like polygons or curve polygons.

12.107.2 Member Function Documentation

12.107.2.1 OGRCurvePolygon * OGRSurface::CastToCurvePolygon (OGRSurface * *poSurface*) [static]

Cast to curve polygon.

The passed in geometry is consumed and a new one returned (or NULL in case of failure)

Parameters

poSurface	the input geometry - ownership is passed to the method.

Returns

new geometry.

Referenced by OGRGeometryFactory::forceTo().

12.107.2.2 OGRPolygon * OGRSurface::CastToPolygon (OGRSurface * *poSurface*) [static]

Cast to polygon.

The passed in geometry is consumed and a new one returned (or NULL in case of failure)

Parameters

poSurface	the input geometry - ownership is passed to the method.

Returns

new geometry.

Referenced by OGRMultiSurface::CastToMultiPolygon(), and OGRGeometryFactory::forceToPolygon().

12.107.2.3 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 **OGRCurvePolygon** (p. **??**).

12.107.2.4 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

poPoint	point to be set with an internal point.

Returns

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

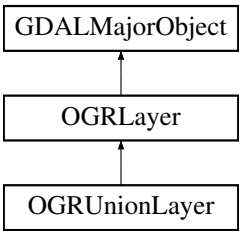
Implemented in **OGRPolygon** (p. **??**), and **OGRCurvePolygon** (p. **??**).

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

- **ogr_geometry.h**
- ogrsurface.cpp

12.108 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** (GIntBig nFeatureId)

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

  Create and write a new feature within a layer.
- virtual OGRErr **ISetFeature** (**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 GIntBig **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** (int iGeomField, **OGREnvelope** *psExtent, int bForce=TRUE)

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

- virtual OGRErr **GetExtent** (**OGREnvelope** *psExtent, int bForce)

  Fetch the extent of this layer.

- virtual void **SetSpatialFilter** (**OGRGeometry** *poGeomIn)

  Set a new spatial filter.

- virtual void **SetSpatialFilter** (int iGeomField, **OGRGeometry** *)

  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.

Additional Inherited Members

12.108.1 Member Function Documentation

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

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

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

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (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.

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

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

Parameters

iGeomField	the index of the geometry field on which to compute the extent.

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. **??**).

References CPLError(), OGRFeatureDefn::GetGeomFieldIndex(), OGRLayer::GetLayerDefn(), and GetLayer↩Defn().

Referenced by GetExtent().

12.108.1.2 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

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayer** (p. **??**).

References GetExtent().

12.108.1.3 OGRFeature * OGRUnionLayer::GetFeature (GIntBig *nFID*) [virtual]

Fetch a feature by its identifier.

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

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (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 (with **GetNextFeature()** (p. **??**)) 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

nFID	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(), ResetReading(), and SetSpatialFilter().

12.108.1.4 GIntBig 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. **??**).

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

Parameters

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

Returns

feature count, -1 if count not known.

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetFeatureCount(), and ResetReading().

12.108.1.5 OGRwkbGeometryType OGRUnionLayer::GetGeomType ( ) [virtual]

Return the layer geometry type.

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

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

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

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

Returns

the geometry type

Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. **??**).

References OGRLayer::GetGeomType(), OGRGeomFieldDefn::GetType(), and wkbNone.

12.108.1.6 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::AddGeomFieldDefn(), OGRFeatureDefn::DeleteFieldDefn(), OGRFeatureDefn::DeleteGeomFieldDefn(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetFieldIndex(), OGRFeatureDefn::GetGeomFieldCount(), OGRFeatureDefn::GetGeomFieldDefn(), OGRFeatureDefn::GetGeomFieldIndex(), OGRLayer::GetLayerDefn(), OGRFieldDefn::GetNameRef(), OGRGeomFieldDefn::GetNameRef(), GetSpatialRef(), OGRGeomFieldDefn::GetSpatialRef(), OGRGeomFieldDefn::GetType(), OFTString, OGRSpatialReference::Reference(), OGRFeatureDefn::Reference(), OGRFeatureDefn::SetGeomType(), OGRGeomFieldDefn::SetSpatialRef(), OGRGeomFieldDefn::SetType(), and wkbNone.

Referenced by GetExtent(), GetNextFeature(), ICreateFeature(), ISetFeature(), SetAttributeFilter(), and SetSpatialFilter().

12.108.1.7 virtual const char* OGRUnionLayer::GetName ( ) [inline], [virtual]

Return the layer name.

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

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. **??**).

Referenced by ICreateFeature(), and ISetFeature().

12.108.1.8 OGRFeature * OGRUnionLayer::GetNextFeature ( ) [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity 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.

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

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::GetGeomFieldRef(), GetLayerDefn(), OGRLayer::GetNextFeature(), and ResetReading().

12.108.1.9 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.

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

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(), OGRGeomFieldDefn::GetSpatialRef(), and OGRSpatialReference::Reference().

Referenced by GetLayerDefn().

12.108.1.10 OGRErr OGRUnionLayer::ICreateFeature (OGRFeature * *poFeature*) [virtual]

Create and write a new feature within a layer.

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

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

Reimplemented from **OGRLayer** (p. **??**).

References CPLError(), OGRLayer::CreateFeature(), OGRFeature::GetFID(), OGRFeature::GetFieldAsString(), GetLayerDefn(), GetName(), OGRFeature::IsFieldSet(), OGRFeature::SetFID(), and OGRFeature::SetFrom().

12.108.1.11 OGRErr OGRUnionLayer::ISetFeature (OGRFeature * *poFeature*) [virtual]

Rewrite an existing feature.

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

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

Parameters

poFeature	the feature to write.

Returns

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

Since

GDAL 2.0

Reimplemented from **OGRLayer** (p. **??**).

References CPLError(), OGRFeature::GetFID(), OGRFeature::GetFieldAsString(), GetLayerDefn(), GetName(), OGRFeature::IsFieldSet(), OGRLayer::SetFeature(), OGRFeature::SetFID(), and OGRFeature::SetFrom().

12.108.1.12 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(), GetNextFeature(), and SetSpatialFilter().

12.108.1.13 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 interprete the WHERE clause in which case the capabilities will be broader than those of OGR SQL.

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

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

Parameters

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

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

References CPLStrdup(), GetLayerDefn(), and OGRLayer::SetAttributeFilter().

12.108.1.14 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

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

Returns

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

Reimplemented from **OGRLayer** (p. **??**).

References CSLDestroy(), CSLDuplicate(), and OGRLayer::SetIgnoredFields().

12.108.1.15 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

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

Reimplemented from **OGRLayer** (p. **??**).

Referenced by GetFeature().

12.108.1.16 void OGRUnionLayer::SetSpatialFilter (int *iGeomField,* OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**Get↩SpatialRef()** (p. **??**)). In the future this may be generalized.

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

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

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

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayer** (p. **??**).

References CPLError(), GetLayerDefn(), and ResetReading().

12.108.1.17 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() 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().

12.108.1.18 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.
- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using **CreateGeomField()** (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 Rollback↩ Transaction() 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.

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

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

Parameters

pszCap	the name of the capability to test.

Returns

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

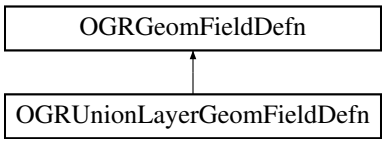
Implements **OGRLayer** (p. **??**).

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

- ogrunionlayer.h
- ogrunionlayer.cpp

12.109 OGRUnionLayerGeomFieldDefn Class Reference

Inheritance diagram for OGRUnionLayerGeomFieldDefn:



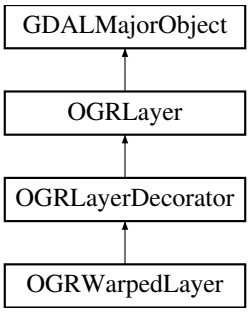
Additional Inherited Members

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

- ogrunionlayer.h
- ogrunionlayer.cpp

12.110 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 void **SetSpatialFilter** (int iGeomField, **OGRGeometry** ∗)

  Set a new spatial filter.
- virtual void **SetSpatialFilterRect** (int iGeomField, double dfMinX, double dfMinY, double dfMaxX, double df↩ MaxY)

  Set a new rectangular spatial filter.
- virtual **OGRFeature** ∗ **GetNextFeature** ()

  Fetch the next available feature from this layer.
- virtual **OGRFeature** ∗ **GetFeature** (GIntBig nFID)

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

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

  Create and write a new feature within a layer.
- virtual **OGRFeatureDefn** ∗ **GetLayerDefn** ()

  Fetch the schema information for this layer.
- virtual **OGRSpatialReference** ∗ **GetSpatialRef** ()

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

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

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

Additional Inherited Members

12.110.1 Member Function Documentation

12.110.1.1 OGRErr OGRWarpedLayer::GetExtent (int *iGeomField,* OGREnvelope ∗ *psExtent,* int *bForce* = TRUE) [virtual]

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

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

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **GetExtent()** (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.

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

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

Parameters

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::GetExtent().

Referenced by GetExtent().

12.110.1.2 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

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References GetExtent().

12.110.1.3 OGRFeature * OGRWarpedLayer::GetFeature (GIntBig *nFID*) [virtual]

Fetch a feature by its identifier.

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

If this method returns a non-NULL feature, it is guaranteed that its feature id (**OGRFeature::GetFID()** (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 (with **GetNextFeature()** (p. **??**)) 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

nFID	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().

12.110.1.4 GIntBig 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. **??**).

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

Parameters

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

Returns

feature count, -1 if count not known.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::GetFeatureCount().

12.110.1.5 OGRFeatureDefn * OGRWarpedLayer::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.

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRFeatureDefn::Clone(), OGRLayer::GetLayerDefn(), and OGRFeatureDefn::Reference().

Referenced by SetSpatialFilter().

12.110.1.6 OGRFeature * OGRWarpedLayer::GetNextFeature ( ) [virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity 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.

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

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::GetGeomFieldRef(), and OGRLayer::GetNextFeature().

12.110.1.7 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.

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

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. **??**).

References OGRLayer::GetSpatialRef().

12.110.1.8 OGRErr OGRWarpedLayer::ICreateFeature (OGRFeature * *poFeature*) [virtual]

Create and write a new feature within a layer.

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

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::CreateFeature().

12.110.1.9 OGRErr OGRWarpedLayer::ISetFeature (OGRFeature * *poFeature*) [virtual]

Rewrite an existing feature.

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

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

Parameters

poFeature	the feature to write.

Returns

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

Since

GDAL 2.0

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::SetFeature().

12.110.1.10 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

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

12.110.1.11 void OGRWarpedLayer::SetSpatialFilter (int *iGeomField,* OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**GetSpatialRef()** (p. **??**)). In the future this may be generalized.

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

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

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

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayerDecorator** (p. **??**).

References CPLError(), OGRGeometry::getEnvelope(), GetLayerDefn(), OGRLayerDecorator::ResetReading(), OGRLayer::SetSpatialFilter(), and OGRLayer::SetSpatialFilterRect().

12.110.1.12 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

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::SetSpatialFilterRect().

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

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the **GetNextFeature()** (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 as the geometry field definition it corresponds to (as returned by **GetLayerDefn()** (p. **??**)->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->**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_SetSpatialFilterRectEx()** (p. **??**).

Parameters

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

Since

GDAL 1.11

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::SetSpatialFilterRect().

12.110.1.14 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.

- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using **CreateGeomField()** (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.

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

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

Parameters

pszCap	the name of the capability to test.

Returns

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

Reimplemented from **OGRLayerDecorator** (p. **??**).

References OGRLayer::TestCapability().

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

- ogrwarpedlayer.h
- ogrwarpedlayer.cpp

12.111 osr_cs_wkt_parse_context Struct Reference

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

- osr_cs_wkt.h

12.112 osr_cs_wkt_tokens Struct Reference

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

- osr_cs_wkt.c

12.113 ParseContext Struct Reference

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

- cpl_minixml.cpp

12.114 PCIDatums Struct Reference

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

- ogr_srs_pci.cpp

12.115 projUV Struct Reference

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

- ogrct.cpp

12.116 RingBuffer Class Reference

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

- cpl_vsil_curl_streaming.cpp

12.117 SFRegion Class Reference

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

- cpl_vsil_sparsefile.cpp

12.118 StackContext Struct Reference

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

- cpl_minixml.cpp

12.119 swq_col_def Struct Reference

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

- swq.h

12.120 swq_custom_func_registrar Class Reference

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

- swq.h

12.121 swq_expr_node Class Reference

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

- swq.h
- swq_expr_node.cpp

12.122 swq_field_list Class Reference

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

- swq.h

12.123 swq_join_def Struct Reference

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

- swq.h

12.124 swq_op_registrar Class Reference

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

- swq.h
- swq_op_registrar.cpp

12.125 swq_operation Struct Reference

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

- swq.h

12.126 swq_order_def Struct Reference

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

- swq.h

12.127 swq_parse_context Class Reference

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

- swq.h

12.128 swq_select Class Reference

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

- swq.h
- swq_select.cpp

12.129 swq_select_parse_options Class Reference

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

- swq.h

12.130 swq_summary Struct Reference

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

- swq.h

12.131 swq_table_def Struct Reference

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

- swq.h

12.132 tm_unz_s Struct Reference

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

- cpl_minizip_unzip.h

12.133 tm_zip_s Struct Reference

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

- cpl_minizip_zip.h

12.134 unz_file_info_internal_s Struct Reference

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

- cpl_minizip_unzip.cpp

12.135 unz_file_info_s Struct Reference

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

- cpl_minizip_unzip.h

12.136 unz_file_pos_s Struct Reference

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

- cpl_minizip_unzip.h

12.137 unz_global_info_s Struct Reference

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

- cpl_minizip_unzip.h

12.138 unz_s Struct Reference

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

- cpl_minizip_unzip.cpp

12.139 VSIArchiveContent Struct Reference

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

- cpl_vsi_virtual.h

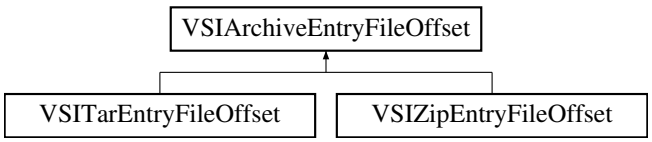
12.140 VSIArchiveEntry Struct Reference

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

- cpl_vsi_virtual.h

12.141 VSIArchiveEntryFileOffset Class Reference

Inheritance diagram for VSIArchiveEntryFileOffset:

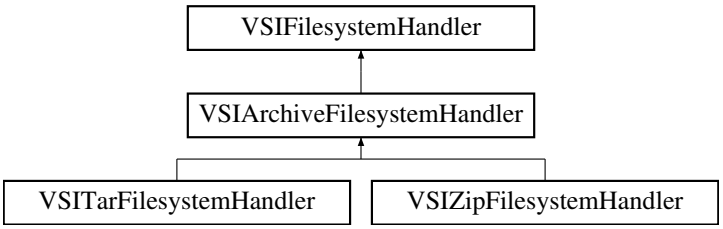


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil_abstract_archive.cpp

12.142 VSIArchiveFilesystemHandler Class Reference

Inheritance diagram for VSIArchiveFilesystemHandler:

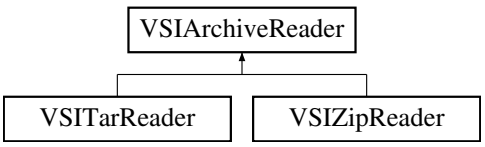


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil_abstract_archive.cpp

12.143 VSIArchiveReader Class Reference

Inheritance diagram for VSIArchiveReader:

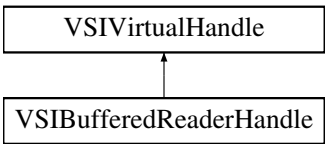


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil_abstract_archive.cpp

12.144 VSIBufferedReaderHandle Class Reference

Inheritance diagram for VSIBufferedReaderHandle:



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

- cpl_vsil_buffered_reader.cpp

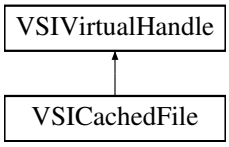
12.145 VSICacheChunk Class Reference

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

- cpl_vsil_cache.cpp

12.146 VSICachedFile Class Reference

Inheritance diagram for VSICachedFile:

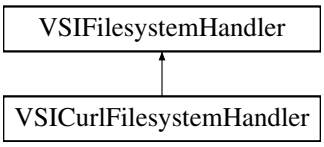


The documentation for this class was generated from the following file:

- cpl_vsil_cache.cpp

12.147 VSICurlFilesystemHandler Class Reference

Inheritance diagram for VSICurlFilesystemHandler:

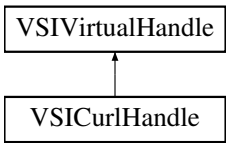


The documentation for this class was generated from the following file:

- cpl_vsil_curl.cpp

12.148 VSICurlHandle Class Reference

Inheritance diagram for VSICurlHandle:

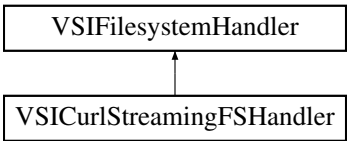


The documentation for this class was generated from the following file:

- cpl_vsil_curl.cpp

12.149 VSICurlStreamingFSHandler Class Reference

Inheritance diagram for VSICurlStreamingFSHandler:

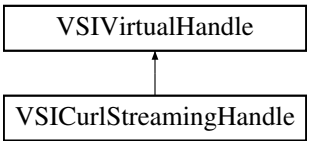


The documentation for this class was generated from the following file:

- cpl_vsil_curl_streaming.cpp

12.150 VSICurlStreamingHandle Class Reference

Inheritance diagram for VSICurlStreamingHandle:



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

- cpl_vsil_curl_streaming.cpp

12.151 VSIDIR Struct Reference

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

- vsipreload.cpp

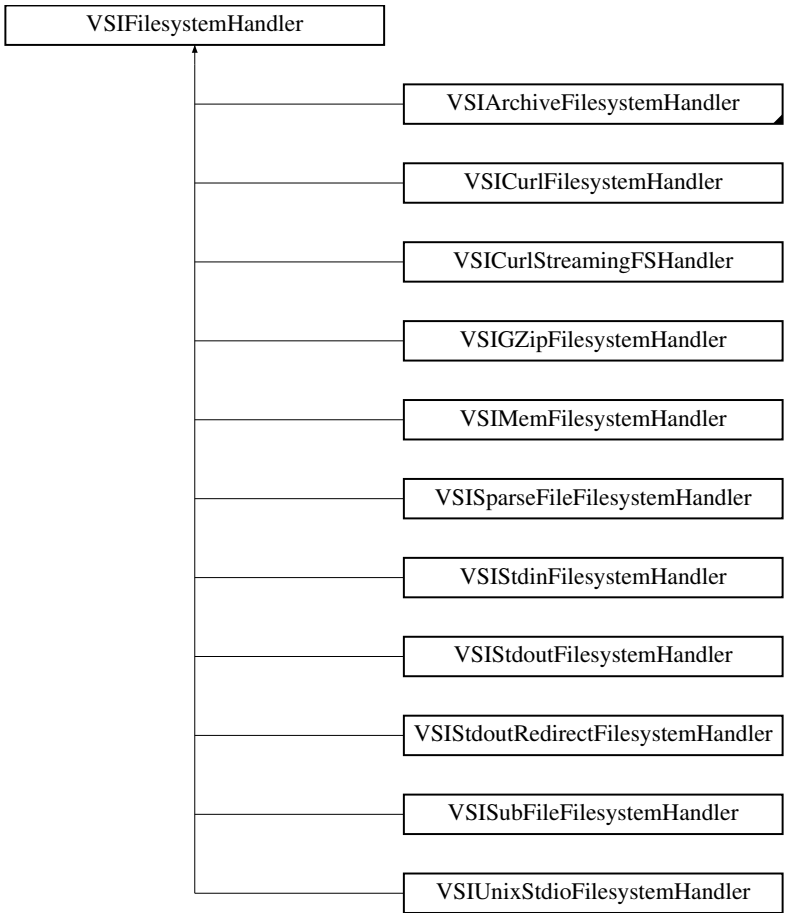
12.152 VSIFileManager Class Reference

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

- cpl_vsi_virtual.h
- cpl_vsil.cpp

12.153 VSIFilesystemHandler Class Reference

Inheritance diagram for VSIFilesystemHandler:

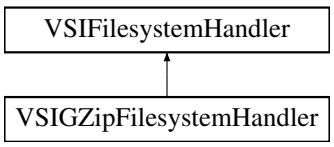


The documentation for this class was generated from the following file:

- cpl_vsi_virtual.h

12.154 VSIGZipFilesystemHandler Class Reference

Inheritance diagram for VSIGZipFilesystemHandler:

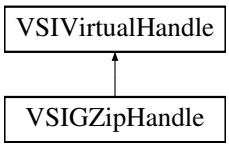


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

12.155 VSIGZipHandle Class Reference

Inheritance diagram for VSIGZipHandle:

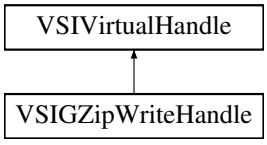


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

12.156 VSIGZipWriteHandle Class Reference

Inheritance diagram for VSIGZipWriteHandle:



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

- cpl_vsil_gzip.cpp

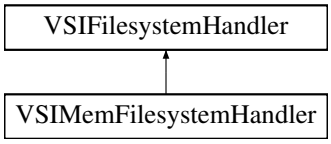
12.157 VSIMemFile Class Reference

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

- cpl_vsi_mem.cpp

12.158 VSIMemFilesystemHandler Class Reference

Inheritance diagram for VSIMemFilesystemHandler:

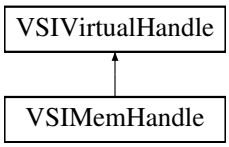


The documentation for this class was generated from the following file:

- cpl_vsi_mem.cpp

12.159 VSIMemHandle Class Reference

Inheritance diagram for VSIMemHandle:



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

- cpl_vsi_mem.cpp

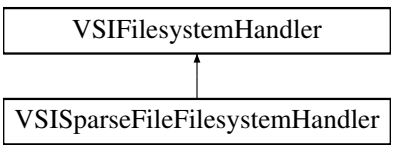
12.160 VSIReadDirRecursiveTask Struct Reference

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

- cpl_vsil.cpp

12.161 VSISparseFileFilesystemHandler Class Reference

Inheritance diagram for VSISparseFileFilesystemHandler:

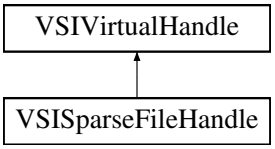


The documentation for this class was generated from the following file:

- cpl_vsil_sparsefile.cpp

12.162 VSISparseFileHandle Class Reference

Inheritance diagram for VSISparseFileHandle:

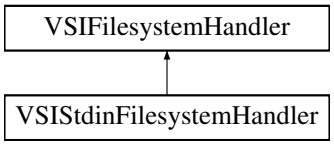


The documentation for this class was generated from the following file:

- cpl_vsil_sparsefile.cpp

12.163 VSIStdinFilesystemHandler Class Reference

Inheritance diagram for VSIStdinFilesystemHandler:

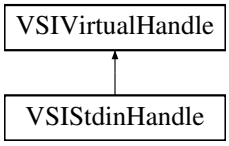


The documentation for this class was generated from the following file:

- cpl_vsil_stdin.cpp

12.164 VSIStdinHandle Class Reference

Inheritance diagram for VSIStdinHandle:

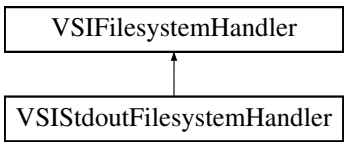


The documentation for this class was generated from the following file:

- cpl_vsil_stdin.cpp

12.165 VSIStdoutFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutFilesystemHandler:

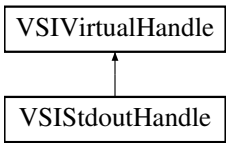


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

12.166 VSIStdoutHandle Class Reference

Inheritance diagram for VSIStdoutHandle:

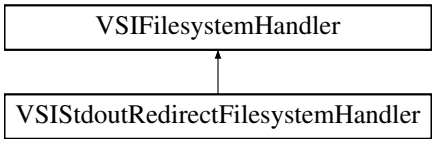


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

12.167 VSIStdoutRedirectFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutRedirectFilesystemHandler:

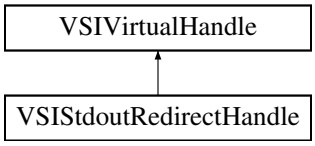


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

12.168 VSIStdoutRedirectHandle Class Reference

Inheritance diagram for VSIStdoutRedirectHandle:

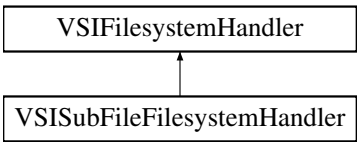


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

12.169 VSISubFileFilesystemHandler Class Reference

Inheritance diagram for VSISubFileFilesystemHandler:

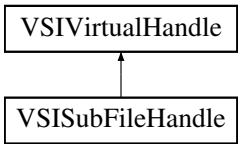


The documentation for this class was generated from the following file:

- cpl_vsil_subfile.cpp

12.170 VSISubFileHandle Class Reference

Inheritance diagram for VSISubFileHandle:

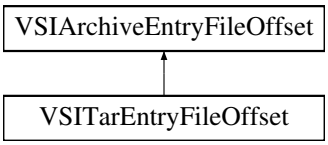


The documentation for this class was generated from the following file:

- cpl_vsil_subfile.cpp

12.171 VSITarEntryFileOffset Class Reference

Inheritance diagram for VSITarEntryFileOffset:

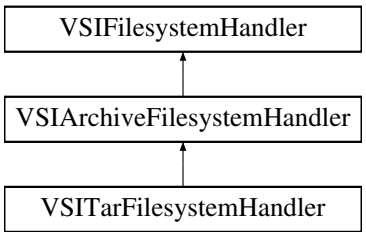


The documentation for this class was generated from the following file:

- cpl_vsil_tar.cpp

12.172 VSITarFilesystemHandler Class Reference

Inheritance diagram for VSITarFilesystemHandler:

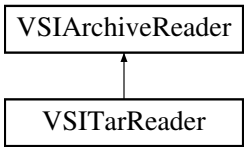


The documentation for this class was generated from the following file:

- cpl_vsil_tar.cpp

12.173 VSITarReader Class Reference

Inheritance diagram for VSITarReader:

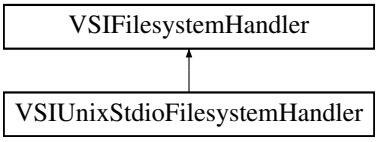


The documentation for this class was generated from the following file:

- cpl_vsil_tar.cpp

12.174 VSIUnixStdioFilesystemHandler Class Reference

Inheritance diagram for VSIUnixStdioFilesystemHandler:

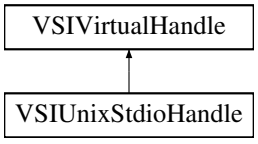


The documentation for this class was generated from the following file:

- cpl_vsil_unix_stdio_64.cpp

12.175 VSIUnixStdioHandle Class Reference

Inheritance diagram for VSIUnixStdioHandle:

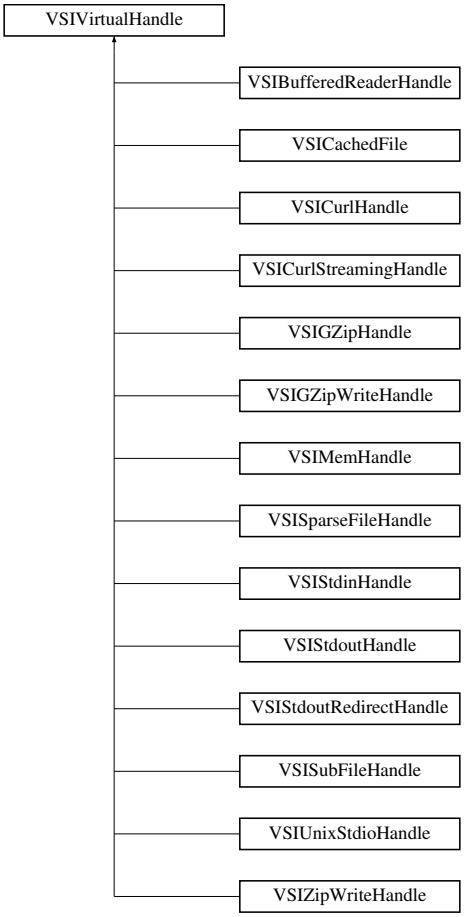


The documentation for this class was generated from the following file:

- cpl_vsil_unix_stdio_64.cpp

12.176 VSIVirtualHandle Class Reference

Inheritance diagram for VSIVirtualHandle:

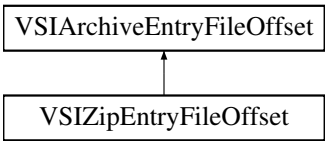


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil.cpp

12.177 VSIZipEntryFileOffset Class Reference

Inheritance diagram for VSIZipEntryFileOffset:

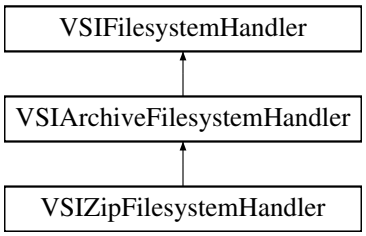


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

12.178 VSIZipFilesystemHandler Class Reference

Inheritance diagram for VSIZipFilesystemHandler:

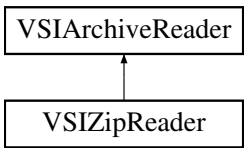


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

12.179 VSIZipReader Class Reference

Inheritance diagram for VSIZipReader:

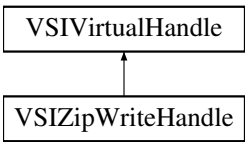


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

12.180 VSIZipWriteHandle Class Reference

Inheritance diagram for VSIZipWriteHandle:



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

- cpl_vsil_gzip.cpp

12.181 WriteFuncStruct Struct Reference

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

- cpl_vsil_curl.cpp
- cpl_vsil_curl_streaming.cpp

12.182 yyalloc Union Reference

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

- osr_cs_wkt_parser.c
- swq_parser.cpp

12.183 zip_fileinfo Struct Reference

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

- cpl_minizip_zip.h

12.184 zip_internal Struct Reference

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

- cpl_minizip_zip.cpp

12.185 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)

- GIntBig **CPLAtoGIntBig** (const char *pszString)
- GIntBig **CPLAtoGIntBigEx** (const char *pszString, int bWarn, int *pbOverflow)
- void * **CPLScanPointer** (const char *, int)
- int **CPLPrintString** (char *, const char *, int)
- int **CPLPrintStringFill** (char *, const char *, int)
- int **CPLPrintInt32** (char *, GInt32, 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 *pn↩ OutBytes)

  Uncompress a buffer compressed with ZLib DEFLATE compression.
- char * **CPLsetlocale** (int category, const char *locale)

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 the user works in a localized environment and the user's input will be done according to 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

nptr	Pointer to string to convert.

Returns

Converted value, if any.

References CPLStrtod().

Referenced by OGRSpatialReference::CloneGeogCS(), CPLScanDouble(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::Fixup(), OGRSpatialReference::GetAngular↩Units(), OGRFeature::GetFieldAsDouble(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference↩::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OG↩RSpatialReference::GetTargetLinearUnits(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference↩::importFromERM(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::importFromOzi(), OGR↩SpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::import↩FromWMSAUTO(), OGRSpatialReference::IsSameGeogCS(), OGRSpatialReference::IsSameVertCS(), OPTGet↩ParameterInfo(), OGRFeature::SetField(), OGRSpatialReference::SetGeogCS(), and OGRSpatialReference::Set↩StatePlane().

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

nptr	Pointer to string to convert.

point	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

nptr	The string to convert.

Returns

Converted value, if any. Zero on failure.

References CPLStrtodDelim().

Referenced by OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromProj4(), and OGR↩ GeocodeCreateSession().

13.1.2.4 GIntBig CPLAtoGIntBig (const char * *pszString*)

Convert a string to a 64 bit signed integer.

Parameters

pszString	String containing 64 bit signed integer.

Returns

64 bit signed integer.

Since

GDAL 2.0

13.1.2.5 GIntBig CPLAtoGIntBigEx (const char * *pszString,* int *bWarn,* int * *pbOverflow*)

Convert a string to a 64 bit signed integer.

Parameters

pszString	String containing 64 bit signed integer.

bWarn	Issue a warning if an overflow occurs during conversion
pbOverflow	Pointer to an integer to store if an overflow occured, or NULL

Returns

64 bit signed integer.

Since

GDAL 2.0

References CPLError().

Referenced by OGRFeature::GetFieldAsInteger64(), and OGRFeature::SetField().

13.1.2.6 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 VSICalloc() 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

nCount	number of objects to allocate.
nSize	size (in bytes) of object to allocate.

Returns

pointer to newly allocated memory, only NULL if nSize * nCount is NULL.

References CPLMalloc().

Referenced by CPLCreateXMLNode(), CPLHashSetNew(), CPLHTTPFetch(), CSLTokenizeString2(), OGR_SR↩ SNode::exportToWkt(), OGRPolygon::exportToWkt(), OGR_SRSNode::FixupOrdering(), CPLODBCStatement::↩ GetColumns(), OGRBuildPolygonFromEdges(), OGRFeature::OGRFeature(), OGRGeocodeCreateSession(), and OPTGetParameterList().

13.1.2.7 int CPLCheckForFile (char * *pszFilename*, char ** *papszSiblingFiles*)

Check for file existance.

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 **VSIStatL()** (p. **??**) is used to test for the files existance, and no case insensitive testing is done.

Parameters

pszFilename	name of file to check for - filename case updated in some cases.
papszSibling↩ Files	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 VSIStatL().

13.1.2.8 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

pszPath	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().

13.1.2.9 void CPLCloseShared (FILE * *fp*)

Close shared file.

Dereferences the indicated file handle, and closes it if the reference count has dropped to zero. A **CPLError()** (p. **??**) is issued if the file is not in the shared file list.

Parameters

fp	file handle from **CPLOpenShared()** (p. **??**) to deaccess.

References CPLError(), and VSIFCloseL().

13.1.2.10 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 **CSLDestroy()** (p. **??**).

Parameters

pszOldFilename	path to old prototype file.
pszNew↩ Filename	path to new prototype file.
papszFileList	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().

13.1.2.11 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

dfDec	Angle in decimal degrees.

Returns

Angle in packed DMS format.

Referenced by OGRSpatialReference::exportToUSGS().

13.1.2.12 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

fp	File handle to write to.

References CPLDebug().

13.1.2.13 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

pszBaseDir	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 relitivizing.
pszTarget	the filename to be changed to be relative to pszBaseDir.
pbGotRelative	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().

13.1.2.14 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

pszBuffer	pointer to the targeting character buffer.
nBufferSize	maximum size of the string to read (not including termonating '\0').
fp	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().

13.1.2.15 const char* CPLFormCIFilename (const char * *pszPath,* const char * *pszBasename,* const char * *pszExtension*)

Case insensitive file searching, returing 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 equivelent to **CPLFormFilename()** (p. **??**) on case insensitive file systems (like Windows).

Parameters

pszPath	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.
pszBasename	file basename. May optionally have path and/or extension. May not be NULL.
pszExtension	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(), VSIIsCaseSensitiveFS(), and VSIStatExL().

13.1.2.16 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

pszPath	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.
pszBasename	file basename. May optionally have path and/or extension. Must *NOT* be NULL.
pszExtension	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 CPLCorrespondingPaths(), CPLFormCIFilename(), CPLGenerateTempFilename(), and CPL↩ UnlinkTree().

13.1.2.17 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

pszStem	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().

13.1.2.18 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

pszFullFilename	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 CPLCorrespondingPaths().

13.1.2.19 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 no 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 temporary 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 useful
// restore old value
CPLSetConfigOption(pszKey, pszOldVal);
CPLFree(pszOldVal);
```

Parameters

pszKey	the key of the option to retrieve
pszDefault	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 CPLDebug(), CPLGenerateTempFilename(), CPLHTTPFetch(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRSpatialReference::exportToProj4(), OGRFeatureDefn::GetGeomType(), GOA2GetAccessToken(), GOA2GetAuthorizationURL(), GOA2GetRefreshToken(), OGRSpatialReference::morphFromESRI(), OGR_G_CreateFromGML(), OGRGeometryFactory::organizePolygons(), OGRFeature::SetField(), OGRSimpleCurve::transform(), and OGRSpatialReference::Validate().

13.1.2.20 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().

13.1.2.21 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

pszFilename	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().

13.1.2.22 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

pszPathBuf	the buffer into which the path is placed.
nMaxLength	the buffer size, MAX_PATH+1 is suggested.

Returns

FALSE on failure or TRUE on success.

13.1.2.23 const char* CPLGetExtension (const char * *pszFullFilename*)

Extract filename extension from full filename.

Returns a string containing the extention 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

pszFullFilename	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 OGRGeocodeCreateSession().

13.1.2.24 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

pszFullFilename	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().

13.1.2.25 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

pszFilename	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 CPLCorrespondingPaths().

13.1.2.26 CPLSharedFileInfo* CPLGetSharedList (int * *pnCount*)

Fetch list of open shared files.

Parameters

pnCount	place to put the count of entries.

Returns

the pointer to the first in the array of shared file info structures.

13.1.2.27 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-prending or post-pending an underscore.

Some of these issues may be worked on in the future.

Parameters

pszLibrary	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.
pszSymbolName	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().

13.1.2.28 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

pszFilename	the filename with path to test.

Returns

TRUE if the filename is relative or FALSE if it is absolute.

Referenced by CPLExtractRelativePath(), and CPLProjectRelativeFilename().

13.1.2.29 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

nSize	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(), CPLBinaryToHex(), CPLCalloc(), CPLEscapeString(), CPL↩ ForceToASCII(), CPLFormCIFilename(), CPLGetCurrentDir(), CPLHashSetNew(), CPLHTTPFetch(), CPLList↩ Append(), CPLListInsert(), CPLParseNameValue(), CPLPrintTime(), CPLPushErrorHandlerEx(), CPLQuad↩ TreeCreate(), CPLScanDouble(), CPLScanLong(), CPLScanString(), CPLScanUIntBig(), CPLScanULong(), C↩ PLSerializeXMLTree(), CPLStrdup(), CPLUnescapeString(), CSLDuplicate(), CSLSetNameValue(), CSLSet↩ NameValueSeparator(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGR_↩ SRSNode::exportToWkt(), CPLODBCStatement::Fetch(), OGRSpatialReference::importFromCRSURL(), OGR↩ SpatialReference::importFromPanorama(), OGRSpatialReference::importFromPCI(), CPLODBCDriverInstaller::↩ InstallDriver(), OGRSpatialReference::morphToESRI(), OGR_G_ExportToGMLEx(), OGRFeature::OGRFeature(), OGRFeatureDefn::OGRFeatureDefn(), OGRLayer::ReorderField(), OGRFeatureDefn::ReorderFieldDefns(), O↩ GRFeature::SetField(), OGRSpatialReference::SetFromUserInput(), CPLStringList::SetNameValue(), and OGR↩ Proj4CT::Transform().

13.1.2.30 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 VSIFOpen() or **VSIFOpenL()** (p. **??**) function is used to actually open the file, when an existing file handle can't be shared.

Parameters

pszFilename	the name of the file to open.
pszAccess	the normal fopen()/VSIFOpen() style access string.
bLarge	If TRUE **VSIFOpenL()** (p. **??**) (for large files) will be used instead of VSIFOpen().

Returns

a file handle or NULL if opening fails.

References CPLRealloc(), CPLStrdup(), and VSIFOpenL().

13.1.2.31 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: deg = ang/1000000 (fractional portion truncated)
3. The minutes are separated out: min = (ang - deg ∗ 1000000) / 1000 (fractional portion truncated)
4. The seconds are then computed: sec = ang - deg ∗ 1000000 - min ∗ 1000
5. The total angle in seconds is computed: sec = deg ∗ 3600.0 + min ∗ 60.0 + 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

dfPacked	Angle in packed DMS format.

Returns

Angle in decimal degrees.

Referenced by OGRSpatialReference::importFromUSGS().

13.1.2.32 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

pszBuffer	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.
pszFormat	Format specifier (for example, "%16.9E").
dfValue	Numerical value to print.
pszLocale	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(), CPLsnprintf(), and CPLsprintf().

13.1.2.33 int CPLPrintInt32 (char ∗ *pszBuffer,* GInt32 *iValue,* int *nMaxLen*)

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	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.

iValue	Numerical value to print.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

References CPLPrintString().

Referenced by OGRSpatialReference::exportToPCI().

13.1.2.34 int CPLPrintPointer (char * *pszBuffer,* void * *pValue,* int *nMaxLen*)

Print pointer value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	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.
pValue	Pointer to ASCII encode.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

References CPLPrintString().

13.1.2.35 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

pszDest	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
pszSrc	Pointer to the source buffer.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

Referenced by CPLPrintDouble(), CPLPrintInt32(), CPLPrintPointer(), CPLPrintTime(), and CPLPrintUIntBig().

13.1.2.36 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

pszDest	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
pszSrc	Pointer to the source buffer.
nMaxLen	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().

13.1.2.37 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

pszBuffer	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.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.
pszFormat	Controls the output format. Options are the same as for strftime(3) function.
poBrokenTime	Pointer to the broken-down time structure. May be requested with the VSIGMTime() and VSILocalTime() functions.
pszLocale	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 ajusted differently from the C/POSIX defaults. To solve these problems this option was introdiced.

Returns

Number of characters printed.

References CPLMalloc(), CPLPrintString(), and CPLsetlocale().

13.1.2.38 int CPLPrintUIntBig (char * *pszBuffer,* GUIntBig *iValue,* int *nMaxLen*)

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

Parameters

pszBuffer	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.

iValue	Numerical value to print.
nMaxLen	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns

Number of characters printed.

References CPLPrintString().

13.1.2.39 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

pszProjectDir	the directory relative to which the secondary files path should be interpreted.
pszSecondary↩ Filename	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().

13.1.2.40 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 ... equivelent 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 **CPLRead↩ Line()** (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

fp	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().

13.1.2.41 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

fp	file pointer opened with **VSIFOpenL()** (p. **??**).
nMaxCars	maximum number of characters allowed, or -1 for no limit.
papszOptions	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 readched.

Since

GDAL 1.7.0

References CPLError(), VSIFReadL(), VSIFSeekL(), and VSIFTellL().

Referenced by CPLReadLineL(), and CSLLoad2().

13.1.2.42 const char* CPLReadLineL (VSILFILE * *fp*)

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. **??**), but reading from a large file API handle.

Parameters

fp	file pointer opened with **VSIFOpenL()** (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().

13.1.2.43 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 VSI↩ Realloc() 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 VSI↩ Malloc() to allocate a new block.

Parameters

pData	existing memory block which should be copied to the new block.
nNewSize	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(), OGRFeatureDefn::AddGeomFieldDefn(), CPLHTTPParse↩ MultipartMime(), CPLOpenShared(), CSLTokenizeString2(), OGRMultiPoint::exportToWkt(), CPLODBC↩ Statement::Fetch(), OGRSpatialReference::importFromESRI(), OGR_SRSNode::InsertChild(), and OGRProj4C↩ T::TransformEx().

13.1.2.44 const char* CPLResetExtension (const char * *pszPath*, const char * *pszExt*)

Replace the extension with the provided one.

Parameters

pszPath	the input path, this string is not altered.
pszExt	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 CPLStrlcat(), and CPLStrlcpy().

13.1.2.45 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

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	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().

13.1.2.46 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

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	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().

13.1.2.47 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

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	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().

13.1.2.48 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

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	The maximum number of character to read. Less characters will be read if a null character is encountered.
bTrimSpaces	If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.
bNormalize	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().

13.1.2.49 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

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	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().

13.1.2.50 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

pszString	String containing characters to be scanned. It may be terminated with a null character.
nMaxLength	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().

13.1.2.51 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

pszKey	the key of the option
pszValue	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().

13.1.2.52 char* CPLsetlocale (int *category*, const char * *locale*)

Prevents parallel executions of setlocale().

Calling setlocale() concurrently from two or more threads is a potential data race. A mutex is used to provide a critical region so that only one thread at a time can be executing setlocale().

The return should not be freed, and copied quickly as it may be invalidated by a following next call to **CPLsetlocale()** (p. **??**).

Parameters

category	See your compiler's documentation on setlocale.
locale	See your compiler's documentation on setlocale.

Returns

See your compiler's documentation on setlocale.

Referenced by CPLPrintTime().

13.1.2.53 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 **CPLSetConfig↩Option()** (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

pszKey	the key of the option
pszValue	the value of the option, or NULL to clear a setting.

References CSLSetNameValue().

13.1.2.54 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

pszString	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::exportToPretty↩ Wkt(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::exportToWkt(), OGRPoint::exportToWkt(), OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), CPLODBC↩ Statement::GetColumns(), OGRLayer::GetFeature(), OGRFeature::GetFieldAsString(), GOA2GetAccessToken(), GOA2GetAuthorizationURL(), GOA2GetRefreshToken(), OGRSpatialReference::importFromCRSURL(), OGR↩ SpatialReference::importFromESRI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::import↩ FromURN(), OGRStyleMgr::InitStyleString(), CPLStringList::InsertString(), OGRSpatialReference::morphFromE↩ SRI(), OGRSpatialReference::morphToESRI(), OGR_G_ExportToGMLEx(), OGR_SRSNode::OGR_SRSNode(), OGRFeatureDefn::OGRFeatureDefn(), OGRGeocodeCreateSession(), OGRLayer::SetAttributeFilter(), OGR↩ UnionLayer::SetAttributeFilter(), OGRFieldDefn::SetDefault(), OGRFeature::SetField(), OGRSpatialReference::↩ SetFromUserInput(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRFieldDefn::SetName(), OGRGeomFieldDefn::SetName(), OGRFeature::SetStyleString(), OGR_SRSNode::SetValue(), and VSIReadDir↩ Recursive().

13.1.2.55 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

pszString	input string to be converted.

Returns

pointer to the same string, pszString.

13.1.2.56 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

nptr	Pointer to string to convert.
endptr	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 CPLStrtodDelim().

Referenced by CPLAtof(), CPLsscanf(), OGRFieldDefn::IsDefaultDriverSpecific(), and OGRFeature::SetField().

13.1.2.57 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

nptr	Pointer to string to convert.
endptr	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.
point	Decimal delimiter.

Returns

Converted value, if any.

Referenced by CPLAtofDelim(), CPLAtofM(), CPLStrtod(), and CPLStrtofDelim().

13.1.2.58 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

nptr	Pointer to string to convert.
endptr	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().

13.1.2.59 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 strtof(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

nptr	Pointer to string to convert.
endptr	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.
point	Decimal delimiter.

Returns

Converted value, if any.

References CPLStrtodDelim().

Referenced by CPLStrtof().

13.1.2.60 int CPLUnlinkTree (const char * *pszPath*)

Returns

0 on successful completion, -1 if function fails.

References CPLError(), CPLFormFilename(), CPLStrdup(), CSLDestroy(), VSIRmdir(), VSIStatL(), and VSIUnlink().

13.1.2.61 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

ptr	input buffer.
nBytes	size of input buffer in bytes.
nLevel	ZLib compression level (-1 for default).
outptr	output buffer, or NULL to let the function allocate it.
nOutAvailableBytes	size of output buffer if provided, or ignored.
pnOutBytes	pointer to a size_t, 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

13.1.2.62 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

ptr	input buffer.
nBytes	size of input buffer in bytes.
outptr	output buffer, or NULL to let the function allocate it.
nOutAvailableBytes	size of output buffer if provided, or ignored.
pnOutBytes	pointer to a size_t, 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)
- void **CPLErrorSetState** (CPLErr eErrClass, int err_no, const char *pszMsg)
- 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 **CPLError()** (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 CPLError().

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 CPLError(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 determinted 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

pszCategory	name of the debugging message category.
pszFormat	printf() style format string for message to display. Remaining arguments are assumed to be for format.

References CPLGetConfigOption(), CPLsprintf(), and CPLvsnprintf().

Referenced by CPLDumpSharedList(), CPLFGets(), CPLHTTPFetch(), CPLQuadTreeGetAdvisedMaxDepth(), CPLSetErrorHandlerEx(), CPLUnescapeString(), CPLvsnprintf(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::curveToLineString(), OGRSpatialReference::Dereference(), OGRGeometry::Distance(), CPLODBCSession::EstablishSession(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGR_SRSNode::FixupOrdering(), OGRSpatialReference::GetAxis(), OGRLayer::GetGeomType(), CPLODBCStatement::GetTables(), GOA2GetAccessToken(), GOA2GetRefreshToken(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPanorama(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromUSGS(), CPLODBCDriverInstaller::InstallDriver(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGR_G_ExportToGMLEx(), OGRBuildPolygonFromEdges(), OGRGeometryFactory::organizePolygons(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetLocalCS(), OGRSpatialReference::SetProjCS(), 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

pszMessage	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 occured, 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

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

Referenced by _CPLAssert(), OGRLayer::AlterFieldDefn(), OGRGeometry::Boundary(), OGRGeometry::Buffer(), OGRLineString::CastToLinearRing(), OGRGeometry::Centroid(), OGRLayer::Clip(), OGRGeometry::Contains(), OGRGeometry::ConvexHull(), CPLAtoGIntBigEx(), CPLCloseShared(), CPLCorrespondingPaths(), CPLEscapeString(), CPLGetSymbol(), CPLHTTPFetch(), CPLHTTPParseMultipartMime(), CPLMalloc(), CPLParseXMLString(), CPLQuadTreeInsert(), CPLReadLine2L(), CPLRealloc(), CPLSerializeXMLTreeToFile(), CPLsscanf(), CPLStrdup(), CPLUnlinkTree(), CPLVirtualMemDerivedNew(), CPLVirtualMemFileMapNew(), CPLVirtualMemNew(), OGRLayer::CreateField(), OGRLayer::CreateGeomField(), OGRGeometry::Crosses(), CSLLoad2(), OGRGeometryFactory::curveToLineString(), OGRLayer::DeleteField(), OGRGeometry::Difference(), OGRGeometry::Disjoint(), OGRGeometry::Distance(), OGRLayer::Erase(), OGRGeometry::exportToJson(), OGRGeometry::exportToKML(), OGRSpatialReference::exportToMICoordSys(), OGRSpatialReference::exportToProj4(), CPLODBCStatement::Fetch(), OGRGenSQLResultsLayer::GetExtent(), OGRUnionLayer::GetExtent(), OGRFeature::GetFieldAsInteger(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetGeomFieldDefn(), OGRSimpleCurve::getSubLine(), GOA2GetAccessToken(), GOA2GetRefreshToken(), OGRUnionLayer::ICreateFeature(), OGRLayer::Identity(), OGRSpatialReference::importFromCRSURL(), OGRSpatialReference::importFromEPSGA(), OGRSpatialReference::importFromMICoordSys(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPanorama(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromUrl(), OGRSpatialReference::importFromURN(), OGRSpatialReference::importFromUSGS(), OGRSimpleCurve::importFromWkb(), OGRMultiSurface::importFromWkt(), OGRSpatialReference::importFromWMSAUTO(), CPLStringList::InsertStringDirectly(), OGRLayer::Intersection(), OGRGeometry::Intersection(), OGRUnionLayer::ISetFeature(), OGR_Dr_CopyDataSource(), OGR_DS_CreateLayer(), OGR_F_GetFieldDefnRef(), 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(), OGR_G_SetPointCount(), OGR_G_SetPoints(), OGRBuildPolygonFromEdges(), OGRCreateCoordinateTransformation(), OGRGeocode(), OGRGeocodeCreateSession(), OGRGeocodeReverse(), OGRGeometryFactory::organizePolygons(), OSRCalcInvFlattening(), OSRCalcSemiMinorFromInvFlattening(), OGRGeometry::Overlaps(), OGRGeometry::Polygonize(), OGRSimpleCurve::Project(), OGRLayer::ReorderField(), OGRLayer::ReorderFields(), OGRSimpleCurve::segmentize(), OGRSpatialReference::SetCompoundCS(), OGRFieldDefn::SetDefault(), OGRSpatialReference::SetEckert(), OGRFeature::SetField(), OGRSimpleCurve::setNumPoints(), OGRWarpedLayer::SetSpatialFilter(), OGRLayer::SetSpatialFilter(), OGRUnionLayer::SetSpatialFilter(), OGRSpatialReference::SetStatePlane(), OGRFieldDefn::SetSubType(), OGRFieldDefn::SetType(), OGRSpatialReference::SetWagner(), OGRGeometry::Simplify(), OGRGeometry::SimplifyPreserveTopology(), OGRLayer::SymDifference(), OGRGeometry::SymDifference(), OGRGeometry::Touches(), OGRSimpleCurve::transform(), OGRProj4CT::TransformEx(), OGRLayer::Union(), OGRGeometry::Union(), OGRGeometry::UnionCascaded(), OGRLayer::Update(), OGRFeature::Validate(), VSIIngestFile(), 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 OGRLayer::Clip(), CPLParseXMLString(), CSLLoad2(), OGRLayer::Erase(), OGRLayer::Identity(), OGRSpatialReference::importFromUrl(), OGRLayer::Intersection(), OGRLayer::SymDifference(), OGRLayer::Union(), and OGRLayer::Update().

13.2.2.6 void CPLErrorSetState (CPLErr *eErrClass*, int *err_no*, const char ∗ *pszMsg*)

Restore an error state, without emitting an error.

Can be useful if a routine might call **CPLErrorReset()** (p. **??**) and one wants to preserve the previous error state.

Since

GDAL 2.0

13.2.2.7 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

13.2.2.8 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.

13.2.2.9 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().

13.2.2.10 CPLErr CPLGetLastErrorType (void)

Fetch the last error type.

Fetches the last error type posted with **CPLError()** (p. **??**), that hasn't been cleared by **CPLErrorReset()** (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().

13.2.2.11 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 **CPL↩ PushErrorHandler()** (p. **??**) call. This method has no effect if there are no error handlers on the current threads error handler stack.

13.2.2.12 void CPLPushErrorHandler (CPLErrorHandler *pfnErrorHandlerNew*)

Push a new CPLError 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 **CPLSetErrorHandler()** (p. **??**) docs have further information on how CPLError handlers work.

Parameters

pfnError↩ HandlerNew	new error handler function.

References CPLPushErrorHandlerEx().

13.2.2.13 void CPLPushErrorHandlerEx (CPLErrorHandler *pfnErrorHandlerNew*, void * *pUserData*)

Push a new CPLError 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 **CPLSetErrorHandler()** (p. **??**) docs have further information on how CPLError handlers work.

Parameters

pfnError↩ HandlerNew	new error handler function.
pUserData	User data to put on the error context.

References CPLMalloc().

Referenced by CPLPushErrorHandler().

13.2.2.14 CPLErrorHandler CPLSetErrorHandler (CPLErrorHandler *pfnErrorHandlerNew*)

Install custom error handler.

Allow the library's user to specify an 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 **CPLError()** (p. **??**) before calling the handler. If the error handler function is passed a CE_Fatal class error and returns, then **CPLError()** (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 **CPLSetErrorHandler()** (p. **??**) apply to all threads in an application, while error handlers set with CPLPushErrorHandler are thread-local. However, any error handlers pushed with CPLPush↩ ErrorHandler (and not removed with CPLPopErrorHandler) take precidence over the global error handlers set with **CPLSetErrorHandler()** (p. **??**). Generally speaking **CPLSetErrorHandler()** (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

pfnError↩ HandlerNew	new error handler function.

Returns

returns the previously installed error handler.

References CPLSetErrorHandlerEx().

13.2.2.15 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 CPLGet↩ LastErrorUserData

Parameters

pfnError↩ HandlerNew	new error handler function.
pUserData	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 fnEqual↩ Func, 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

set	the hash set

References CPLListDestroy(), _CPLList::pData, and _CPLList::psNext.

13.3.2.2 int CPLHashSetEqualPointer (const void * *elt1,* const void * *elt2*)

Equality function for arbitrary pointers

Parameters

elt1	the first arbitrary pointer to compare
elt2	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

elt1	the first string to compare. May be NULL.
elt2	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

set	the hash set.
fnIterFunc	the function called on each element.
user_data	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

elt	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

elt	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

set	the hash set
elt	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

set	the hash set
elt	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, CPLHashSetHash↩ Pointer 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

fnHashFunc	hash function. May be NULL.
fnEqualFunc	equal function. May be NULL.
fnFreeEltFunc	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

set	the hash set
elt	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

set	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 useful.
- **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

psResult	pointer to the return value of **CPLHTTPFetch()** (p. **??**)

References CSLDestroy(), CPLHTTPResult::nMimePartCount, CPLHTTPResult::pabyData, CPLMimePart↩ ::papszHeaders, CPLHTTPResult::papszHeaders, CPLHTTPResult::pasMimePart, CPLHTTPResult::psz↩ ContentType, and CPLHTTPResult::pszErrBuf.

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

13.4.2.2 int CPLHTTPEnabled (void)

Return if CPLHTTP services can be useful.

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

pszURL	valid URL recognized by underlying download library (libcurl)
papszOptions	option list as a NULL-terminated array of strings. May be NULL. The following options are handled : • TIMEOUT=val, where val is in seconds • HEADERS=val, where val is an extra header to use when getting a web page. For example "Accept: application/x-ogcwkt" • HTTPAUTH=[BASIC/NTLM/GSSNEGOTIATE/ANY] to specify an authentication scheme to use. • USERPWD=userid:password to specify a user and password for authentication • POSTFIELDS=val, where val is a nul-terminated string to be passed to the server with a POST request. • PROXY=val, to make requests go through a proxy server, where val is of the form proxy.server.com:port_number • PROXYUSERPWD=val, where val is of the form username:password • PROXYAUTH=[BASIC/NTLM/DIGEST/ANY] to specify an proxy authentication scheme to use. • NETRC=[YES/NO] to enable or disable use of $HOME/.netrc, default YES. • CUSTOMREQUEST=val, where val is GET, PUT, POST, DELETE, etc.. (GDAL >= 1.9.0) • COOKIE=val, where val is formatted as COOKIE1=VALUE1; COOKIE2=VALUE2; ... • MAX_RETRY=val, where val is the maximum number of retry attempts if a 503 or 504 HTTP error occurs. Default is 0. (GDAL >= 2.0) • RETRY_DELAY=val, where val is the number of seconds between retry attempts. Default is 30. (GDAL >= 2.0)

Alternatively, if not defined in the papszOptions arguments, the PROXY, PROXYUSERPWD, PROXYAUTH, NE↩ TRC, MAX_RETRY and RETRY_DELAY values are searched in the configuration options named GDAL_HTTP_↩ PROXY, GDAL_HTTP_PROXYUSERPWD, GDAL_PROXY_AUTH, GDAL_HTTP_NETRC, GDAL_HTTP_MAX↩ _RETRY and GDAL_HTTP_RETRY_DELAY.

Returns

a CPLHTTPResult* structure that must be freed by **CPLHTTPDestroyResult()** (p. **??**), or NULL if libcurl support is disabled

References CPLCalloc(), CPLDebug(), CPLError(), CPLGetConfigOption(), CPLMalloc(), CPLSetConfigOption(), CPLStrdup(), CSLDestroy(), CSLTestBoolean(), CPLHTTPResult::nDataLen, CPLHTTPResult::nStatus, CPLHT↩ TPResult::pabyData, CPLHTTPResult::papszHeaders, CPLHTTPResult::pszContentType, CPLHTTPResult::psz↩ ErrBuf, and VSIGetMemFileBuffer().

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

psResult	pointer to the return value of **CPLHTTPFetch()** (p. **??**)

Returns

TRUE if the message contains MIME multipart message.

References CPLError(), CPLRealloc(), CSLCount(), CSLDestroy(), CPLMimePart::nDataLen, CPLHTTPResult::n↩ DataLen, CPLHTTPResult::nMimePartCount, CPLMimePart::pabyData, CPLHTTPResult::pabyData, CPLMime↩ Part::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

pszRefresh↩ Token	the refresh token from **GOA2GetRefreshToken()** (p. **??**).
pszScope	the scope for which it is valid.

Returns

access token, to be freed with CPLFree(), null on failure.

References CPLStringList::AddString(), CPLDebug(), CPLError(), CPLGetConfigOption(), CPLHTTPDestroy↩ Result(), 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

pszScope	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

pszAuthToken	the authorization token from the user.
pszScope	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

psList	pointer to list head.
pData	pointer to inserted data object. May be NULL.

Returns

pointer to the head of modified list.

References CPLListGetLast(), CPLMalloc(), _CPLList::pData, and _CPLList::psNext.

Referenced by CPLListInsert().

13.5.3.2 int CPLListCount (CPLList ∗ *psList*)

Return the number of elements in a list.

Parameters

psList	pointer to list head.

Returns

number of elements in a list.

References _CPLList::psNext.

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

psList	pointer to list head.

References _CPLList::psNext.

Referenced by CPLHashSetDestroy().

13.5.3.4 CPLList* CPLListGet (CPLList * *psList,* int *nPosition*)

Return the pointer to the specified element in a list.

Parameters

psList	pointer to list head.
nPosition	the index of the element in the list, 0 being the first element

Returns

pointer to the specified element in a list.

References _CPLList::psNext.

13.5.3.5 void* CPLListGetData (CPLList * *psElement*)

Return pointer to the data object contained in given list element.

Parameters

psElement	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

psList	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

psElement	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

psList	pointer to list head.
pData	pointer to inserted data object. May be NULL.
nPosition	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 CPLList* CPLListRemove (CPLList * *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

psList	pointer to list head.
nPosition	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.
- void **CPLAddXMLAttributeAndValue** (**CPLXMLNode** *psParent, const char *pszName, const char *psz↩ Value)

  Create an attribute 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 CPLAddXMLAttributeAndValue (CPLXMLNode * *psParent*, const char * *pszName*, const char * *pszValue*)

Create an attribute and text value.

This is function is a convenient short form for:

```
1 CPLXMLNode *psAttributeNode;
2
3 psAttributeNode = CPLCreateXMLNode( psParent, CXT_Attribute, pszName );
4 CPLCreateXMLNode( psAttributeNode, CXT_Text, pszValue );
```

It creates a CXT_Attribute node, with a CXT_Text child, and attaches the element to the passed parent.

Parameters

psParent	the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.
pszName	the attribute name to create.
pszValue	the text to attach to the attribute. Must not be NULL.

Since

GDAL 2.0

References CPLCreateXMLNode(), CXT_Attribute, and CXT_Text.

13.6.4.2 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

psParent	the node to attach the child to. May not be NULL.
psChild	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.

13.6.4.3 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

psOlderSibling	the node to attach the sibling after.
psNewSibling	the node to add at the end of psOlderSiblings psNext chain.

References CPLXMLNode::psNext.

13.6.4.4 void CPLCleanXMLElementName (char * *pszTarget*)

Make string into safe XML token.

Modifiies 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

pszTarget	the string to be adjusted. It is altered in place.

13.6.4.5 CPLXMLNode* CPLCloneXMLTree (CPLXMLNode * *psTree*)

Copy tree.

Creates a deep copy of a **CPLXMLNode** (p. **??**) tree.

Parameters

psTree	the tree to duplicate.

Returns

a copy of the whole tree.

References CPLCreateXMLNode(), CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

13.6.4.6 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:

```
1 CPLXMLNode *psTextNode;
2 CPLXMLNode *psElementNode;
3
4 psElementNode = CPLCreateXMLNode( psParent, CXT_Element, pszName );
5 psTextNode = CPLCreateXMLNode( psElementNode, CXT_Text, pszValue );
6
7 return psElementNode;
```

It creates a CXT_Element node, with a CXT_Text child, and attaches the element to the passed parent.

Parameters

psParent	the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.
pszName	the element name to create.
pszValue	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.

13.6.4.7 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

poParent	the parent to which this node should be attached as a child. May be NULL to keep as free standing.
eType	the type of the newly created node
pszText	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 CPLAddXMLAttributeAndValue(), CPLCloneXMLTree(), CPLCreateXMLElementAndValue(), and CPLSetXMLValue().

13.6.4.8 void CPLDestroyXMLNode (CPLXMLNode * *psNode*)

Destroy a tree.

This function frees resources associated with a **CPLXMLNode** (p. **??**) and all its children nodes.

Parameters

psNode	the tree to free.

References CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by CPLParseXMLString(), OGRSpatialReference::exportToXML(), OGRSpatialReference::importFromXML(), and OGR_G_CreateFromGML().

13.6.4.9 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. Seaching 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

psRoot	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
pszPath	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().

13.6.4.10 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

psRoot	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
pszPath	the list of element names in the path (dot separated). An empty path means get the value of the psRoot node.
pszDefault	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.

13.6.4.11 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

pszFilename	the file to open.

Returns

NULL on failure, or the document tree on success.

References CPLParseXMLString(), and VSIIngestFile().

13.6.4.12 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 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

pszString	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().

13.6.4.13 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

psParent	the node to the child is attached to.
psChild	the child to remove.

Returns

TRUE on success or FALSE if the child was not found.

References CPLXMLNode::psChild, and CPLXMLNode::psNext.

13.6.4.14 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 childs first child, and so on.

Use **CPLGetXMLNode()** (p. **??**) to find a specific child, or along a specific node path.

Parameters

psRoot	the subtree to search. This should be a node of type CXT_Element. NULL is safe.
pszElement	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.

13.6.4.15 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

psNode	the node to serialize.

Returns

the document on success or NULL on failure.

References CPLMalloc(), and CPLXMLNode::psNext.

Referenced by CPLSerializeXMLTreeToFile(), and OGRSpatialReference::exportToXML().

13.6.4.16 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

psTree	the document tree to write.
pszFilename	the name of the file to write to.

Returns

TRUE on success, FALSE otherwise.

References CPLError(), CPLSerializeXMLTree(), VSIFCloseL(), VSIFOpenL(), and VSIFWriteL().

13.6.4.17 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

psRoot	the subdocument to be updated.
pszPath	the dot seperated path to the target element/attribute.
pszValue	the text value to assign.

Returns

TRUE on success.

References CPLCreateXMLNode(), CPLStrdup(), CSLDestroy(), CXT_Attribute, CXT_Element, CXT_Text, CPL↩ XMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

13.6.4.18 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

psRoot	the document to operate on.
pszNamespace	the name space prefix (not including colon), or NULL.
bRecurse	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>
#include <strings.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 **CPLQuadTreeInsertWithBounds** (**CPLQuadTree** *hQuadtree, void *hFeature, const **CPLRectObj** *psBounds)
- void ** **CPLQuadTreeSearch** (const **CPLQuadTree** *hQuadtree, const **CPLRectObj** *pAoi, int *pn↩ FeatureCount)
- 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

pGlobalBounds	a pointer to the global extent of all the elements that will be inserted
pfnGetBounds	a user provided function to get the bounding box of the inserted elements. If it is set to NU↩ LL, then **CPLQuadTreeInsertWithBounds()** (p. **??**) must be used, and extra memory will be used to keep features bounds in the quad tree.

Returns

a newly allocated quadtree

References CPLMalloc().

13.9.2.2 void CPLQuadTreeDestroy (CPLQuadTree * *hQuadTree*)

Destroy a quadtree

Parameters

hQuadTree	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

hQuadTree	the quad tree
pfnForeach	the function called on each element.
pUserData	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

nExpected↩ Features	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

hQuadTree	the quad tree
hFeature	the feature to insert

References CPLError().

13.9.2.6 void CPLQuadTreeInsertWithBounds (CPLQuadTree * *hQuadTree*, void * *hFeature*, const CPLRectObj * *psBounds*)

Insert a feature into a quadtree

Parameters

hQuadTree	the quad tree

hFeature	the feature to insert
psBounds	bounds of the feature

13.9.2.7 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

hQuadTree	the quad tree
pAoi	the pointer to the area of interest
pnFeatureCount	the user data provided to the function.

Returns

an array of features that must be freed with CPLFree

13.9.2.8 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

hQuadTree	the quad tree
nBucketCapacity	the maximum capactiy of a node of a quadtree

13.9.2.9 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

hQuadTree	the quad tree
nMaxDepth	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 *pszDelimeter, int nCSLTFlags)
- char ** **CSLLoad** (const char *pszFname)
- char ** **CSLLoad2** (const char *pszFname, int nMaxLines, int nMaxCols, char **papszOptions)
- int **CSLFindString** (char **, const char *)
- int **CSLFindStringCaseSensitive** (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)
- 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 **CPLvsnprintf** (char *str, size_t size, const char *fmt, va_list args)
- int **CPLsnprintf** (char *str, size_t size, const char *fmt,...)
- int **CPLsprintf** (char *str, const char *fmt,...)
- int **CPLprintf** (const char *fmt,...)
- int **CPLsscanf** (const char *str, const char *fmt,...)
- 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 *psz↩ DstEncoding)
- int **CPLIsUTF8** (const char *pabyData, int nLen)
- char * **CPLForceToASCII** (const char *pabyData, int nLen, char chReplacementChar)
- int **CPLStrlenUTF8** (const char *pszUTF8Str)
- **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

nBytes	number of bytes of binary data in pabyData.
pabyData	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

pszEncoding	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 reconstitued 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 equivelent (<, >, " 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

pszInput	the string to escape.
nLength	The number of bytes of data to preserve. If this is -1 the strlen(pszString) function will be used to compute the length.
nScheme	the encoding scheme to use.

Returns

an escaped, zero terminated string that should be freed with CPLFree() when no longer needed.

References CPLError(), CPLMalloc(), CPLsprintf(), 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

pabyData	input string to test
nLen	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.
chReplacementChar	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

pszValue	the string to analyze

Returns

returns the type of the value contained in the string.

13.10.2.6 int CPLIsUTF8 (const char * *pabyData*, int *nLen*)

Test if a string is encoded as UTF-8.

Parameters

pabyData	input string to test
nLen	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

Referenced by OGRFeature::Validate().

13.10.2.7 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 delimeter when forming name and value strings.

Eventually CSLFetchNameValue() and friends may be modified to use **CPLParseNameValue()** (p. **??**).

Parameters

pszNameValue	string in "NAME=VALUE" format.
ppszKey	optional pointer though which to return the name portion.

Returns

the value portion (pointing into original string).

References CPLMalloc().

Referenced by CSLMerge(), CSLSetNameValueSeparator(), and OGRGeometryFactory::curveToLineString().

13.10.2.8 int CPLprintf (const char * *fmt*, ...)

printf() wrapper that is not sensitive to LC_NUMERIC settings.

This function has the same contract as standard printf(), except that formatting of floating-point numbers will use decimal point, whatever the current locale is set.

Parameters

fmt	formatting string
...	arguments

Returns

the number of characters (excluding terminating nul) written in output buffer.

Since

GDAL 2.0

References CPLvsnprintf().

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

pszSource	a NULL terminated string.
pszSrcEncoding	the source encoding.
pszDstEncoding	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

pwszSource	the source wchar_t string, terminated with a 0 wchar_t.
pszSrcEncoding	the source encoding, typically CPL_ENC_UCS2.
pszDstEncoding	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

pszSource	input multi-byte character string.
pszSrcEncoding	source encoding, typically CPL_ENC_UTF8.
pszDstEncoding	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 int CPLsnprintf (char * *str,* size_t *size,* const char * *fmt,* ...)

snprintf() wrapper that is not sensitive to LC_NUMERIC settings.

This function has the same contract as standard snprintf(), except that formatting of floating-point numbers will use decimal point, whatever the current locale is set.

Parameters

str	output buffer
size	size of the output buffer (including space for terminating nul)
fmt	formatting string
...	arguments

Returns

the number of characters (excluding terminating nul) that would be written if size is big enough

Since

GDAL 2.0

References CPLvsnprintf().

Referenced by CPLPrintDouble(), and OGRFeature::GetFieldAsString().

13.10.2.13 int CPLsprintf (char * *str,* const char * *fmt,* ...)

sprintf() wrapper that is not sensitive to LC_NUMERIC settings.

This function has the same contract as standard sprintf(), except that formatting of floating-point numbers will use decimal point, whatever the current locale is set.

Parameters

str	output buffer (must be large enough to hold the result)
fmt	formatting string
...	arguments

Returns

the number of characters (excluding terminating nul) written in output buffer.

Since

GDAL 2.0

References CPLvsnprintf().

Referenced by CPLDebug(), CPLEscapeString(), CPLPrintDouble(), CSLSetNameValue(), OGRSpatial↩ Reference::exportToProj4(), CPLString::FormatC(), and OGRFeature::SetField().

13.10.2.14 int CPLsscanf (const char * *str*, const char * *fmt*, ...)

sscanf() wrapper that is not sensitive to LC_NUMERIC settings.

This function has the same contract as standard sscanf(), except that formatting of floating-point numbers will use decimal point, whatever the current locale is set.

CAUTION: only works with a very limited number of formatting strings, consisting only of "%lf" and regular characters.

param str input string

Parameters

fmt	formatting string
...	arguments

Returns

the number of matched patterns;

Since

GDAL 2.0

References CPLError(), and CPLStrtod().

13.10.2.15 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 occured !\n");
```

Parameters

pszDest	destination buffer. Must be NUL terminated before running CPLStrlcat
pszSrc	source string. Must be NUL terminated
nDestSize	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().

13.10.2.16 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 occured !\n");
```

Parameters

pszDest	destination buffer
pszSrc	source string. Must be NUL terminated
nDestSize	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 CPLStrlcat().

13.10.2.17 int CPLStrlenUTF8 (const char * *pszUTF8Str*)

Return the number of UTF-8 characters of a nul-terminated string.

This is different from strlen() which returns the number of bytes.

Parameters

pszUTF8Str	a nul-terminated UTF-8 string

Returns

the number of UTF-8 characters.

Referenced by OGRFeature::Validate().

13.10.2.18 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. Useful to test if a string contains at least nMaxLen characters without reading the full string up to the NUL terminating character.

Parameters

pszStr	a NUL terminated string
nMaxLen	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().

13.10.2.19 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

pszInput	the input string. This is a zero terminated string.
pnLength	location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.
nScheme	the escaped scheme to undo (see **CPLEscapeString()** (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().

Referenced by OGRFeature::FillUnsetWithDefault().

13.10.2.20 CPLString CPLURLAddKVP (const char * *pszURL,* const char * *pszKey,* const char * *pszValue*)

Return a new URL with a new key=value pair.

Parameters

pszURL	the URL.
pszKey	the key to find.
pszValue	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().

13.10.2.21 CPLString CPLURLGetValue (const char * *pszURL*, const char * *pszKey*)

Return the value matching a key from a key=value pair in a URL.

Parameters

pszURL	the URL.
pszKey	the key to find.

Returns

the value of empty string if not found.

Since

GDAL 1.9.0

References CPLString::ifind().

13.10.2.22 int CPLvsnprintf (char * *str*, size_t *size*, const char * *fmt*, va_list *args*)

vsnprintf() wrapper that is not sensitive to LC_NUMERIC settings.

This function has the same contract as standard vsnprintf(), except that formatting of floating-point numbers will use decimal point, whatever the current locale is set.

Parameters

str	output buffer
size	size of the output buffer (including space for terminating nul)
fmt	formatting string
args	arguments

Returns

the number of characters (excluding terminating nul) that would be written if size is big enough

Since

GDAL 2.0

References CPLDebug().

Referenced by CPLDebug(), CPLprintf(), CPLsnprintf(), and CPLsprintf().

13.10.2.23 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

papszStrList	the string list to count.

Returns

the number of entries.

Referenced by CPLStringList::Count(), CPLCorrespondingPaths(), CPLHTTPParseMultipartMime(), CSLDuplicate(), CSLSetNameValueSeparator(), OGRFeature::Equal(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::GetAttrNode(), OGRStyleTable::GetStyleName(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromWMSAUTO(), OGRStyleTable::IsExist(), OGRFeature::SetField(), OGRSpatialReference::SetNode(), and VSIReadDirRecursive().

13.10.2.24 void CSLDestroy (char ** papszStrList)

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

Parameters

papszStrList	the list to free.

Referenced by OGRStyleTable::Clear(), CPLODBCStatement::Clear(), CPLStringList::Clear(), CPLGetXMLNode(), CPLHTTPDestroyResult(), CPLHTTPFetch(), CPLHTTPParseMultipartMime(), CPLSetXMLValue(), CPLUnlinkTree(), OGRSpatialReference::exportToPCI(), OGR_SRSNode::exportToWkt(), OGRSpatialReference::GetAttrNode(), OGRStyleMgr::GetPart(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromWMSAUTO(), OGRStyleTable::LoadStyleTable(), OGR_G_ExportToGMLEx(), OGRFeature::SetField(), OGRUnionLayer::SetIgnoredFields(), OGRSpatialReference::SetNode(), OGRFeature::UnsetField(), and VSIReadDirRecursive().

13.10.2.25 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

papszStrList	the input string list.

Returns

newly allocated copy.

References CPLMalloc(), CPLStrdup(), and CSLCount().

Referenced by OGRStyleTable::Clone(), CSLMerge(), OGRFeature::SetField(), and OGRUnionLayer::SetIgnoredFields().

13.10.2.26 int CSLFindName (char ** *papszStrList*, const char * *pszName*)

Find StringList entry with given key name.

Parameters

papszStrList	the string list to search.
pszName	the key value to look for (case insensitive).

Returns

-1 on failure or the list index of the first occurance matching the given key.

Referenced by CPLStringList::FindName().

13.10.2.27 int CSLFindString (char ** *papszList*, const char * *pszTarget*)

Find a string within a string list (case insensitive).

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

papszList	the string list to be searched.
pszTarget	the string to be searched for.

Returns

the index of the string within the list or -1 on failure.

Referenced by OGR_SRSNode::FixupOrdering().

13.10.2.28 int CSLFindStringCaseSensitive (char ** *papszList*, const char * *pszTarget*)

Find a string within a string list(case sensitive)

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.

Parameters

papszList	the string list to be searched.
pszTarget	the string to be searched for.

Returns

the index of the string within the list or -1 on failure.

Since

GDAL 2.0

13.10.2.29 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

pszFname	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().

13.10.2.30 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

pszFname	the name of the file to read.
nMaxLines	maximum number of lines to read before stopping, or -1 for no limit.
nMaxCols	maximum number of characters in a line before stopping, or -1 for no limit.
papszOptions	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(), VSIFEofL(), and VSIFOpenL().

Referenced by CSLLoad().

13.10.2.31 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 precidence.

Parameters

papszOrig	the original list, being modified.
papszOverride	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().

13.10.2.32 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

papszHaystack	the string list to be searched.
pszNeedle	the substring to be searched for.

Returns

the index of the string within the list or -1 on failure.

13.10.2.33 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

papszList	the original list, the modified version is returned.
pszName	the name to be assigned a value. This should be a well formed token (no spaces or very special characters).
pszValue	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(), and CPLsprintf().

Referenced by CPLSetConfigOption(), CPLSetThreadLocalConfigOption(), and CSLMerge().

13.10.2.34 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 desireable to include some white space in the new separator. eg. ": " or " = ".

Parameters

papszList	the list to update. Component strings may be freed but the list array will remain at the same location.

pszSeparator	the new separator string to insert.

References CPLMalloc(), CPLParseNameValue(), and CSLCount().

13.10.2.35 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

pszValue	the string should be tested.

Returns

TRUE or FALSE.

Referenced by OGRLayer::Clip(), CPLHTTPFetch(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRGeometry::dumpReadable(), OGRFeature::DumpReadable(), OGRLayer::Erase(), OGRSpatialReference::exportToProj4(), CPLStringList::FetchBoolean(), OGRFeatureDefn::GetGeomType(), OGRLayer::Identity(), OGRLayer::Intersection(), OGR_G_CreateFromGML(), OGR_G_ExportToGMLEx(), OGRGeocodeCreateSession(), OGRGeometryFactory::organizePolygons(), OGRFeature::SetField(), OGRLayer::SymDifference(), OGRSimpleCurve::transform(), OGRLayer::Union(), OGRLayer::Update(), and OGRSpatialReference::Validate().

13.10.2.36 char** CSLTokenizeString2 (const char * *pszString,* const char * *pszDelimiters,* int *nCSLTFlags*)

Tokenize a string.

This function will split a string into tokens based on specified' delimeter(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.

```
1 char **papszTokens;
2 int i;
3
4 papszTokens =
5     CSLTokenizeString2( pszCommand, " \t\n",
6                         CSLT_HONOURSTRINGS | CSLT_ALLOWEMPTYTOKENS );
7
8 for( i = 0; papszTokens != NULL && papszTokens[i] != NULL; i++ )
9     printf( "arg %d: '%s'", papszTokens[i] );
10 CSLDestroy( papszTokens );
```

Parameters

pszString	the string to be split into tokens.
pszDelimiters	one or more characters to be used as token delimeters.
nCSLTFlags	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(), OGR_G_ExportToGMLEx(), and OGRFeature::SetField().

13.11 cpl_virtualmem.h File Reference

```
#include "cpl_port.h"
#include "cpl_vsi.h"
```

Typedefs

- typedef struct **CPLVirtualMem** **CPLVirtualMem**
- typedef void(∗ **CPLVirtualMemCachePageCbk**) (**CPLVirtualMem** ∗ctxt, size_t nOffset, void ∗pPageToFill, size_t nToFill, void ∗pUserData)
- typedef void(∗ **CPLVirtualMemUnCachePageCbk**) (**CPLVirtualMem** ∗ctxt, size_t nOffset, const void ∗pPageToBeEvicted, size_t nToBeEvicted, void ∗pUserData)
- typedef void(∗ **CPLVirtualMemFreeUserData**) (void ∗pUserData)

Enumerations

- enum **CPLVirtualMemAccessMode** { **VIRTUALMEM_READONLY**, **VIRTUALMEM_READONLY_ENFORCED**, **VIRTUALMEM_READWRITE** }

Functions

- size_t **CPLGetPageSize** (void)
- **CPLVirtualMem** ∗ **CPLVirtualMemNew** (size_t nSize, size_t nCacheSize, size_t nPageSizeHint, int bSingleThreadUsage, **CPLVirtualMemAccessMode** eAccessMode, **CPLVirtualMemCachePageCbk** pfnCachePage, **CPLVirtualMemUnCachePageCbk** pfnUnCachePage, **CPLVirtualMemFreeUserData** pfnFreeUserData, void ∗pCbkUserData)
- int **CPLIsVirtualMemFileMapAvailable** (void)
- **CPLVirtualMem** ∗ **CPLVirtualMemFileMapNew** (VSILFILE ∗fp, vsi_l_offset nOffset, vsi_l_offset nLength, **CPLVirtualMemAccessMode** eAccessMode, **CPLVirtualMemFreeUserData** pfnFreeUserData, void ∗pCbkUserData)
- **CPLVirtualMem** ∗ **CPLVirtualMemDerivedNew** (**CPLVirtualMem** ∗pVMemBase, vsi_l_offset nOffset, vsi_l_offset nSize, **CPLVirtualMemFreeUserData** pfnFreeUserData, void ∗pCbkUserData)
- void **CPLVirtualMemFree** (**CPLVirtualMem** ∗ctxt)
- void ∗ **CPLVirtualMemGetAddr** (**CPLVirtualMem** ∗ctxt)
- size_t **CPLVirtualMemGetSize** (**CPLVirtualMem** ∗ctxt)
- int **CPLVirtualMemIsFileMapping** (**CPLVirtualMem** ∗ctxt)
- **CPLVirtualMemAccessMode** **CPLVirtualMemGetAccessMode** (**CPLVirtualMem** ∗ctxt)

- size_t **CPLVirtualMemGetPageSize** (**CPLVirtualMem** ∗ctxt)
- int **CPLVirtualMemIsAccessThreadSafe** (**CPLVirtualMem** ∗ctxt)
- void **CPLVirtualMemDeclareThread** (**CPLVirtualMem** ∗ctxt)
- void **CPLVirtualMemUnDeclareThread** (**CPLVirtualMem** ∗ctxt)
- void **CPLVirtualMemPin** (**CPLVirtualMem** ∗ctxt, void ∗pAddr, size_t nSize, int bWriteOp)
- void **CPLVirtualMemManagerTerminate** (void)

13.11.1 Detailed Description

Virtual memory management.

This file provides mechanism to define virtual memory mappings, whose content is allocated transparently and filled on-the-fly. Those virtual memory mappings can be much larger than the available RAM, but only parts of the virtual memory mapping, in the limit of the allowed the cache size, will actually be physically allocated.

This exploits low-level mechanisms of the operating system (virtual memory allocation, page protection and handler of virtual memory exceptions).

It is also possible to create a virtual memory mapping from a file or part of a file.

The current implementation is Linux only.

13.11.2 Typedef Documentation

13.11.2.1 typedef struct CPLVirtualMem CPLVirtualMem

Opaque type that represents a virtual memory mapping.

13.11.2.2 typedef void(∗ CPLVirtualMemCachePageCbk) (CPLVirtualMem ∗ctxt, size_t nOffset, void ∗pPageToFill, size_t nToFill, void ∗pUserData)

Callback triggered when a still unmapped page of virtual memory is accessed. The callback has the responsibility of filling the page with relevant values

Parameters

ctxt	virtual memory handle.
nOffset	offset of the page in the memory mapping.
pPageToFill	address of the page to fill. Note that the address might be a temporary location, and not at **CPLVirtualMemGetAddr()** (p. **??**) + nOffset.
nToFill	number of bytes of the page.
pUserData	user data that was passed to **CPLVirtualMemNew()** (p. **??**).

13.11.2.3 typedef void(∗ CPLVirtualMemFreeUserData) (void ∗pUserData)

Callback triggered when a virtual memory mapping is destroyed.

Parameters

pUserData	user data that was passed to **CPLVirtualMemNew()** (p. **??**).

13.11.2.4 typedef void(∗ CPLVirtualMemUnCachePageCbk) (CPLVirtualMem ∗ctxt, size_t nOffset, const void ∗pPageToBeEvicted, size_t nToBeEvicted, void ∗pUserData)

Callback triggered when a dirty mapped page is going to be freed. (saturation of cache, or termination of the virtual memory mapping).

Parameters

ctxt	virtual memory handle.
nOffset	offset of the page in the memory mapping.
pPageToBeEvicted	address of the page that will be flushed. Note that the address might be a temporary location, and not at **CPLVirtualMemGetAddr()** (p. **??**) + nOffset.
nToBeEvicted	number of bytes of the page.
pUserData	user data that was passed to **CPLVirtualMemNew()** (p. **??**).

13.11.3 Enumeration Type Documentation

13.11.3.1 enum CPLVirtualMemAccessMode

Access mode of a virtual memory mapping.

Enumerator

VIRTUALMEM_READONLY The mapping is meant at being read-only, but writes will not be prevented. Note that any content written will be lost.

VIRTUALMEM_READONLY_ENFORCED The mapping is meant at being read-only, and this will be enforced through the operating system page protection mechanism.

VIRTUALMEM_READWRITE The mapping is meant at being read-write, and modified pages can be saved thanks to the pfnUnCachePage callback

13.11.4 Function Documentation

13.11.4.1 size_t CPLGetPageSize (void)

Return the size of a page of virtual memory.

Returns

the page size.

Since

GDAL 1.11

13.11.4.2 int CPLIsVirtualMemFileMapAvailable (void)

Return if virtual memory mapping of a file is available.

Returns

TRUE if virtual memory mapping of a file is available.

Since

GDAL 1.11

13.11.4.3 void CPLVirtualMemDeclareThread (CPLVirtualMem * *ctxt*)

Declare that a thread will access a virtual memory mapping.

This function must be called by a thread that wants to access the content of a virtual memory mapping, except if the virtual memory mapping has been created with bSingleThreadUsage = TRUE.

This function must be paired with **CPLVirtualMemUnDeclareThread()** (p. **??**).

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Since

GDAL 1.11

13.11.4.4 CPLVirtualMem* CPLVirtualMemDerivedNew (CPLVirtualMem * *pVMemBase*, vsi_l_offset *nOffset*, vsi_l_offset *nSize*, CPLVirtualMemFreeUserData *pfnFreeUserData*, void * *pCbkUserData*)

Create a new virtual memory mapping derived from an other virtual memory mapping.

This may be useful in case of creating mapping for pixel interleaved data.

The new mapping takes a reference on the base mapping.

Parameters

pVMemBase	Base virtual memory mapping
nOffset	Offset in the base virtual memory mapping from which to start the new mapping.
nSize	Size of the base virtual memory mapping to expose in the the new mapping.
pfnFreeUserData	callback that is called when the object is destroyed.
pCbkUserData	user data passed to pfnFreeUserData.

Returns

a virtual memory object that must be freed by **CPLVirtualMemFree()** (p. **??**), or NULL in case of failure.

Since

GDAL 1.11

References CPLError().

13.11.4.5 CPLVirtualMem* CPLVirtualMemFileMapNew (VSILFILE * *fp*, vsi_l_offset *nOffset*, vsi_l_offset *nLength*, CPLVirtualMemAccessMode *eAccessMode*, CPLVirtualMemFreeUserData *pfnFreeUserData*, void * *pCbkUserData*)

Create a new virtual memory mapping from a file.

The file must be a "real" file recognized by the operating system, and not a VSI extended virtual file.

In VIRTUALMEM_READWRITE mode, updates to the memory mapping will be written in the file.

On Linux AMD64 platforms, the maximum value for nLength is 128 TB. On Linux x86 platforms, the maximum value for nLength is 2 GB.

Only supported on Linux for now.

Parameters

fp	Virtual file handle.
nOffset	Offset in the file to start the mapping from.
nLength	Length of the portion of the file to map into memory.

eAccessMode	Permission to use for the virtual memory mapping. This must be consistent with how the file has been opened.
pfnFreeUser↩ Data	callback that is called when the object is destroyed.
pCbkUserData	user data passed to pfnFreeUserData.

Returns

a virtual memory object that must be freed by **CPLVirtualMemFree()** (p. **??**), or NULL in case of failure.

Since

GDAL 1.11

References CPLError().

13.11.4.6 void CPLVirtualMemFree (CPLVirtualMem * *ctxt*)

Free a virtual memory mapping.

The pointer returned by **CPLVirtualMemGetAddr()** (p. **??**) will no longer be valid. If the virtual memory mapping was created with read/write permissions and that they are dirty (i.e. modified) pages, they will be flushed through the pfnUnCachePage callback before being freed.

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Since

GDAL 1.11

13.11.4.7 CPLVirtualMemAccessMode CPLVirtualMemGetAccessMode (CPLVirtualMem * *ctxt*)

Return the access mode of the virtual memory mapping.

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Returns

the access mode of the virtual memory mapping.

Since

GDAL 1.11

References VIRTUALMEM_READONLY.

13.11.4.8 void* CPLVirtualMemGetAddr (CPLVirtualMem * *ctxt*)

Return the pointer to the start of a virtual memory mapping.

The bytes in the range [p:p+CPLVirtualMemGetSize()-1] where p is the pointer returned by this function will be valid, until **CPLVirtualMemFree()** (p. **??**) is called.

Note that if a range of bytes used as an argument of a system call (such as read() or write()) contains pages that have not been "realized", the system call will fail with EFAULT. **CPLVirtualMemPin()** (p. **??**) can be used to work around this issue.

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Returns

the pointer to the start of a virtual memory mapping.

Since

GDAL 1.11

13.11.4.9 size_t CPLVirtualMemGetPageSize (CPLVirtualMem * *ctxt*)

Return the page size associated to a virtual memory mapping.

The value returned will be at least **CPLGetPageSize()** (p. **??**), but potentially larger.

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Returns

the page size

Since

GDAL 1.11

13.11.4.10 size_t CPLVirtualMemGetSize (CPLVirtualMem * *ctxt*)

Return the size of the virtual memory mapping.

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Returns

the size of the virtual memory mapping.

Since

GDAL 1.11

13.11.4.11 int CPLVirtualMemIsAccessThreadSafe (CPLVirtualMem * *ctxt*)

Return TRUE if this memory mapping can be accessed safely from concurrent threads.

The situation that can cause problems is when several threads try to access a page of the mapping that is not yet mapped.

The return value of this function depends on whether bSingleThreadUsage has been set of not in **CPLVirtual↩ MemNew()** (p. **??**) and/or the implementation.

On Linux, this will always return TRUE if bSingleThreadUsage = FALSE.

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Returns

TRUE if this memory mapping can be accessed safely from concurrent threads.

Since

GDAL 1.11

13.11.4.12 int CPLVirtualMemIsFileMapping (CPLVirtualMem * *ctxt*)

Return if the virtal memory mapping is a direct file mapping.

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Returns

TRUE if the virtal memory mapping is a direct file mapping.

Since

GDAL 1.11

13.11.4.13 void CPLVirtualMemManagerTerminate (void)

Cleanup any resource and handlers related to virtual memory.

This function must be called after the last CPLVirtualMem object has been freed.

Since

GDAL 2.0

13.11.4.14 CPLVirtualMem* CPLVirtualMemNew (size_t *nSize,* size_t *nCacheSize,* size_t *nPageSizeHint,* int *bSingleThreadUsage,* CPLVirtualMemAccessMode *eAccessMode,* CPLVirtualMemCachePageCbk *pfnCachePage,* CPLVirtualMemUnCachePageCbk *pfnUnCachePage,* CPLVirtualMemFreeUserData *pfnFreeUserData,* void * *pCbkUserData*)

Create a new virtual memory mapping.

This will reserve an area of virtual memory of size nSize, whose size might be potentially much larger than the physical memory available. Initially, no physical memory will be allocated. As soon as memory pages will be accessed, they will be allocated transparently and filled with the pfnCachePage callback. When the allowed cache size is reached, the least recently used pages will be unallocated.

On Linux AMD64 platforms, the maximum value for nSize is 128 TB. On Linux x86 platforms, the maximum value for nSize is 2 GB.

Only supported on Linux for now.

Note that on Linux, this function will install a SIGSEGV handler. The original handler will be restored by **CPL↩ VirtualMemManagerTerminate()** (p. **??**).

Parameters

nSize	size in bytes of the virtual memory mapping.
nCacheSize	size in bytes of the maximum memory that will be really allocated (must ideally fit into RAM).
nPageSizeHint	hint for the page size. Must be a multiple of the system page size, returned by **CPLGetPageSize()** (p. **??**). Minimum value is generally 4096. Might be set to 0 to let the function determine a default page size.
bSingleThreadUsage	set to TRUE if there will be no concurrent threads that will access the virtual memory mapping. This can optimize performance a bit.
eAccessMode	permission to use for the virtual memory mapping.
pfnCachePage	callback triggered when a still unmapped page of virtual memory is accessed. The callback has the responsibility of filling the page with relevant values.
pfnUnCachePage	callback triggered when a dirty mapped page is going to be freed (saturation of cache, or termination of the virtual memory mapping). Might be NULL.
pfnFreeUserData	callback that can be used to free pCbkUserData. Might be NULL
pCbkUserData	user data passed to pfnCachePage and pfnUnCachePage.

Returns

a virtual memory object that must be freed by **CPLVirtualMemFree()** (p. **??**), or NULL in case of failure.

Since

GDAL 1.11

References CPLError().

13.11.4.15 void CPLVirtualMemPin (CPLVirtualMem * *ctxt*, void * *pAddr*, size_t *nSize*, int *bWriteOp*)

Make sure that a region of virtual memory will be realized.

Calling this function is not required, but might be useful when debugging a process with tools like gdb or valgrind that do not naturally like segmentation fault signals.

It is also needed when wanting to provide part of virtual memory mapping to a system call such as read() or write(). If read() or write() is called on a memory region not yet realized, the call will fail with EFAULT.

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).
pAddr	the memory region to pin.
nSize	the size of the memory region.
bWriteOp	set to TRUE if the memory are will be accessed in write mode.

Since

GDAL 1.11

13.11.4.16 void CPLVirtualMemUnDeclareThread (CPLVirtualMem * *ctxt*)

Declare that a thread will stop accessing a virtual memory mapping.

This function must be called by a thread that will no longer access the content of a virtual memory mapping, except if the virtual memory mapping has been created with bSingleThreadUsage = TRUE.

This function must be paired with **CPLVirtualMemDeclareThread()** (p. **??**).

Parameters

ctxt	context returned by **CPLVirtualMemNew()** (p. **??**).

Since

GDAL 1.11

13.12 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 **VSIIngestFile** (VSILFILE *fp, const char *pszFilename, GByte **ppabyRet, vsi_l_offset *pnSize, GIntBig nMaxSize)

  Ingest a file into memory.
- int **VSIStatL** (const char *, VSIStatBufL *)

  Get filesystem object info.
- int **VSIStatExL** (const char *pszFilename, VSIStatBufL *psStatBuf, int nFlags)

  Get filesystem object info.
- int **VSIIsCaseSensitiveFS** (const char *pszFilename)

  Returns if the filenames of the filesystem are case sensitive.
- void * **VSIFGetNativeFileDescriptorL** (VSILFILE *)

  Returns the "native" file descriptor for the virtual handle.

- void ∗ **VSIMalloc2** (size_t nSize1, size_t nSize2)
- void ∗ **VSIMalloc3** (size_t nSize1, size_t nSize2, size_t nSize3)
- GIntBig **CPLGetPhysicalRAM** (void)
- GIntBig **CPLGetUsablePhysicalRAM** (void)
- 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 **VSIUnlink** (const char ∗pathname)

  Delete a file.
- int **VSIRename** (const char ∗oldpath, const char ∗newpath)

  Rename a file.
- void **VSIInstallMemFileHandler** (void)

  Install "memory" file system handler.
- void **VSIInstallSubFileHandler** (void)
- void **VSIInstallCurlFileHandler** (void)

  Install /vsicurl/ HTTP/FTP file system handler (requires libcurl)
- void **VSIInstallCurlStreamingFileHandler** (void)

  Install /vsicurl_streaming/ HTTP/FTP file system handler (requires libcurl)
- void **VSIInstallGZipFileHandler** (void)

  Install GZip file system handler.
- void **VSIInstallZipFileHandler** (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 nData↩ Length, int bTakeOwnership)

  Create memory "file" from a buffer.
- GByte ∗ **VSIGetMemFileBuffer** (const char ∗pszFilename, vsi_l_offset ∗pnDataLength, int bUnlinkAndSeize)

  Fetch buffer underlying memory file.
- void **VSIStdoutSetRedirection** (VSIWriteFunction pFct, FILE ∗stream)

13.12.1 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.

13.12.2 Function Documentation

13.12.2.1 GIntBig CPLGetPhysicalRAM (void)

Return the total physical RAM in bytes.

Returns

the total physical RAM in bytes (or 0 in case of failure).

Since

GDAL 2.0

Referenced by CPLGetUsablePhysicalRAM().

13.12.2.2 GIntBig CPLGetUsablePhysicalRAM (void)

Return the total physical RAM, usable by a process, in bytes.

This is the same as **CPLGetPhysicalRAM()** (p. **??**) except it will limit to 2 GB for 32 bit processes.

Note: This memory may already be partly used by other processes.

Returns

the total physical RAM, usable by a process, in bytes (or 0 in case of failure).

Since

GDAL 2.0

References CPLGetPhysicalRAM().

13.12.2.3 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

fp	file handle opened with **VSIFOpenL()** (p. **??**).

Returns

0 on success or -1 on failure.

Referenced by CPLCloseShared(), CPLSerializeXMLTreeToFile(), CSLLoad2(), and VSIIngestFile().

13.12.2.4 int VSIFEofL (VSILFILE * *fp*)

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occured during the previous read operation. The end-of-file flag is cleared by a successful **VSIFSeekL()** (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

fp	file handle opened with **VSIFOpenL()** (p. **??**).

Returns

TRUE if at EOF else FALSE.

Referenced by CSLLoad2().

13.12.2.5 int VSIFFlushL (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

fp	file handle opened with **VSIFOpenL()** (p. **??**).

Returns

0 on success or -1 on error.

13.12.2.6 void* VSIFGetNativeFileDescriptorL (VSILFILE * *fp*)

Returns the "native" file descriptor for the virtual handle.

This will only return a non-NULL value for "real" files handled by the operating system (to be opposed to GDAL virtual file systems).

On POSIX systems, this will be a integer value ("fd") cast as a void*. On Windows systems, this will be the HANDLE.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. **??**).

Returns

the native file descriptor, or NULL.

13.12.2.7 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

pszFilename	the filename to be created.
pabyData	the data buffer for the file.
nDataLength	the length of buffer in bytes.
bTakeOwnership	TRUE to transfer "ownership" of buffer or FALSE.

Returns

open file handle on created file (see **VSIFOpenL()** (p. **??**)).

References VSIInstallMemFileHandler().

13.12.2.8 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 VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

Parameters

pszFilename	the file to open. UTF-8 encoded.
pszAccess	access requested (ie. "r", "r+", "w".

Returns

NULL on failure, or the file handle.

Referenced by CPLOpenShared(), CPLSerializeXMLTreeToFile(), CSLLoad2(), and VSIIngestFile().

13.12.2.9 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

fp	file handle opened with **VSIFOpenL()** (p. **??**).
pszFormat	the printf style format string.

Returns

the number of bytes written or -1 on an error.

References VSIFWriteL().

13.12.2.10 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 VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters

pBuffer	the buffer into which the data should be read (at least nCount * nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.
fp	file handle opened with **VSIFOpenL()** (p. **??**).

Returns

number of objects successfully read.

Referenced by CPLReadLine2L(), and VSIIngestFile().

13.12.2.11 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 VSIFileHandler virtualization and may work on unusual filesystems such as in memory or /vsicurl/.

Parameters

nRanges	number of ranges to read.
ppData	array of nRanges buffer into which the data should be read (ppData[i] must be at list panSizes[i] bytes).
panOffsets	array of nRanges offsets at which the data should be read.
panSizes	array of nRanges sizes of objects to read (in bytes).
fp	file handle opened with **VSIFOpenL()** (p. **??**).

Returns

0 in case of success, -1 otherwise.

Since

GDAL 1.9.0

13.12.2.12 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 VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. **??**).
nOffset	offset in bytes.
nWhence	one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns

0 on success or -1 one failure.

Referenced by CPLReadLine2L(), and VSIIngestFile().

13.12.2.13 vsi_l_offset VSIFTellL (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

fp	file handle opened with **VSIFOpenL()** (p. **??**).

Returns

file offset in bytes.

Referenced by CPLReadLine2L(), and VSIIngestFile().

13.12.2.14 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

fp	file handle opened with **VSIFOpenL()** (p. **??**).
nNewSize	new size in bytes.

Returns

0 on success

Since

GDAL 1.9.0

13.12.2.15 size_t VSIFWriteL (const void * *pBuffer*, size_t *nSize*, size_t *nCount*, VSILFILE * *fp*)

Write bytes to file.

Writess 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

pBuffer	the buffer from which the data should be written (at least nCount * nSize bytes in size.
nSize	size of objects to read in bytes.
nCount	number of objects to read.
fp	file handle opened with **VSIFOpenL()** (p. **??**).

Returns

number of objects successfully written.

Referenced by CPLSerializeXMLTreeToFile(), and VSIFPrintfL().

13.12.2.16 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 existance.

Parameters

pszFilename	the name of the file to grab the buffer of.
pnDataLength	(file) length returned in this variable.
bUnlinkAndSeize	TRUE to remove the file, or FALSE to leave unaltered.

Returns

pointer to memory buffer or NULL on failure.

References CPLDebug().

Referenced by CPLHTTPFetch().

13.12.2.17 int VSIIngestFile (VSILFILE * *fp*, const char * *pszFilename*, GByte ** *ppabyRet*, vsi_l_offset * *pnSize*, GIntBig *nMaxSize*)

Ingest a file into memory.

Read the whole content of a file into a memory buffer.

Either fp or pszFilename can be NULL, but not both at the same time.

If fp is passed non-NULL, it is the responsibility of the caller to close it.

If non-NULL, the returned buffer is guaranteed to be NUL-terminated.

Parameters

fp	file handle opened with **VSIFOpenL()** (p. **??**).
pszFilename	filename.
ppabyRet	pointer to the target buffer. *ppabyRet must be freed with VSIFree()
pnSize	pointer to variable to store the file size. May be NULL.
nMaxSize	maximum size of file allowed. If no limit, set to a negative value.

Returns

TRUE in case of success.

Since

GDAL 1.11

References CPLError(), VSIFCloseL(), VSIFOpenL(), VSIFReadL(), VSIFSeekL(), and VSIFTellL().

Referenced by CPLParseXMLFile().

13.12.2.18 void VSIInstallCurlFileHandler (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/resource or /vsicurl/ftp://path/to/remote/resource where path/to/remote/resource is the URL of a remote resource.

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

13.12.2.19 void VSIInstallCurlStreamingFileHandler (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 dowloading, you should use the /vsicurl/ file system handler instead.

Recognized filenames are of the form /vsicurl_streaming/http://path/to/remote/resource or /vsicurl_streaming/ftp↩ ://path/to/remote/resource where path/to/remote/resource is the URL of a remote resource.

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

13.12.2.20 void VSIInstallGZipFileHandler (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/Read↩ InZip`

Since

GDAL 1.6.0

13.12.2.21 void VSIInstallMemFileHandler (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. **VSIFOpenL()** (p. **??**), VSIMkDir(), etc) is called.

This code example demonstrates using GDAL to translate from one memory buffer to another.

```
1 GByte *ConvertBufferFormat( GByte *pabyInData, vsi_l_offset nInDataLength,
2                             vsi_l_offset *pnOutDataLength )
3 {
4     // create memory file system object from buffer.
5     VSIFCloseL( VSIFileFromMemBuffer( "/vsimem/work.dat", pabyInData,
6                                       nInDataLength, FALSE ) );
7
8     // Open memory buffer for read.
9     GDALDatasetH hDS = GDALOpen( "/vsimem/work.dat", GA_ReadOnly );
10
11     // Get output format driver.
12     GDALDriverH hDriver = GDALGetDriverByName( "GTiff" );
13     GDALDatasetH hOutDS;
14
15     hOutDS = GDALCreateCopy( hDriver, "/vsimem/out.tif", hDS, TRUE, NULL,
16                              NULL, NULL );
17
18     // close source file, and "unlink" it.
19     GDALClose( hDS );
20     VSIUnlink( "/vsimem/work.dat" );
```

```
21
22     // seize the buffer associated with the output file.
23
24     return VSIGetMemFileBuffer( "/vsimem/out.tif", pnOutDataLength, TRUE );
25 }
```

Referenced by VSIFileFromMemBuffer().

13.12.2.22 void VSIInstallSparseFileHandler (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.

13.12.2.23 void VSIInstallStdinHandler (void)

Install /vsistdin/ file system handler.

A special file handler is installed that allows reading from the standard input steam.

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

13.12.2.24 void VSIInstallStdoutHandler (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

13.12.2.25 void VSIInstallSubFileHandler (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 **VSIStatL()** (p. **??**), **VSIFOpenL()** (p. **??**) and operations based on the file handle returned by **VSIFOpenL()** (p. **??**) operate properly.

13.12.2.26 void VSIInstallTarFileHandler (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

Syntaxic 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

13.12.2.27 void VSIInstallZipFileHandler (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 were 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

Syntaxic 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/Read↩InZip`

Since

GDAL 1.6.0

13.12.2.28 int VSIIsCaseSensitiveFS (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

pszFilename	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().

13.12.2.29 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().

13.12.2.30 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().

13.12.2.31 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

pszPathname	the path to the directory to create. UTF-8 encoded.
mode	the permissions mode.

Returns

0 on success or -1 on an error.

13.12.2.32 char** VSIReadDir (const char * *pszPath*)

Read names in a directory.

This function abstracts access to directory contains. 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

pszPath	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().

13.12.2.33 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

pszPathIn	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(), VSI↩ ReadDir(), and VSIStatL().

13.12.2.34 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

oldpath	the name of the file to be renamed. UTF-8 encoded.
newpath	the name the file should be given. UTF-8 encoded.

Returns

0 on success or -1 on an error.

13.12.2.35 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

pszDirname	the path of the directory to be deleted. UTF-8 encoded.

Returns

0 on success or -1 on an error.

Referenced by CPLUnlinkTree().

13.12.2.36 int VSIStatExL (const char * *pszFilename,* VSIStatBufL * *psStatBuf,* int *nFlags*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSIStatBufL structure. For portability, only use the st_size (size in bytes) and st_mode (file type). This method is similar to VSIStat(), 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

pszFilename	the path of the filesystem object to be queried. UTF-8 encoded.
psStatBuf	the structure to load with information.
nFlags	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 VSIStatL().

13.12.2.37 int VSIStatL (const char * *pszFilename,* VSIStatBufL * *psStatBuf*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSIStatBufL structure. For portability, only use the st_size (size in bytes) and st_mode (file type). This method is similar to VSIStat(), 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

pszFilename	the path of the filesystem object to be queried. UTF-8 encoded.
psStatBuf	the structure to load with information.

Returns

0 on success or -1 on an error.

References VSIStatExL().

Referenced by CPLCheckForFile(), CPLUnlinkTree(), and VSIReadDirRecursive().

13.12.2.38 void VSIStdoutSetRedirection (VSIWriteFunction *pFct,* FILE * *stream*)

Set an alternative write function and output file handle instead of fwrite() / stdout.

Parameters

pFct	Function with same signature as fwrite()
stream	File handle on which to output. Passed to pFct.

Since

GDAL 2.0

13.12.2.39 int VSIUnlink (const char * *pszFilename*)

Delete a file.

Deletes a file object from the file system.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX unlink() function.

Parameters

pszFilename	the path of the file to be deleted. UTF-8 encoded.

Returns

0 on success or -1 on an error.

Referenced by CPLUnlinkTree().

13.13 ogr_api.h File Reference

```
#include "cpl_progress.h"
#include "cpl_minixml.h"
#include "ogr_core.h"
```

Functions

- OGRErr **OGR_G_CreateFromWkb** (unsigned char *, OGRSpatialReferenceH, OGRGeometryH *, int)

  Create a geometry object of the appropriate type from it's well known binary representation.
- OGRErr **OGR_G_CreateFromWkt** (char **, OGRSpatialReferenceH, OGRGeometryH *)

  Create a geometry object of the appropriate type from it's 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.
- OGRGeometryH **OGR_G_ForceTo** (OGRGeometryH hGeom, **OGRwkbGeometryType** eTargetType, char **papszOptions)

  Convert to another geometry type.
- 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 well known binary format.
- OGRErr **OGR_G_ExportToIsoWkb** (OGRGeometryH, OGRwkbByteOrder, unsigned char *)

  Convert a geometry into SFSQL 1.2 / ISO SQL/MM Part 3 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.
- OGRErr **OGR_G_ExportToIsoWkt** (OGRGeometryH, char **)

  Convert a geometry into SFSQL 1.2 / ISO SQL/MM Part 3 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.
- OGRGeometryH **OGR_G_Value** (OGRGeometryH, double dfDistance)

  Fetch point at given distance along curve.
- 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_DEP↩ RECATED("Non standard method. Use **OGR_G_SymDifference**() instead")

  Compute symmetric difference (deprecated)
- double **OGR_G_GetArea** (OGRGeometryH) CPL_WARN_DEPRECATED("Non standard method. Use **O**↩ **GR_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 x 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_SetPointCount** (OGRGeometryH hGeom, int nNewPointCount)

  Set number of points in a 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).
- void **OGR_G_SetPoints** (OGRGeometryH hGeom, int nPointsIn, void ∗pabyX, int nXStride, void ∗pabyY, int nYStride, void ∗pabyZ, int nZStride)

  Assign all points in a point or a line string geometry.
- 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.
- int **OGR_G_HasCurveGeometry** (OGRGeometryH, int bLookForNonLinear)

  Returns if this geometry is or has curve geometry.
- OGRGeometryH **OGR_G_GetLinearGeometry** (OGRGeometryH hGeom, double dfMaxAngleStepSize↩ Degrees, char ∗∗papszOptions)

  Return, possibly approximate, linear version of this geometry.
- OGRGeometryH **OGR_G_GetCurveGeometry** (OGRGeometryH hGeom, char ∗∗papszOptions)

  Return curve version of this geometry.
- 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.
- void **OGRSetNonLinearGeometriesEnabledFlag** (int bFlag)

  Set flag to enable/disable returning non-linear geometries in the C API.
- int **OGRGetNonLinearGeometriesEnabledFlag** (void)

  Get flag to enable/disable returning non-linear geometries in the C API.
- 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. **??**).*
- **OGRFieldSubType OGR_Fld_GetSubType** (OGRFieldDefnH)

  Fetch subtype of this field.
- void **OGR_Fld_SetSubType** (OGRFieldDefnH, **OGRFieldSubType**)

  *Set the subtype of this field. This should never be done to an **OGRFieldDefn** (p. **??**) that is already part of an **OGRFeatureDefn** (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.
- int **OGR_Fld_IsNullable** (OGRFieldDefnH hDefn)

  Return whether this field can receive null values.
- void **OGR_Fld_SetNullable** (OGRFieldDefnH hDefn, int)

  Set whether this field can receive null values.
- const char * **OGR_Fld_GetDefault** (OGRFieldDefnH hDefn)

  Get default field value.
- void **OGR_Fld_SetDefault** (OGRFieldDefnH hDefn, const char *)

  Set default field value.
- int **OGR_Fld_IsDefaultDriverSpecific** (OGRFieldDefnH hDefn)

  Returns whether the default value is driver specific.
- const char * **OGR_GetFieldTypeName** (**OGRFieldType**)

  Fetch human readable name for a field type.
- const char * **OGR_GetFieldSubTypeName** (**OGRFieldSubType**)

  Fetch human readable name for a field subtype.
- int **OGR_AreTypeSubTypeCompatible** (**OGRFieldType** eType, **OGRFieldSubType** eSubType)

  Return if type and subtype are compatible.
- OGRGeomFieldDefnH **OGR_GFld_Create** (const char *, **OGRwkbGeometryType**) CPL_WARN_UNUS↩ ED_RESULT

  Create a new field geometry definition.
- void **OGR_GFld_Destroy** (OGRGeomFieldDefnH)

  Destroy a geometry field definition.
- void **OGR_GFld_SetName** (OGRGeomFieldDefnH, const char *)

  Reset the name of this field.
- const char * **OGR_GFld_GetNameRef** (OGRGeomFieldDefnH)

  Fetch name of this field.
- **OGRwkbGeometryType OGR_GFld_GetType** (OGRGeomFieldDefnH)

  Fetch geometry type of this field.
- void **OGR_GFld_SetType** (OGRGeomFieldDefnH, **OGRwkbGeometryType**)

  *Set the geometry type of this field. This should never be done to an **OGRGeomFieldDefn** (p. **??**) that is already part of an **OGRFeatureDefn** (p. **??**).*
- OGRSpatialReferenceH **OGR_GFld_GetSpatialRef** (OGRGeomFieldDefnH)

  Fetch spatial reference system of this field.
- void **OGR_GFld_SetSpatialRef** (OGRGeomFieldDefnH, OGRSpatialReferenceH hSRS)

Set the spatial reference of this field.

- int **OGR_GFld_IsNullable** (OGRGeomFieldDefnH hDefn)

  Return whether this geometry field can receive null values.
- void **OGR_GFld_SetNullable** (OGRGeomFieldDefnH hDefn, int)

  Set whether this geometry field can receive null values.
- int **OGR_GFld_IsIgnored** (OGRGeomFieldDefnH hDefn)

  Return whether this field should be omitted when fetching features.
- void **OGR_GFld_SetIgnored** (OGRGeomFieldDefnH hDefn, int)

  Set whether this field should be omitted when fetching features.
- 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.
- int **OGR_FD_GetGeomFieldCount** (OGRFeatureDefnH hFDefn)

  Fetch number of geometry fields on the passed feature definition.
- OGRGeomFieldDefnH **OGR_FD_GetGeomFieldDefn** (OGRFeatureDefnH hFDefn, int i)

  Fetch geometry field definition of the passed feature definition.
- int **OGR_FD_GetGeomFieldIndex** (OGRFeatureDefnH hFDefn, const char ∗pszName)

  Find geometry field by name.

- void **OGR_FD_AddGeomFieldDefn** (OGRFeatureDefnH hFDefn, OGRGeomFieldDefnH hGFldDefn)

  Add a new field definition to the passed feature definition.
- OGRErr **OGR_FD_DeleteGeomFieldDefn** (OGRFeatureDefnH hFDefn, int iGeomField)

  Delete an existing geometry field definition.
- int **OGR_FD_IsSame** (OGRFeatureDefnH hFDefn, OGRFeatureDefnH hOtherFDefn)

  Test if the feature definition is identical to the other one.
- 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.
- GIntBig **OGR_F_GetFieldAsInteger64** (OGRFeatureH, int)

  Fetch field value as integer 64 bit.
- 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 GIntBig * **OGR_F_GetFieldAsInteger64List** (OGRFeatureH, int, int *)

  Fetch field value as a list of 64 bit 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.

- int **OGR_F_GetFieldAsDateTimeEx** (OGRFeatureH hFeat, int iField, int *pnYear, int *pnMonth, int *pnDay, int *pnHour, int *pnMinute, float *pfSecond, int *pnTZFlag)

  Fetch field value as date and time.

- void **OGR_F_SetFieldInteger** (OGRFeatureH, int, int)

  Set field to integer value.

- void **OGR_F_SetFieldInteger64** (OGRFeatureH, int, GIntBig)

  Set field to 64 bit 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_SetFieldInteger64List** (OGRFeatureH, int, int, const GIntBig *)

  Set field to list of 64 bit 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.

- void **OGR_F_SetFieldDateTimeEx** (OGRFeatureH, int, int, int, int, int, int, float, int)

  Set field to datetime.

- int **OGR_F_GetGeomFieldCount** (OGRFeatureH hFeat)

  *Fetch number of geometry fields on this feature This will always be the same as the geometry field count for the **OGRFeatureDefn** (p. **??**).*

- OGRGeomFieldDefnH **OGR_F_GetGeomFieldDefnRef** (OGRFeatureH hFeat, int iField)

  Fetch definition for this geometry field.

- int **OGR_F_GetGeomFieldIndex** (OGRFeatureH hFeat, const char *pszName)

  Fetch the geometry field index given geometry field name.

- OGRGeometryH **OGR_F_GetGeomFieldRef** (OGRFeatureH hFeat, int iField)

  Fetch an handle to feature geometry.

- OGRErr **OGR_F_SetGeomFieldDirectly** (OGRFeatureH hFeat, int iField, OGRGeometryH hGeom)

  Set feature geometry of a specified geometry field.

- OGRErr **OGR_F_SetGeomField** (OGRFeatureH hFeat, int iField, OGRGeometryH hGeom)

  Set feature geometry of a specified geometry field.

- GIntBig **OGR_F_GetFID** (OGRFeatureH)

  Get feature identifier.

- OGRErr **OGR_F_SetFID** (OGRFeatureH, GIntBig)

  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.*
- void **OGR_F_FillUnsetWithDefault** (OGRFeatureH hFeat, int bNotNullableOnly, char ∗∗papszOptions)

  Fill unset fields with default values that might be defined.
- int **OGR_F_Validate** (OGRFeatureH, int nValidateFlags, int bEmitError)

  Validate that a feature meets constraints of its schema.
- 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.
- void **OGR_L_SetSpatialFilterEx** (OGRLayerH, int iGeomField, OGRGeometryH hGeom)

  Set a new spatial filter.
- void **OGR_L_SetSpatialFilterRectEx** (OGRLayerH, int iGeomField, double dfMinX, double dfMinY, double dfMaxX, double dfMaxY)

  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, GIntBig)

  Move read cursor to the nIndex'th feature in the current resultset.
- OGRFeatureH **OGR_L_GetFeature** (OGRLayerH, GIntBig)

  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, GIntBig)

  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_FindFieldIndex** (OGRLayerH, const char *, int bExactMatch)

  Find the index of field in a layer.
- GIntBig **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.
- OGRErr **OGR_L_GetExtentEx** (OGRLayerH, int iGeomField, **OGREnvelope** *psExtent, int bForce)

  Fetch the extent of this layer, on the specified geometry field.
- 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_CreateGeomField** (OGRLayerH hLayer, OGRGeomFieldDefnH hFieldDefn, int bForce)

  Create a new geometry 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, **OGRwkb↩ GeometryType**, 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. **??**).*
- 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_U↩ NUSED_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.
- int **OGRGetDriverCount** (void)

  Fetch the number of registered drivers.

- OGRSFDriverH **OGRGetDriver** (int)

  Fetch the indicated driver.
- OGRSFDriverH **OGRGetDriverByName** (const char ∗)

  Fetch the indicated driver.
- 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_AddStyle** (OGRStyleTableH hStyleTable, 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. This function is the same as the C++ method ***OGRStyleTable::AddStyle()*** *(*p. **??***).*
- 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)

13.13.1 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. **??**)

13.13.2 Function Documentation

13.13.2.1 int OGR_AreTypeSubTypeCompatible (OGRFieldType *eType,* OGRFieldSubType *eSubType*)

Return if type and subtype are compatible.

Parameters

eType	the field type.
eSubType	the field subtype.

Returns

TRUE if type and subtype are compatible

Since

GDAL 2.0

References OFSTBoolean, OFSTFloat32, OFSTInt16, OFSTNone, OFTInteger, OFTIntegerList, OFTReal, and OFTRealList.

Referenced by OGRFieldDefn::SetSubType(), and OGRFieldDefn::SetType().

13.13.2.2 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.

Deprecated Use GDALCreateCopy() in GDAL 2.0

Parameters

hDriver	handle to the driver on which data source creation is based.
hSrcDS	source datasource
pszNewName	the name for the new data source.
papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns

NULL is returned on failure, or a new **OGRDataSource** (p. **??**) handle on success.

References CPLError(), OGRLayer::GetLayerDefn(), and OGRFeatureDefn::GetName().

13.13.2.3 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.

Deprecated Use GDALCreate() in GDAL 2.0

Parameters

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

Returns

NULL is returned on failure, or a new **OGRDataSource** (p. **??**) handle on success.

13.13.2.4 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 ODrCDeleteData↩ Source.

Deprecated Use GDALDeleteDataset() in GDAL 2

Parameters

hDriver	handle to the driver on which data source deletion is based.

pszDataSource	the name of the datasource to delete.

Returns

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

13.13.2.5 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().

Parameters

hDriver	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.

13.13.2.6 OGRDataSourceH OGR_Dr_Open (OGRSFDriverH *hDriver,* const char * *pszName,* int *bUpdate*)

Attempt to open file with this driver.

NOTE: Starting with GDAL 2.0, it is *NOT* safe to cast the returned handle to OGRDataSource*. If a C++ object is needed, the handle should be cast to GDALDataset*. Similarly, the returned OGRSFDriverH handle should be cast to GDALDriver*, and NOT* OGRSFDriver*.

Deprecated Use GDALOpenEx() in GDAL 2.0

Parameters

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

Returns

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

13.13.2.7 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 mispelling.

Deprecated Use GDALGetMetadataItem(hDriver, GDAL_DCAP_CREATE) in GDAL 2.0

Parameters

hDriver	handle to the driver to test the capability against.
pszCap	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

13.13.2.8 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.

Deprecated Use GDALDatasetCopyLayer() in GDAL 2.0

Parameters

hDS	handle to the data source where to create the new layer
hSrcLayer	handle to the source layer.
pszNewName	the name of the layer to create.
papszOptions	a StringList of name=value options. Options are driver specific.

Returns

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

13.13.2.9 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.

Deprecated Use GDALDatasetCreateLayer() in GDAL 2.0

Parameters

hDS	The dataset handle.
pszName	the name for the new layer. This should ideally not match any existing layer on the datasource.
hSpatialRef	handle to the coordinate system to use for the new layer, or NULL if no coordinate system is available.
eType	the geometry type for the layer. Use wkbUnknown if there are no constraints on the types geometry to be written.

papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns

NULL is returned on failure, or a new **OGRLayer** (p. **??**) handle on success.

Example:

```
1 #include "ogrsf_frmts.h"
2 #include "cpl_string.h"
3
4 ...
5
6         OGRLayerH *hLayer;
7         char     **papszOptions;
8
9         if( OGR_DS_TestCapability( hDS, ODsCCreateLayer ) )
10         {
11             ...
12         }
13
14         papszOptions = CSLSetNameValue( papszOptions, "DIM", "2" );
15         hLayer = OGR_DS_CreateLayer( hDS, "NewLayer", NULL, wkbUnknown,
16                                      papszOptions );
17         CSLDestroy( papszOptions );
18
19         if( hLayer == NULL )
20         {
21             ...
22         }
```

References CPLError().

13.13.2.10 OGRErr OGR_DS_DeleteLayer (OGRDataSourceH *hDS*, int *iLayer*)

Delete the indicated layer from the datasource.

If this method is supported the ODsCDeleteLayer capability will test TRUE on the **OGRDataSource** (p. **??**).

Deprecated Use GDALDatasetDeleteLayer() in GDAL 2.0

Parameters

hDS	handle to the datasource
iLayer	the index of the layer to delete.

Returns

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

13.13.2.11 void OGR_DS_Destroy (OGRDataSourceH *hDataSource*)

Closes opened datasource and releases allocated resources.

This method is the same as the C++ method OGRDataSource::DestroyDataSource().

Deprecated Use GDALClose() in GDAL 2.0

Parameters

hDataSource	handle to allocated datasource object.

13.13.2.12 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 **OGRLayer** (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.

Deprecated Use GDALDatasetExecuteSQL() in GDAL 2.0

Parameters

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

Returns

an handle to a **OGRLayer** (p. **??**) containing the results of the query. Deallocate with **OGR_DS_ReleaseResultSet()** (p. **??**).

13.13.2.13 OGRSFDriverH OGR_DS_GetDriver (OGRDataSourceH *hDS*)

Returns the driver that the dataset was opened with.

NOTE: Starting with GDAL 2.0, it is *NOT* safe to cast the returned handle to OGRSFDriver*. If a C++ object is needed, the handle should be cast to GDALDriver*.

Deprecated Use GDALGetDatasetDriver() in GDAL 2.0

Parameters

hDS	handle to the datasource

Returns

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

13.13.2.14 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.

Deprecated Use GDALDatasetGetLayer() in GDAL 2.0

Parameters

hDS	handle to the data source from which to get the layer.
iLayer	a layer number between 0 and **OGR_DS_GetLayerCount()** (p. **??**)-1.

Returns

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

13.13.2.15 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.

Deprecated Use GDALDatasetGetLayerByName() in GDAL 2.0

Parameters

hDS	handle to the data source from which to get the layer.
pszLayerName	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.

13.13.2.16 int OGR_DS_GetLayerCount (OGRDataSourceH *hDS*)

Get the number of layers in this data source.

Deprecated Use GDALDatasetGetLayerCount() in GDAL 2.0

Parameters

hDS	handle to the data source from which to get the number of layers.

Returns

layer count.

13.13.2.17 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.

Deprecated Use GDALGetDescription() in GDAL 2.0

Parameters

hDS	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.

13.13.2.18 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.

Deprecated Use GDALDatasetReleaseResultSet() in GDAL 2.0

Parameters

hDS	an handle to the data source on which was executed an SQL query.
hLayer	handle to the result of a previous **OGR_DS_ExecuteSQL()** (p. **??**) call.

13.13.2.19 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.
- **ODsCCreateGeomFieldAfterCreateLayer**: True if the layers of this datasource support CreateGeomField() just after layer creation.
- **ODsCCurveGeometries**: True if this datasource supports writing curve geometries. (GDAL 2.0). In that case, OLCCurveGeometries must also be declared in layers of that dataset.

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

Deprecated Use GDALDatasetTestCapability() in GDAL 2.0

Parameters

hDS	handle to the data source against which to test the capability.
pszCapability	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

13.13.2.20 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

hFeat	handle to the feature to clone.

Returns

an handle to the new feature, exactly matching this feature.

13.13.2.21 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

hDefn	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.

13.13.2.22 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

hFeat	handle to the feature to destroy.

13.13.2.23 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

hFeat	handle to the feature to dump.
fpOut	the stream to write to, such as strout.

13.13.2.24 int OGR_F_Equal (OGRFeatureH *hFeat*, OGRFeatureH *hOtherFeat*)

Test if two features are the same.

Two features are considered equal if the 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

hFeat	handle to one of the feature.
hOtherFeat	handle to the other feature to test this one against.

Returns

TRUE if they are equal, otherwise FALSE.

13.13.2.25 void OGR_F_FillUnsetWithDefault (OGRFeatureH *hFeat*, int *bNotNullableOnly*, char ** *papszOptions*)

Fill unset fields with default values that might be defined.

This function is the same as the C++ method **OGRFeature::FillUnsetWithDefault()** (p. **??**).

Parameters

hFeat	handle to the feature.
bNotNullable↩ Only	if we should fill only unset fields with a not-null constraint.
papszOptions	unused currently. Must be set to NULL.

Since

GDAL 2.0

13.13.2.26 OGRFeatureDefnH OGR_F_GetDefnRef (OGRFeatureH *hFeat*)

Fetch feature definition.

This function is the same as the C++ method **OGRFeature::GetDefnRef()** (p. **??**).

Parameters

hFeat	handle to the feature to get the feature definition from.

Returns

an handle to the feature definition object on which feature depends.

13.13.2.27 GIntBig OGR_F_GetFID (OGRFeatureH *hFeat*)

Get feature identifier.

This function is the same as the C++ method **OGRFeature::GetFID()** (p. **??**). Note: since GDAL 2.0, this method returns a GIntBig (previously a long)

Parameters

hFeat	handle to the feature from which to get the feature identifier.

Returns

feature id or OGRNullFID if none has been assigned.

13.13.2.28 GByte* OGR_F_GetFieldAsBinary (OGRFeatureH *hFeat,* int *iField,* int * *pnBytes*)

Fetch field value as binary.

This method only works for OFTBinary and OFTString fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsBinary()** (p. **??**).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnBytes	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.

13.13.2.29 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().

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnYear	(including century)
pnMonth	(1-12)
pnDay	(1-31)
pnHour	(0-23)
pnMinute	(0-59)
pnSecond	(0-59)
pnTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

Returns

TRUE on success or FALSE on failure.

See also

Use **OGR_F_GetFieldAsDateTimeEx()** (p. **??**) for second with millisecond accuracy.

13.13.2.30 int OGR_F_GetFieldAsDateTimeEx (OGRFeatureH *hFeat*, int *iField*, int * *pnYear*, int * *pnMonth*, int * *pnDay*, int * *pnHour*, int * *pnMinute*, float * *pfSecond*, 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().

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnYear	(including century)
pnMonth	(1-12)

pnDay	(1-31)
pnHour	(0-23)
pnMinute	(0-59)
pfSecond	(0-59 with millisecond accuracy)
pnTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

Returns

TRUE on success or FALSE on failure.

Since

GDAL 2.0

13.13.2.31 double OGR_F_GetFieldAsDouble (OGRFeatureH *hFeat,* int *iField*)

Fetch field value as a double.

OFTString features will be translated using **CPLAtof()** (p. **??**). 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

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value.

13.13.2.32 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

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnCount	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.

13.13.2.33 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

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value.

13.13.2.34 GIntBig OGR_F_GetFieldAsInteger64 (OGRFeatureH *hFeat,* int *iField*)

Fetch field value as integer 64 bit.

OFTInteger are promoted to 64 bit. OFTString features will be translated using **CPLAtoGIntBig()** (p. **??**). 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::GetFieldAsInteger64()** (p. **??**).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.

Returns

the field value.

Since

GDAL 2.0

13.13.2.35 const GIntBig* OGR_F_GetFieldAsInteger64List (OGRFeatureH *hFeat,* int *iField,* int * *pnCount*)

Fetch field value as a list of 64 bit integers.

Currently this function only works for OFTInteger64List fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsInteger64List()** (p. **??**).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnCount	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.

Since

GDAL 2.0

13.13.2.36 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

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pnCount	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.

13.13.2.37 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

hFeat	handle to the feature that owned the field.
iField	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.

13.13.2.38 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 **CSL↩ Count()** (p. **??**).

This function is the same as the C++ method **OGRFeature::GetFieldAsStringList()** (p. **??**).

Parameters

hFeat	handle to the feature that owned the field.
iField	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.

13.13.2.39 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

hFeat	handle to the feature to get the fields count from.

Returns

count of fields.

13.13.2.40 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

hFeat	handle to the feature on which the field is found.
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.

References CPLError().

13.13.2.41 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

hFeat	handle to the feature on which the field is found.
pszName	the name of the field to search for.

Returns

the field index, or -1 if no matching field is found.

13.13.2.42 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

hFeat	handle to the feature to get geometry from.

Returns

an handle to internal feature geometry. This object should not be modified.

References OGRGeometryFactory::forceTo(), OGRFeature::GetGeometryRef(), OGRGeometry::getGeometryType(), OGR_GT_GetLinear(), OGR_GT_IsNonLinear(), OGRGetNonLinearGeometriesEnabledFlag(), OGRFeature::SetGeomFieldDirectly(), and OGRFeature::StealGeometry().

13.13.2.43 int OGR_F_GetGeomFieldCount (OGRFeatureH *hFeat*)

Fetch number of geometry fields on this feature This will always be the same as the geometry field count for the **OGRFeatureDefn** (p. **??**).

This function is the same as the C++ method **OGRFeature::GetGeomFieldCount()** (p. **??**).

Parameters

hFeat	handle to the feature to get the geometry fields count from.

Returns

count of geometry fields.

Since

GDAL 1.11

13.13.2.44 OGRGeomFieldDefnH OGR_F_GetGeomFieldDefnRef (OGRFeatureH *hFeat*, int *i*)

Fetch definition for this geometry field.

This function is the same as the C++ method **OGRFeature::GetGeomFieldDefnRef()** (p. **??**).

Parameters

hFeat	handle to the feature on which the field is found.
i	the field to fetch, from 0 to GetGeomFieldCount()-1.

Returns

an handle to the field definition (from the **OGRFeatureDefn** (p. **??**)). This is an internal reference, and should not be deleted or modified.

Since

GDAL 1.11

13.13.2.45 int OGR_F_GetGeomFieldIndex (OGRFeatureH *hFeat*, const char * *pszName*)

Fetch the geometry field index given geometry field name.

This is a cover for the **OGRFeatureDefn::GetGeomFieldIndex()** (p. **??**) method.

This function is the same as the C++ method **OGRFeature::GetGeomFieldIndex()** (p. **??**).

Parameters

hFeat	handle to the feature on which the geometry field is found.
pszName	the name of the geometry field to search for.

Returns

the geometry field index, or -1 if no matching geometry field is found.

Since

GDAL 1.11

13.13.2.46 OGRGeometryH OGR_F_GetGeomFieldRef (OGRFeatureH *hFeat,* int *iField*)

Fetch an handle to feature geometry.

This function is the same as the C++ method **OGRFeature::GetGeomFieldRef()** (p. **??**).

Parameters

hFeat	handle to the feature to get geometry from.
iField	geometry field to get.

Returns

an handle to internal feature geometry. This object should not be modified.

Since

GDAL 1.11

References OGRGeometryFactory::forceTo(), OGRGeometry::getGeometryType(), OGRFeature::GetGeomFieldRef(), OGR_GT_GetLinear(), OGR_GT_IsNonLinear(), OGRGetNonLinearGeometriesEnabledFlag(), OGRFeature::SetGeomFieldDirectly(), and OGRFeature::StealGeometry().

13.13.2.47 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

hFeat	handle to the feature on which field is found.
iField	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.

13.13.2.48 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

hFeat	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.

13.13.2.49 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

hFeat	handle to the feature on which the field is.
iField	the field to test.

Returns

TRUE if the field has been set, otherwise false.

References CPLError(), OGRFeature::GetFieldCount(), and OGRFeature::IsFieldSet().

13.13.2.50 OGRErr OGR_F_SetFID (OGRFeatureH *hFeat,* GIntBig *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

hFeat	handle to the feature to set the feature id to.
nFID	the new feature identifier value to assign.

Returns

On success OGRERR_NONE, or on failure some other value.

13.13.2.51 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

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
nBytes	the number of bytes in pabyData array.
pabyData	the data to apply.

13.13.2.52 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

hFeat	handle to the feature that owned the field.

iField	the field to set, from 0 to GetFieldCount()-1.
nYear	(including century)
nMonth	(1-12)
nDay	(1-31)
nHour	(0-23)
nMinute	(0-59)
nSecond	(0-59)
nTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

See also

Use **OGR_F_SetFieldDateTimeEx()** (p. **??**) for second with millisecond accuracy.

13.13.2.53 void OGR_F_SetFieldDateTimeEx (OGRFeatureH *hFeat,* int *iField,* int *nYear,* int *nMonth,* int *nDay,* int *nHour,* int *nMinute,* float *fSecond,* int *nTZFlag*)

Set field to datetime.

This method currently only has an effect for OFTDate, OFTTime and OFTDateTime fields.

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
nYear	(including century)
nMonth	(1-12)
nDay	(1-31)
nHour	(0-23)
nMinute	(0-59)
fSecond	(0-59, with millisecond accuracy)
nTZFlag	(0=unknown, 1=localtime, 100=GMT, see data model for details)

Since

GDAL 2.0

13.13.2.54 void OGR_F_SetFieldDouble (OGRFeatureH *hFeat,* int *iField,* double *dfValue*)

Set field to double value.

OFTInteger, OFTInteger64 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

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
dfValue	the value to assign.

13.13.2.55 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 OFTIntegerList, OFTInteger64List, OFTRealList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. **??**).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
nCount	the number of values in the list being assigned.
padfValues	the values to assign.

13.13.2.56 void OGR_F_SetFieldInteger (OGRFeatureH *hFeat,* int *iField,* int *nValue*)

Set field to integer value.

OFTInteger, OFTInteger64 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

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
nValue	the value to assign.

13.13.2.57 void OGR_F_SetFieldInteger64 (OGRFeatureH *hFeat,* int *iField,* GIntBig *nValue*)

Set field to 64 bit integer value.

OFTInteger, OFTInteger64 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

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
nValue	the value to assign.

Since

GDAL 2.0

13.13.2.58 void OGR_F_SetFieldInteger64List (OGRFeatureH *hFeat,* int *iField,* int *nCount,* const GIntBig * *panValues*)

Set field to list of 64 bit integers value.

This function currently on has an effect of OFTIntegerList, OFTInteger64List and OFTRealList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. **??**).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.

nCount	the number of values in the list being assigned.
panValues	the values to assign.

Since

GDAL 2.0

13.13.2.59 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, OFTInteger64List and OFTRealList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. **??**).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
nCount	the number of values in the list being assigned.
panValues	the values to assign.

13.13.2.60 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

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
psValue	handle on the value to assign.

13.13.2.61 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. OFTInteger64 fields will be set based on an **CPLAtoGIntBig()** (p. **??**) conversion of the string. OFTReal fields will be set based on an **CPLAtof()** (p. **??**) conversion of the string. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. **??**).

Parameters

hFeat	handle to the feature that owned the field.
iField	the field to fetch, from 0 to GetFieldCount()-1.
pszValue	the value to assign.

13.13.2.62 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

hFeat	handle to the feature that owned the field.
iField	the field to set, from 0 to GetFieldCount()-1.
papszValues	the values to assign.

13.13.2.63 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

hFeat	handle to the feature to set to.
hOtherFeat	handle to the feature from which geometry, and field values will be copied.
bForgiving	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.

13.13.2.64 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

hFeat	handle to the feature to set to.
hOtherFeat	handle to the feature from which geometry, and field values will be copied.
panMap	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.
bForgiving	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.

13.13.2.65 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

hFeat	handle to the feature on which new geometry is applied to.
hGeom	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).

13.13.2.66 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 (even in case of failure of that function).

This function is the same as the C++ method **OGRFeature::SetGeometryDirectly** (p. **??**).

Parameters

hFeat	handle to the feature on which to apply the geometry.
hGeom	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).

13.13.2.67 OGRErr OGR_F_SetGeomField (OGRFeatureH *hFeat,* int *iField,* OGRGeometryH *hGeom*)

Set feature geometry of a specified geometry field.

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::SetGeomField()** (p. **??**).

Parameters

hFeat	handle to the feature on which new geometry is applied to.
iField	geometry field to set.
hGeom	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).

13.13.2.68 OGRErr OGR_F_SetGeomFieldDirectly (OGRFeatureH *hFeat,* int *iField,* OGRGeometryH *hGeom*)

Set feature geometry of a specified geometry field.

This function updates the features geometry, and operate exactly as SetGeomField(), except that this function assumes ownership of the passed geometry (even in case of failure of that function).

This function is the same as the C++ method **OGRFeature::SetGeomFieldDirectly** (p. **??**).

Parameters

hFeat	handle to the feature on which to apply the geometry.
iField	geometry field to set.
hGeom	handle to the new geometry to apply to feature.

Returns

OGRERR_NONE if successful, or OGRERR_FAILURE if the index is invalid, or OGR_UNSUPPORTED_↩ GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. **??**) (checking not yet implemented).

Since

GDAL 1.11

13.13.2.69 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

hFeat	handle to the feature to set style to.
pszStyle	the style string to apply to this feature, cannot be NULL.

13.13.2.70 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

hFeat	handle to the feature to set style to.
pszStyle	the style string to apply to this feature, cannot be NULL.

13.13.2.71 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.

13.13.2.72 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

hFeat	handle to the feature on which the field is.
iField	the field to unset.

13.13.2.73 int OGR_F_Validate (OGRFeatureH *hFeat,* int *nValidateFlags,* int *bEmitError*)

Validate that a feature meets constraints of its schema.

The scope of test is specified with the nValidateFlags parameter.

Regarding OGR_F_VAL_WIDTH, the test is done assuming the string width must be interpreted as the number of UTF-8 characters. Some drivers might interpret the width as the number of bytes instead. So this test is rather conservative (if it fails, then it will fail for all interpretations).

This function is the same as the C++ method **OGRFeature::Validate()** (p. **??**).

Parameters

hFeat	handle to the feature to validate.
nValidateFlags	OGR_F_VAL_ALL or combination of OGR_F_VAL_NULL, OGR_F_VAL_GEOM_TYPE, O↩ GR_F_VAL_WIDTH and OGR_F_VAL_ALLOW_NULL_WHEN_DEFAULT with '|' operator
bEmitError	TRUE if a **CPLError()** (p. **??**) must be emitted when a check fails

Returns

TRUE if all enabled validation tests pass.

Since

GDAL 2.0

13.13.2.74 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 existance based on this **O↩ GRFeatureDefn** (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

hDefn	handle to the feature definition to add the field definition to.
hNewField	handle to the new field definition.

13.13.2.75 void OGR_FD_AddGeomFieldDefn (OGRFeatureDefnH *hDefn,* OGRGeomFieldDefnH *hNewGeomField*)

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_CreateGeom↩ Field()** (p. **??**) instead.

This function should only be called while there are no **OGRFeature** (p. **??**) objects in existance based on this **OG↩ RFeatureDefn** (p. **??**). The **OGRGeomFieldDefn** (p. **??**) passed in is copied, and remains the responsibility of the caller.

This function is the same as the C++ method **OGRFeatureDefn::AddGeomFieldDefn()** (p. **??**).

Parameters

hDefn	handle to the feature definition to add the geometry field definition to.
hNewGeomField	handle to the new field definition.

Since

GDAL 1.11

13.13.2.76 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

pszName	the name to be assigned to this layer/class. It does not need to be unique.

Returns

handle to the newly created feature definition.

13.13.2.77 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 existance based on this **OGRFeatureDefn** (p. **??**).

This method is the same as the C++ method **OGRFeatureDefn::DeleteFieldDefn()** (p. **??**).

Parameters

hDefn	handle to the feature definition.
iField	the index of the field defintion.

Returns

OGRERR_NONE in case of success.

Since

OGR 1.9.0

13.13.2.78 OGRErr OGR_FD_DeleteGeomFieldDefn (OGRFeatureDefnH *hDefn,* int *iGeomField*)

Delete an existing geometry field definition.

To delete an existing geometry field definition from a layer definition, do not use this function directly, but use OGR_L_DeleteGeomField() instead (*not implemented yet*)

This method should only be called while there are no **OGRFeature** (p. **??**) objects in existance based on this **OGRFeatureDefn** (p. **??**).

This method is the same as the C++ method **OGRFeatureDefn::DeleteGeomFieldDefn()** (p. **??**).

Parameters

hDefn	handle to the feature definition.
iGeomField	the index of the geometry field defintion.

Returns

OGRERR_NONE in case of success.

Since

GDAL 1.11

13.13.2.79 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

hDefn	handle to the feature definition on witch **OGRFeature** (p. **??**) are based on.

Returns

the updated reference count.

13.13.2.80 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

hDefn	handle to the feature definition to be destroyed.

13.13.2.81 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

hDefn	handle to the feature definition to get the fields count from.

Returns

count of fields.

13.13.2.82 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

hDefn	handle to the feature definition to get the field definition from.
iField	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.

13.13.2.83 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

hDefn	handle to the feature definition to get field index from.
pszFieldName	the field name to search for.

Returns

the field index, or -1 if no match found.

13.13.2.84 int OGR_FD_GetGeomFieldCount (OGRFeatureDefnH *hDefn*)

Fetch number of geometry fields on the passed feature definition.

This function is the same as the C++ **OGRFeatureDefn::GetGeomFieldCount()** (p. **??**).

Parameters

hDefn	handle to the feature definition to get the fields count from.

Returns

count of geometry fields.

Since

GDAL 1.11

13.13.2.85 OGRGeomFieldDefnH OGR_FD_GetGeomFieldDefn (OGRFeatureDefnH *hDefn,* int *iGeomField*)

Fetch geometry field definition of the passed feature definition.

This function is the same as the C++ method **OGRFeatureDefn::GetGeomFieldDefn()** (p. **??**).

Parameters

hDefn	handle to the feature definition to get the field definition from.

iGeomField	the geometry field to fetch, between 0 and GetGeomFieldCount()-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.

Since

GDAL 1.11

13.13.2.86 int OGR_FD_GetGeomFieldIndex (OGRFeatureDefnH *hDefn,* const char * *pszGeomFieldName*)

Find geometry field by name.

The geometry field index of the first geometry field matching the passed field name (case insensitively) is returned.

This function is the same as the C++ method **OGRFeatureDefn::GetGeomFieldIndex** (p. **??**).

Parameters

hDefn	handle to the feature definition to get field index from.
pszGeomFieldName	the geometry field name to search for.

Returns

the geometry field index, or -1 if no match found.

13.13.2.87 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. **??**).

Starting with GDAL 1.11, this method returns GetGeomFieldDefn(0)->GetType().

Parameters

hDefn	handle to the feature definition to get the geometry type from.

Returns

the base type for all geometry related to this definition.

References OGR_GT_GetLinear(), OGR_GT_IsNonLinear(), and OGRGetNonLinearGeometriesEnabledFlag().

13.13.2.88 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

hDefn	handle to the feature definition to get the name from.

Returns

the name. This name is internal and should not be modified, or freed.

13.13.2.89 int OGR_FD_GetReferenceCount (OGRFeatureDefnH *hDefn*)

Fetch current reference count.

This function is the same as the C++ method **OGRFeatureDefn::GetReferenceCount()** (p. **??**).

Parameters

hDefn	handle to the feature definition on witch **OGRFeature** (p. **??**) are based on.

Returns

the current reference count.

13.13.2.90 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. **??**).

Starting with GDAL 1.11, this method returns GetGeomFieldDefn(0)->IsIgnored().

Parameters

hDefn	handle to the feature definition on witch **OGRFeature** (p. **??**) are based on.

Returns

ignore state

13.13.2.91 int OGR_FD_IsSame (OGRFeatureDefnH *hFDefn,* OGRFeatureDefnH *hOtherFDefn*)

Test if the feature definition is identical to the other one.

Parameters

hFDefn	handle to the feature definition on witch **OGRFeature** (p. **??**) are based on.
hOtherFDefn	handle to the other feature definition to compare to.

Returns

TRUE if the feature definition is identical to the other one.

Since

OGR 1.11

13.13.2.92 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

hDefn	handle to the feature definition on which **OGRFeature** (p. **??**) are based on.

Returns

ignore state

13.13.2.93 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

hDefn	handle to the feature definition on witch **OGRFeature** (p. **??**) are based on.

Returns

the updated reference count.

13.13.2.94 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

hDefn	handle to the feature definition to be released.

13.13.2.95 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. **??**).

Starting with GDAL 1.11, this method calls GetGeomFieldDefn(0)->SetIgnored().

Parameters

hDefn	handle to the feature definition on witch **OGRFeature** (p. **??**) are based on.
bIgnore	ignore state

13.13.2.96 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. **??**).

Starting with GDAL 1.11, this method calls GetGeomFieldDefn(0)->SetType().

Parameters

hDefn	handle to the layer or feature definition to set the geometry type to.
eType	the new type to assign.

13.13.2.97 void OGR_FD_SetStyleIgnored (OGRFeatureDefnH *hDefn,* int *bIgnore*)

Set whether the style can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::SetStyleIgnored()** (p. **??**).

Parameters

hDefn	handle to the feature definition on witch **OGRFeature** (p. **??**) are based on.
bIgnore	ignore state

13.13.2.98 OGRFieldDefnH OGR_Fld_Create (const char * *pszName,* OGRFieldType *eType*)

Create a new field definition.

By default, fields have no width, precision, are nullable and not ignored.

This function is the same as the CPP method **OGRFieldDefn::OGRFieldDefn()** (p. **??**).

Parameters

pszName	the name of the new field definition.
eType	the type of the new field definition.

Returns

handle to the new field definition.

13.13.2.99 void OGR_Fld_Destroy (OGRFieldDefnH *hDefn*)

Destroy a field definition.

Parameters

hDefn	handle to the field definition to destroy.

13.13.2.100 const char* OGR_Fld_GetDefault (OGRFieldDefnH *hDefn*)

Get default field value.

This function is the same as the C++ method **OGRFieldDefn::GetDefault()** (p. **??**).

Parameters

hDefn	handle to the field definition.

Returns

default field value or NULL.

Since

GDAL 2.0

13.13.2.101 OGRJustification OGR_Fld_GetJustify (OGRFieldDefnH *hDefn*)

Get the justification for this field.

This function is the same as the CPP method **OGRFieldDefn::GetJustify()** (p. **??**).

Note: no driver is know to use the concept of field justification.

Parameters

hDefn	handle to the field definition to get justification from.

Returns

the justification.

13.13.2.102 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

hDefn	handle to the field definition.

Returns

the name of the field definition.

13.13.2.103 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

hDefn	handle to the field definition to get precision from.

Returns

the precision.

13.13.2.104 OGRFieldSubType OGR_Fld_GetSubType (OGRFieldDefnH *hDefn*)

Fetch subtype of this field.

This function is the same as the CPP method **OGRFieldDefn::GetSubType()** (p. **??**).

Parameters

hDefn	handle to the field definition to get subtype from.

Returns

field subtype.

Since

GDAL 2.0

13.13.2.105 OGRFieldType OGR_Fld_GetType (OGRFieldDefnH *hDefn*)

Fetch type of this field.

This function is the same as the CPP method **OGRFieldDefn::GetType()** (p. **??**).

Parameters

hDefn	handle to the field definition to get type from.

Returns

field type.

13.13.2.106 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

hDefn	handle to the field definition to get width from.

Returns

the width, zero means no specified width.

13.13.2.107 int OGR_Fld_IsDefaultDriverSpecific (OGRFieldDefnH *hDefn*)

Returns whether the default value is driver specific.

Driver specific default values are those that are *not* NULL, a numeric value, a litteral value enclosed between single quote characters, CURRENT_TIMESTAMP, CURRENT_TIME, CURRENT_DATE or datetime literal value.

This function is the same as the C++ method **OGRFieldDefn::IsDefaultDriverSpecific()** (p. **??**).

Parameters

hDefn	handle to the field definition

Returns

TRUE if the default value is driver specific.

Since

GDAL 2.0

13.13.2.108 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

hDefn	handle to the field definition

Returns

ignore state

13.13.2.109 int OGR_Fld_IsNullable (OGRFieldDefnH *hDefn*)

Return whether this field can receive null values.

By default, fields are nullable.

Even if this method returns FALSE (i.e not-nullable field), it doesn't mean that **OGRFeature::IsFieldSet()** (p. **??**) will necessary return TRUE, as fields can be temporary unset and null/not-null validation is usually done when **OGRLayer::CreateFeature()** (p. **??**)/SetFeature() is called.

This method is the same as the C++ method **OGRFieldDefn::IsNullable()** (p. **??**).

Parameters

hDefn	handle to the field definition

Returns

TRUE if the field is authorized to be null.

Since

GDAL 2.0

13.13.2.110 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

hDefn	handle to the field definition to set to.
pszNameIn	the new name to assign.
eTypeIn	the new type (one of the OFT values like OFTInteger).
nWidthIn	the preferred formatting width. Defaults to zero indicating undefined.
nPrecisionIn	number of decimals places for formatting, defaults to zero indicating undefined.
eJustifyIn	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

13.13.2.111 void OGR_Fld_SetDefault (OGRFieldDefnH *hDefn*, const char * *pszDefault*)

Set default field value.

The default field value is taken into account by drivers (generally those with a SQL interface) that support it at field creation time. OGR will generally not automatically set the default field value to null fields by itself when calling **OGRFeature::CreateFeature()** (p. **??**) / OGRFeature::SetFeature(), but will let the low-level layers to do the job. So retrieving the feature from the layer is recommended.

The accepted values are NULL, a numeric value, a litteral value enclosed between single quote characters (and inner single quote characters escaped by repetition of the single quote character), CURRENT_TIMESTAMP, C↩ URRENT_TIME, CURRENT_DATE or a driver specific expression (that might be ignored by other drivers). For a datetime literal value, format should be 'YYYY/MM/DD HH:MM:SS[.sss]' (considered as UTC time).

Drivers that support writing DEFAULT clauses will advertize the GDAL_DCAP_DEFAULT_FIELDS driver metadata item.

This function is the same as the C++ method **OGRFieldDefn::SetDefault()** (p. **??**).

Parameters

hDefn	handle to the field definition.
pszDefault	new default field value or NULL pointer.

Since

GDAL 2.0

13.13.2.112 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++ method **OGRFieldDefn::SetIgnored()** (p. **??**).

Parameters

hDefn	handle to the field definition
ignore	ignore state

13.13.2.113 void OGR_Fld_SetJustify (OGRFieldDefnH *hDefn,* OGRJustification *eJustify*)

Set the justification for this field.

Note: no driver is know to use the concept of field justification.

This function is the same as the CPP method **OGRFieldDefn::SetJustify()** (p. **??**).

Parameters

hDefn	handle to the field definition to set justification to.
eJustify	the new justification.

13.13.2.114 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

hDefn	handle to the field definition to apply the new name to.
pszName	the new name to apply.

13.13.2.115 void OGR_Fld_SetNullable (OGRFieldDefnH *hDefn,* int *bNullableIn*)

Set whether this field can receive null values.

By default, fields are nullable, so this method is generally called with FALSE to set a not-null constraint.

Drivers that support writing not-null constraint will advertize the GDAL_DCAP_NOTNULL_FIELDS driver metadata item.

This method is the same as the C++ method **OGRFieldDefn::SetNullable()** (p. **??**).

Parameters

hDefn	handle to the field definition
bNullableIn	FALSE if the field must have a not-null constraint.

Since

GDAL 2.0

13.13.2.116 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

hDefn	handle to the field definition to set precision to.
nPrecision	the new precision.

13.13.2.117 void OGR_Fld_SetSubType (OGRFieldDefnH *hDefn,* OGRFieldSubType *eSubType*)

Set the subtype of this field. This should never be done to an **OGRFieldDefn** (p. **??**) that is already part of an **OGRFeatureDefn** (p. **??**).

This function is the same as the CPP method **OGRFieldDefn::SetSubType()** (p. **??**).

Parameters

hDefn	handle to the field definition to set type to.
eSubType	the new field subtype.

Since

GDAL 2.0

13.13.2.118 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

hDefn	handle to the field definition to set type to.
eType	the new field type.

13.13.2.119 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

hDefn	handle to the field definition to set width to.
nNewWidth	the new width.

13.13.2.120 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

hGeom	existing geometry container.
hNewSubGeom	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 OGR_GT_IsCurve(), OGR_GT_IsSubClassOf(), wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, and wkbGeometryCollection.

13.13.2.121 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

hGeom	existing geometry.
hNewSubGeom	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 OGR_GT_IsCurve(), OGR_GT_IsSubClassOf(), wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, and wkbGeometryCollection.

13.13.2.122 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

hGeom	handle to the geometry to add a point to.
dfX	x coordinate of point to add.
dfY	y coordinate of point to add.
dfZ	z coordinate of point to add.

References CPLError(), wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

13.13.2.123 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

hGeom	handle to the geometry to add a point to.
dfX	x coordinate of point to add.
dfY	y coordinate of point to add.

References CPLError(), wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

13.13.2.124 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

dfCenterX	center X
dfCenterY	center Y
dfZ	center Z
dfPrimaryRadius	X radius of ellipse.
dfSecondary↩ Radius	Y radius of ellipse.
dfRotation	rotation of the ellipse clockwise.
dfStartAngle	angle to first point on arc (clockwise of X-positive)
dfEndAngle	angle to last point on arc (clockwise of X-positive)
dfMaxAngle↩ StepSize↩ Degrees	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().

13.13.2.125 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

hGeom	the geometry to operate on.

Returns

the area or 0.0 for unsupported geometry types.

Since

OGR 1.8.0

References CPLError(), OGR_GT_IsCurve(), OGR_GT_IsSubClassOf(), OGR_GT_IsSurface(), wkbFlatten, wkbGeometryCollection, and wkbMultiSurface.

Referenced by OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsInteger64(), OGRFeature::GetFieldAsString(), OGRFeature::IsFieldSet(), and OGR_G_GetArea().

13.13.2.126 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 reprojected. 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

hGeom	handle on the geometry to apply the new spatial reference system.
hSRS	handle on the new spatial reference system to apply.

13.13.2.127 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 CPLE_NotSupported error.

Parameters

hTarget	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

13.13.2.128 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 nQuad↩ Segs parameter can be used to control how many segements 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

hTarget	the geometry.
dfDist	the buffer distance to be applied. Should be expressed into the same unit as the coordinates of the geometry.
nQuadSegs	the number of segments used to approximate a 90 degree (quadrant) of curvature.

Returns

the newly created geometry, or NULL if an error occurs.

13.13.2.129 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(), wkbFlatten, and wkbPoint.

13.13.2.130 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

hGeom	handle on the geometry to clone from.

Returns

an handle on the copy of the geometry with the spatial reference system as the original.

13.13.2.131 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

hGeom	handle to the geometry.

13.13.2.132 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

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if hThis contains hOther geometry, otherwise FALSE.

13.13.2.133 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

hTarget	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.

13.13.2.134 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

pszGML	The GML fragment for the geometry.

Returns

a geometry on succes, or NULL on error.

References CPLDestroyXMLNode(), CPLError(), CPLGetConfigOption(), CPLParseXMLString(), and CSLTest↩ Boolean().

Referenced by OGRGeometryFactory::createFromGML().

13.13.2.135 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

pabyData	pointer to the input BLOB data.
hSRS	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.

phGeometry	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 **OGR_G_↩ DestroyGeometry()** (p. **??**) after use.
nBytes	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGRGeometryFactory::createFromWkb().

13.13.2.136 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

ppszData	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.
hSRS	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.
phGeometry	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 **OGR_G_↩ DestroyGeometry()** (p. **??**) after use.

Returns

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGRGeometryFactory::createFromWkt().

13.13.2.137 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

eGeometryType	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().

13.13.2.138 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

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if they are crossing, otherwise FALSE.

13.13.2.139 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

hGeom	handle to the geometry to delete.

References OGRGeometryFactory::destroyGeometry().

13.13.2.140 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

hThis	the geometry.
hOther	the other geometry.

Returns

a new geometry representing the difference or NULL if the difference is empty or an error occurs.

13.13.2.141 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

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if they are disjoint, otherwise FALSE.

13.13.2.142 double OGR_G_Distance (OGRGeometryH *hFirst,* OGRGeometryH *hOther*)

Compute distance between two geometries.

Returns the shortest distance between the two geometries. The distance is expressed into the same unit as the coordinates of the 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

hFirst	the first geometry to compare against.
hOther	the other geometry to compare against.

Returns

the distance between the geometries or -1 if an error occurs.

13.13.2.143 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

hGeom	handle on the geometry to dump.
fp	the text file to write the geometry to.
pszPrefix	the prefix to put on each line of output.

13.13.2.144 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

hGeom	handle on the geometry to empty.

13.13.2.145 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

hGeom	handle on the first geometry.
hOther	handle on the other geometry to test against.

Returns

TRUE if equivalent or FALSE otherwise.

References CPLError().

13.13.2.146 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

hGeometry	handle to the geometry.

Returns

A GML fragment or NULL in case of error.

References OGR_G_ExportToGMLEx().

13.13.2.147 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 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.
- SRSDIMENSION_LOC=POSLIST/GEOMETRY/GEOMETRY,POSLIST. (Only valid for FORMAT=GML3, GDAL >= 2.0) Default to POSLIST. For 2.5D geometries, define the location where to attach the srs↩ Dimension attribute. There are diverging implementations. Some put in on the <gml:posList> element, other on the top geometry element.

Note that curve geometries like CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE are not supported in GML 2.

This method is the same as the C++ method **OGRGeometry::exportToGML()** (p. **??**).

Parameters

hGeometry	handle to the geometry.
papszOptions	NULL-terminated list of options.

Returns

A GML fragment or NULL in case of error.

Since

OGR 1.8.0

References CPLDebug(), CPLMalloc(), CPLStrdup(), CSLDestroy(), CSLTestBoolean(), and CSLTokenizeString2().

Referenced by OGRGeometry::exportToGML(), and OGR_G_ExportToGML().

13.13.2.148 OGRErr OGR_G_ExportToIsoWkb (OGRGeometryH *hGeom*, OGRwkbByteOrder *eOrder*, unsigned char * *pabyDstBuffer*)

Convert a geometry into SFSQL 1.2 / ISO SQL/MM Part 3 well known binary format.

This function relates to the SFCOM IWks::ExportToWKB() method. It exports the SFSQL 1.2 and ISO SQL/MM Part 3 extended dimension (Z&M) WKB types

This function is the same as the CPP method OGRGeometry::exportToWkb(OGRwkbByteOrder, unsigned char *, OGRwkbVariant) with eWkbVariant = wkbVariantIso.

Parameters

hGeom	handle on the geometry to convert to a well know binary data from.
eOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyDstBuffer	a buffer into which the binary representation is written. This buffer must be at least **OGR_G_WkbSize()** (p. **??**) byte in size.

Returns

Currently OGRERR_NONE is always returned.

Since

GDAL 2.0

References wkbVariantIso.

13.13.2.149 OGRErr OGR_G_ExportToIsoWkt (OGRGeometryH *hGeom*, char ** *ppszSrcText*)

Convert a geometry into SFSQL 1.2 / ISO SQL/MM Part 3 well known text format.

This function relates to the SFCOM IWks::ExportToWKT() method. It exports the SFSQL 1.2 and ISO SQL/MM Part 3 extended dimension (Z&M) WKB types

This function is the same as the CPP method OGRGeometry::exportToWkt(,wkbVariantIso).

Parameters

hGeom	handle on the geometry to convert to a text format from.
ppszSrcText	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.

Since

GDAL 2.0

References wkbVariantIso.

13.13.2.150 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

hGeometry	handle to the geometry.

Returns

A GeoJSON fragment or NULL in case of error.

Referenced by OGRGeometry::exportToJson().

13.13.2.151 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

hGeometry	handle to the geometry.
papszOptions	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

13.13.2.152 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

hGeometry	handle to the geometry.
pszAltitudeMode	value to write in altitudeMode element, or NULL.

Returns

A KML fragment or NULL in case of error.

Referenced by OGRGeometry::exportToKML().

13.13.2.153 OGRErr OGR_G_ExportToWkb (OGRGeometryH *hGeom,* OGRwkbByteOrder *eOrder,* unsigned char * *pabyDstBuffer*)

Convert a geometry well known binary format.

This function relates to the SFCOM IWks::ExportToWKB() method.

For backward compatibility purposes, it exports the Old-style 99-402 extended dimension (Z) WKB types for types Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon and GeometryCollection. For other geometry types, it is equivalent to **OGR_G_ExportToIsoWkb()** (p. **??**).

This function is the same as the CPP method OGRGeometry::exportToWkb(OGRwkbByteOrder, unsigned char *, OGRwkbVariant) with eWkbVariant = wkbVariantOldOgc.

Parameters

hGeom	handle on the geometry to convert to a well know binary data from.
eOrder	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
pabyDstBuffer	a buffer into which the binary representation is written. This buffer must be at least **OGR_G_WkbSize()** (p. **??**) byte in size.

Returns

Currently OGRERR_NONE is always returned.

13.13.2.154 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.

For backward compatibility purposes, it exports the Old-style 99-402 extended dimension (Z) WKB types for types Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon and GeometryCollection. For other geometry types, it is equivalent to **OGR_G_ExportToIsoWkt()** (p. **??**).

This function is the same as the CPP method **OGRGeometry::exportToWkt()** (p. **??**).

Parameters

hGeom	handle on the geometry to convert to a text format from.
ppszSrcText	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.

13.13.2.155 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

hGeom	handle on the geometry to convert.

13.13.2.156 OGRGeometryH OGR_G_ForceTo (OGRGeometryH *hGeom,* OGRwkbGeometryType *eTargetType,* char ** *papszOptions*)

Convert to another geometry type.

This function is the same as the C++ method **OGRGeometryFactory::forceTo()** (p. **??**).

Parameters

hGeom	the input geometry - ownership is passed to the method.
eTargetType	target output geometry type.
papszOptions	options as a null-terminated list of strings or NULL.

Returns

new geometry.

Since

GDAL 2.0

References OGRGeometryFactory::forceTo().

13.13.2.157 OGRGeometryH OGR_G_ForceToLineString (OGRGeometryH *hGeom*)

Convert to line string.

This function is the same as the C++ method **OGRGeometryFactory::forceToLineString()** (p. **??**).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.10.0

References OGRGeometryFactory::forceToLineString().

13.13.2.158 OGRGeometryH OGR_G_ForceToMultiLineString (OGRGeometryH *hGeom*)

Convert to multilinestring.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiLineString()** (p. **??**).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiLineString().

13.13.2.159 OGRGeometryH OGR_G_ForceToMultiPoint (OGRGeometryH *hGeom*)

Convert to multipoint.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiPoint()** (p. **??**).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiPoint().

13.13.2.160 OGRGeometryH OGR_G_ForceToMultiPolygon (OGRGeometryH *hGeom*)

Convert to multipolygon.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiPolygon()** (p. **??**).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiPolygon().

13.13.2.161 OGRGeometryH OGR_G_ForceToPolygon (OGRGeometryH *hGeom*)

Convert to polygon.

This function is the same as the C++ method **OGRGeometryFactory::forceToPolygon()** (p. **??**).

Parameters

hGeom	handle to the geometry to convert (ownership surrendered).

Returns

the converted geometry (ownership to caller).

Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToPolygon().

13.13.2.162 double OGR_G_GetArea (OGRGeometryH *hGeom*)

Compute geometry area (deprecated)

See also

Deprecated **OGR_G_Area()** (p. **??**)

References OGR_G_Area().

13.13.2.163 OGRGeometryH OGR_G_GetBoundary (OGRGeometryH *hTarget*)

Compute boundary (deprecated)

Deprecated

See also

OGR_G_Boundary() (p. **??**)

13.13.2.164 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

hGeom	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.

13.13.2.165 OGRGeometryH OGR_G_GetCurveGeometry (OGRGeometryH *hGeom,* char ** *papszOptions*)

Return curve version of this geometry.

Returns a geometry that has possibly CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by de-approximating linear into curve geometries.

If the geometry has no curve portion, the returned geometry will be a clone of it.

The ownership of the returned geometry belongs to the caller.

The reverse function is **OGR_G_GetLinearGeometry()** (p. **??**).

This function is the same as C++ method **OGRGeometry::getCurveGeometry()** (p. **??**).

Parameters

hGeom	the geometry to operate on.
papszOptions	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

Returns

a new geometry.

Since

GDAL 2.0

13.13.2.166 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_GetCoordinate↩ Dimension()** (p. **??**) function).

This function is the same as the CPP method **OGRGeometry::getDimension()** (p. **??**).

Parameters

hGeom	handle on the geometry to get the dimension from.

Returns

0 for points, 1 for lines and 2 for surfaces.

13.13.2.167 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

hGeom	handle of the geometry to get envelope from.
psEnvelope	the structure in which to place the results.

13.13.2.168 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

hGeom	handle of the geometry to get envelope from.
psEnvelope	the structure in which to place the results.

Since

OGR 1.9.0

13.13.2.169 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

hGeom	single geometry or geometry container from which to get the number of elements.

Returns

the number of elements.

References OGR_GT_IsSubClassOf(), wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, and wkbGeometry↩ Collection.

13.13.2.170 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

hGeom	handle on the geometry to get name from.

Returns

name used for this geometry type in well known text format.

13.13.2.171 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

hGeom	handle to the geometry container from which to get a geometry from.
iSubGeom	the index of the geometry to fetch, between 0 and getNumGeometries() - 1.

Returns

handle to the requested geometry.

References CPLError(), OGR_GT_IsSubClassOf(), wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, and wkbGeometryCollection.

13.13.2.172 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()** (p. **??**) macro to the return result.

This function is the same as the CPP method **OGRGeometry::getGeometryType()** (p. **??**).

Parameters

hGeom	handle on the geometry to get type from.

Returns

the geometry type code.

References wkbUnknown.

13.13.2.173 OGRGeometryH OGR_G_GetLinearGeometry (OGRGeometryH *hGeom,* double *dfMaxAngleStepSizeDegrees,* char ** *papszOptions*)

Return, possibly approximate, linear version of this geometry.

Returns a geometry that has no CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it, by approximating curve geometries.

The ownership of the returned geometry belongs to the caller.

The reverse function is **OGR_G_GetCurveGeometry()** (p. **??**).

This method relates to the ISO SQL/MM Part 3 ICurve::CurveToLine() and CurvePolygon::CurvePolyToPoly() methods.

This function is the same as C++ method **OGRGeometry::getLinearGeometry()** (p. **??**).

Parameters

hGeom	the geometry to operate on.
dfMaxAngle↩ StepSize↩ Degrees	the largest step in degrees along the arc, zero to use the default setting.
papszOptions	options as a null-terminated list of strings or NULL. See **OGRGeometryFactory::curveTo↩ LineString()** (p. **??**) for valid options.

Returns

a new geometry.

Since

GDAL 2.0

13.13.2.174 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

hGeom	handle to the geometry from which to get the coordinates.
i	the vertex to fetch, from 0 to getNumPoints()-1, zero for a point.
pdfX	value of x coordinate.
pdfY	value of y coordinate.
pdfZ	value of z coordinate.

References CPLError(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getX(), OGRSimpleCurve::getY(), OGRSimpleCurve::getZ(), wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

13.13.2.175 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

hGeom	handle to the geometry from which to get the number of points.

Returns

the number of points.

References OGR_GT_IsCurve(), wkbFlatten, and wkbPoint.

13.13.2.176 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

hGeom	handle to the geometry from which to get the coordinates.
pabyX	a buffer of at least (sizeof(double) * nXStride * nPointCount) bytes, may be NULL.
nXStride	the number of bytes between 2 elements of pabyX.
pabyY	a buffer of at least (sizeof(double) * nYStride * nPointCount) bytes, may be NULL.
nYStride	the number of bytes between 2 elements of pabyY.
pabyZ	a buffer of at last size (sizeof(double) * nZStride * nPointCount) bytes, may be NULL.
nZStride	the number of bytes between 2 elements of pabyZ.

Returns

the number of points

Since

OGR 1.9.0

References CPLError(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getPoints(), wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

13.13.2.177 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

hGeom	handle on the geometry to get spatial reference from.

Returns

a reference to the spatial reference geometry.

13.13.2.178 double OGR_G_GetX (OGRGeometryH *hGeom,* int *i*)

Fetch the x coordinate of a point from a geometry.

Parameters

hGeom	handle to the geometry from which to get the x coordinate.
i	point to get the x coordinate.

Returns

the X coordinate of this point.

References CPLError(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getX(), wkbCircularString, wkb↩Flatten, wkbLineString, and wkbPoint.

13.13.2.179 double OGR_G_GetY (OGRGeometryH *hGeom,* int *i*)

Fetch the x coordinate of a point from a geometry.

Parameters

hGeom	handle to the geometry from which to get the y coordinate.
i	point to get the Y coordinate.

Returns

the Y coordinate of this point.

References CPLError(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getY(), wkbCircularString, wkb↩Flatten, wkbLineString, and wkbPoint.

13.13.2.180 double OGR_G_GetZ (OGRGeometryH *hGeom,* int *i*)

Fetch the z coordinate of a point from a geometry.

Parameters

hGeom	handle to the geometry from which to get the Z coordinate.

i	point to get the Z coordinate.

Returns

the Z coordinate of this point.

References CPLError(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getZ(), wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

13.13.2.181 int OGR_G_HasCurveGeometry (OGRGeometryH *hGeom,* int *bLookForNonLinear*)

Returns if this geometry is or has curve geometry.

Returns if a geometry is or has CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE or MULTISURFACE in it.

If bLookForNonLinear is set to TRUE, it will be actually looked if the geometry or its subgeometries are or contain a non-linear geometry in them. In which case, if the method returns TRUE, it means that **OGR_G_GetLinearGeometry()** (p. **??**) would return an approximate version of the geometry. Otherwise, **OGR_G_GetLinearGeometry()** (p. **??**) would do a conversion, but with just converting container type, like COMPOUNDCURVE -> LINESTRING, MULTICURVE -> MULTILINESTRING or MULTISURFACE -> MULTIPOLYGON, resulting in a "loss-less" conversion.

This function is the same as C++ method **OGRGeometry::hasCurveGeometry()** (p. **??**).

Parameters

hGeom	the geometry to operate on.
bLookForNonLinear	set it to TRUE to check if the geometry is or contains a CIRCULARSTRING.

Returns

TRUE if this geometry is or has curve geometry.

Since

GDAL 2.0

13.13.2.182 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

hGeom	handle on the geometry to assign the well know binary data to.
pabyData	the binary input data.
nSize	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

13.13.2.183 OGRErr OGR_G_ImportFromWkt (OGRGeometryH *hGeom,* char ** *ppszSrcText*)

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

hGeom	handle on the geometry to assign well know text data to.
ppszSrcText	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_UNSUP↩ PORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

13.13.2.184 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_↩ Intersects()** (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

hThis	the geometry.
hOther	the other geometry.

Returns

a new geometry representing the intersection or NULL if there is no intersection or an error occurs.

13.13.2.185 int OGR_G_Intersects (OGRGeometryH *hGeom,* OGRGeometryH *hOtherGeom*)

Do these features intersect?

Currently this is not implemented in a rigerous fashion, and generally just tests whether the envelopes of the two features intersect. Eventually this will be made rigerous.

This function is the same as the CPP method **OGRGeometry::Intersects** (p. **??**).

Parameters

hGeom	handle on the first geometry.
hOtherGeom	handle on the other geometry to test against.

Returns

TRUE if the geometries intersect, otherwise FALSE.

13.13.2.186 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

hGeom	The Geometry to test.

Returns

TRUE if the geometry has no points, otherwise FALSE.

13.13.2.187 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

hGeom	The Geometry to test.

Returns

TRUE if the geometry has no points, otherwise FALSE.

13.13.2.188 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

hGeom	The Geometry to test.

Returns

TRUE if object is simple, otherwise FALSE.

13.13.2.189 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

hGeom	The Geometry to test.

Returns

TRUE if the geometry has no points, otherwise FALSE.

13.13.2.190 double OGR_G_Length (OGRGeometryH *hGeom*)

Compute length of a geometry.

Computes the length for **OGRCurve** (p. **??**) or MultiCurve objects. Undefined for all other geometry types (returns zero).

This function utilizes the C++ get_Length() method.

Parameters

hGeom	the geometry to operate on.

Returns

the lenght or 0.0 for unsupported geometry types.

Since

OGR 1.8.0

References CPLError(), OGR_GT_IsCurve(), OGR_GT_IsSubClassOf(), wkbFlatten, wkbGeometryCollection, and wkbMultiCurve.

13.13.2.191 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 CPLE_NotSupported error.

Parameters

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if they are overlapping, otherwise FALSE.

13.13.2.192 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 CPLE_NotSupported error.

Parameters

hGeom	the geometry to operate on.

Returns

a point guaranteed to lie on the surface or NULL if an error occured.

Since

OGR 1.10

References OGRGeometry::assignSpatialReference(), CPLError(), OGRGeometry::getGeometryType(), OGR↩Geometry::getSpatialReference(), wkbFlatten, and wkbPoint.

Referenced by OGRPolygon::PointOnSurface(), and OGRMultiPolygon::PointOnSurface().

13.13.2.193 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 corresponds to a MultiLinestring, or when reassembling Edges into Polygons is impossible due to topogical 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 CPLE_NotSupported error.

Parameters

hTarget	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

13.13.2.194 OGRErr OGR_G_RemoveGeometry (OGRGeometryH *hGeom,* int *iGeom,* int *bDelete*)

Remove a geometry from an exiting 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 CPP method **OGRGeometryCollection::removeGeometry()** (p. **??**).

Parameters

hGeom	the existing geometry to delete from.
iGeom	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
bDelete	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(), OGR_GT_IsSubClassOf(), wkbCurvePolygon, wkbFlatten, and wkbGeometryCollection.

13.13.2.195 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

hGeom	handle on the geometry to segmentize
dfMaxLength	the maximum distance between 2 points after segmentization

References CPLError().

13.13.2.196 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

hGeom	handle on the geometry to set the dimension of the coordinates.
nNewDimension	New coordinate dimension value, either 2 or 3.

13.13.2.197 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 iPoint is larger than the number of existing points in the linestring, the point count will be increased to accommodate the request.

Parameters

hGeom	handle to the geometry to add a vertex to.
i	the index of the vertex to assign (zero based) or zero for a point.
dfX	input X coordinate to assign.
dfY	input Y coordinate to assign.
dfZ	input Z coordinate to assign (defaults to zero).

References CPLError(), wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

13.13.2.198 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 accommodate the request.

Parameters

hGeom	handle to the geometry to add a vertex to.
i	the index of the vertex to assign (zero based) or zero for a point.
dfX	input X coordinate to assign.
dfY	input Y coordinate to assign.

References CPLError(), wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

13.13.2.199 void OGR_G_SetPointCount (OGRGeometryH *hGeom*, int *nNewPointCount*)

Set number of points in a geometry.

This method primary exists to preset the number of points in a linestring geometry before setPoint() is used to assign them to avoid reallocating the array larger with each call to addPoint().

Parameters

nNewPointCount	the new number of points for geometry.

References CPLError(), OGRSimpleCurve::setNumPoints(), wkbCircularString, wkbFlatten, and wkbLineString.

13.13.2.200 void OGR_G_SetPoints (OGRGeometryH *hGeom*, int *nPointsIn*, void * *pabyX*, int *nXStride*, void * *pabyY*, int *nYStride*, void * *pabyZ*, int *nZStride*)

Assign all points in a point or a line string geometry.

This method clear any existing points assigned to this geometry, and assigns a whole new set.

Parameters

hGeom	handle to the geometry to set the coordinates.
nPointsIn	number of points being passed in padfX and padfY.
padfX	list of X coordinates of points being assigned.
nXStride	the number of bytes between 2 elements of pabyX.
padfY	list of Y coordinates of points being assigned.
nYStride	the number of bytes between 2 elements of pabyY.
padfZ	list of Z coordinates of points being assigned (defaults to NULL for 2D objects).
nZStride	the number of bytes between 2 elements of pabyZ.

References CPLError(), OGRSimpleCurve::setNumPoints(), OGRSimpleCurve::setPoint(), OGRSimpleCurve::setPoints(), wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

13.13.2.201 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

hThis	the geometry.
dTolerance	the distance tolerance for the simplification.

Returns

the simplified geometry or NULL if an error occurs.

Since

OGR 1.8.0

13.13.2.202 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

hThis	the geometry.
dTolerance	the distance tolerance for the simplification.

Returns

the simplified geometry or NULL if an error occurs.

Since

OGR 1.9.0

13.13.2.203 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

hThis	the geometry.
hOther	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

13.13.2.204 OGRGeometryH OGR_G_SymmetricDifference (OGRGeometryH *hThis,* OGRGeometryH *hOther*)

Compute symmetric difference (deprecated)

Deprecated

See also

OGR_G_SymmetricDifference() (p. **??**)

13.13.2.205 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

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if they are touching, otherwise FALSE.

13.13.2.206 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 **OGR↩SpatialReference** (p. **??**) of the **OGRCoordinateTransformation** (p. **??**) will be assigned to the geometry.

This function is the same as the CPP method **OGRGeometry::transform** (p. **??**).

Parameters

hGeom	handle on the geometry to apply the transform to.
hTransform	handle on the transformation to apply.

Returns

OGRERR_NONE on success or an error code.

13.13.2.207 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

hGeom	handle on the geometry to apply the transform to.
hSRS	handle on the spatial reference system to apply.

Returns

OGRERR_NONE on success, or an error code.

13.13.2.208 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

hThis	the geometry.
hOther	the other geometry.

Returns

a new geometry representing the union or NULL if an error occurs.

13.13.2.209 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

hThis	the geometry.

Returns

a new geometry representing the union or NULL if an error occurs.

13.13.2.210 OGRGeometryH OGR_G_Value (OGRGeometryH *hGeom*, double *dfDistance*)

Fetch point at given distance along curve.

This function relates to the SF COM ICurve::get_Value() method.

This function is the same as the C++ method **OGRCurve::Value()** (p. **??**).

Parameters

hGeom	curve geometry.
dfDistance	distance along the curve at which to sample position. This distance should be between zero and get_Length() for this curve.

Returns

a point or NULL.

Since

GDAL 2.0

References OGR_GT_IsCurve().

13.13.2.211 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

hThis	the geometry to compare.
hOther	the other geometry to compare.

Returns

TRUE if hThis is within hOther, otherwise FALSE.

13.13.2.212 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

hGeom	handle on the geometry to get the binary size from.

Returns

size of binary representation in bytes.

13.13.2.213 const char* OGR_GetFieldSubTypeName (OGRFieldSubType *eSubType*)

Fetch human readable name for a field subtype.

This function is the same as the CPP method **OGRFieldDefn::GetFieldSubTypeName()** (p. **??**).

Parameters

eSubType	the field subtype to get name for.

Returns

the name.

Since

GDAL 2.0

References OGRFieldDefn::GetFieldSubTypeName().

13.13.2.214 const char* OGR_GetFieldTypeName (OGRFieldType *eType*)

Fetch human readable name for a field type.

This function is the same as the CPP method **OGRFieldDefn::GetFieldTypeName()** (p. **??**).

Parameters

eType	the field type to get name for.

Returns

the name.

References OGRFieldDefn::GetFieldTypeName().

13.13.2.215 OGRGeomFieldDefnH OGR_GFld_Create (const char * *pszName,* OGRwkbGeometryType *eType*)

Create a new field geometry definition.

This function is the same as the CPP method **OGRGeomFieldDefn::OGRGeomFieldDefn()** (p. **??**).

Parameters

pszName	the name of the new field definition.
eType	the type of the new field definition.

Returns

handle to the new field definition.

Since

GDAL 1.11

13.13.2.216 void OGR_GFld_Destroy (OGRGeomFieldDefnH *hDefn*)

Destroy a geometry field definition.

Parameters

hDefn	handle to the geometry field definition to destroy.

Since

GDAL 1.11

13.13.2.217 const char* OGR_GFld_GetNameRef (OGRGeomFieldDefnH *hDefn*)

Fetch name of this field.

This function is the same as the CPP method **OGRGeomFieldDefn::GetNameRef()** (p. **??**).

Parameters

hDefn	handle to the geometry field definition.

Returns

the name of the geometry field definition.

Since

GDAL 1.11

13.13.2.218 OGRSpatialReferenceH OGR_GFld_GetSpatialRef (OGRGeomFieldDefnH *hDefn*)

Fetch spatial reference system of this field.

This function is the same as the C++ method **OGRGeomFieldDefn::GetSpatialRef()** (p. **??**).

Parameters

hDefn	handle to the geometry field definition

Returns

field spatial reference system.

Since

GDAL 1.11

13.13.2.219 OGRwkbGeometryType OGR_GFld_GetType (OGRGeomFieldDefnH *hDefn*)

Fetch geometry type of this field.

This function is the same as the CPP method **OGRGeomFieldDefn::GetType()** (p. **??**).

Parameters

hDefn	handle to the geometry field definition to get type from.

Returns

field geometry type.

Since

GDAL 1.11

References OGR_GT_GetLinear(), OGR_GT_IsNonLinear(), OGRGetNonLinearGeometriesEnabledFlag(), and wkbUnknown.

13.13.2.220 int OGR_GFld_IsIgnored (OGRGeomFieldDefnH *hDefn*)

Return whether this field should be omitted when fetching features.

This method is the same as the C++ method **OGRGeomFieldDefn::IsIgnored()** (p. **??**).

Parameters

hDefn	handle to the geometry field definition

Returns

ignore state

Since

GDAL 1.11

13.13.2.221 int OGR_GFld_IsNullable (OGRGeomFieldDefnH *hDefn*)

Return whether this geometry field can receive null values.

By default, fields are nullable.

Even if this method returns FALSE (i.e not-nullable field), it doesn't mean that **OGRFeature::IsFieldSet()** (p. **??**) will necessary return TRUE, as fields can be temporary unset and null/not-null validation is usually done when **OGRLayer::CreateFeature()** (p. **??**)/SetFeature() is called.

Note that not-nullable geometry fields might also contain 'empty' geometries.

This method is the same as the C++ method **OGRGeomFieldDefn::IsNullable()** (p. **??**).

Parameters

hDefn	handle to the field definition

Returns

TRUE if the field is authorized to be null.

Since

GDAL 2.0

13.13.2.222 void OGR_GFld_SetIgnored (OGRGeomFieldDefnH *hDefn*, int *ignore*)

Set whether this field should be omitted when fetching features.

This method is the same as the C++ method **OGRGeomFieldDefn::SetIgnored()** (p. **??**).

Parameters

hDefn	handle to the geometry field definition
ignore	ignore state

Since

GDAL 1.11

13.13.2.223 void OGR_GFld_SetName (OGRGeomFieldDefnH *hDefn*, const char * *pszName*)

Reset the name of this field.

This function is the same as the CPP method **OGRGeomFieldDefn::SetName()** (p. **??**).

Parameters

hDefn	handle to the geometry field definition to apply the new name to.
pszName	the new name to apply.

Since

GDAL 1.11

13.13.2.224 void OGR_GFld_SetNullable (OGRGeomFieldDefnH *hDefn*, int *bNullableIn*)

Set whether this geometry field can receive null values.

By default, fields are nullable, so this method is generally called with FALSE to set a not-null constraint.

Drivers that support writing not-null constraint will advertize the GDAL_DCAP_NOTNULL_GEOMFIELDS driver metadata item.

This method is the same as the C++ method **OGRGeomFieldDefn::SetNullable()** (p. **??**).

Parameters

hDefn	handle to the field definition

bNullableIn	FALSE if the field must have a not-null constraint.

Since

GDAL 2.0

13.13.2.225 void OGR_GFld_SetSpatialRef (OGRGeomFieldDefnH *hDefn,* OGRSpatialReferenceH *hSRS*)

Set the spatial reference of this field.

This function is the same as the C++ method **OGRGeomFieldDefn::SetSpatialRef()** (p. **??**).

This function drops the reference of the previously set SRS object and acquires a new reference on the passed object (if non-NULL).

Parameters

hDefn	handle to the geometry field definition
hSRS	the new SRS to apply.

Since

GDAL 1.11

13.13.2.226 void OGR_GFld_SetType (OGRGeomFieldDefnH *hDefn,* OGRwkbGeometryType *eType*)

Set the geometry type of this field. This should never be done to an **OGRGeomFieldDefn** (p. **??**) that is already part of an **OGRFeatureDefn** (p. **??**).

This function is the same as the CPP method **OGRGeomFieldDefn::SetType()** (p. **??**).

Parameters

hDefn	handle to the geometry field definition to set type to.
eType	the new field geometry type.

Since

GDAL 1.11

13.13.2.227 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 existance 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 OLCAlterFieldDefn capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly. Some drivers might also not support all update flags.

This function is the same as the C++ method **OGRLayer::AlterFieldDefn()** (p. **??**).

Parameters

hLayer	handle to the layer.
iField	index of the field whose definition must be altered.
hNewFieldDefn	new field definition
nFlags	combination of ALTER_NAME_FLAG, ALTER_TYPE_FLAG, ALTER_WIDTH_PRECISIO↩ N_FLAG, ALTER_NULLABLE_FLAG and ALTER_DEFAULT_FLAG to indicate which of the name and/or type and/or width and precision fields and/or nullability from the new field definition must be taken into account.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

13.13.2.228 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

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

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

Returns

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

Since

OGR 1.10

13.13.2.229 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

hLayer	handle to the layer

Returns

OGRERR_NONE on success.

13.13.2.230 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

hLayer	handle to the layer to write the feature to.
hFeat	the handle of the feature to write to disk.

Returns

OGRERR_NONE on success.

13.13.2.231 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 existance 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.

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

This function is the same as the C++ method **OGRLayer::CreateField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

13.13.2.232 OGRErr OGR_L_CreateGeomField (OGRLayerH *hLayer,* OGRGeomFieldDefnH *hField,* int *bApproxOK*)

Create a new geometry field on a layer.

You must use this to create new geometry 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 existance 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.

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

This function is the same as the C++ method **OGRLayer::CreateField()** (p. **??**).

Parameters

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

Returns

OGRERR_NONE on success.

Since

OGR 1.11

13.13.2.233 OGRErr OGR_L_DeleteFeature (OGRLayerH *hLayer,* GIntBig *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

hLayer	handle to the layer
nFID	the feature id to be deleted from the layer

Returns

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

13.13.2.234 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 existance 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

hLayer	handle to the layer.
iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

13.13.2.235 OGRErr OGR_L_Erase (OGRLayerH *pLayerInput,* OGRLayerH *pLayerMethod,* OGRLayerH *pLayerResult,* char ∗∗ *papszOptions,* 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

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

Returns

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

Since

OGR 1.10

13.13.2.236 int OGR_L_FindFieldIndex (OGRLayerH *hLayer,* const char * , int *bExactMatch*)

Find the index of field in a layer.

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

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

This method is the same as the C++ method **OGRLayer::FindFieldIndex()** (p. **??**).

Returns

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

13.13.2.237 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

hLayer	handle to the layer from which to get extent.
psExtent	the structure in which the extent value will be returned.
bForce	Flag indicating whether the extent should be computed even if it is expensive.

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

13.13.2.238 OGRErr OGR_L_GetExtentEx (OGRLayerH *hLayer,* int *iGeomField,* OGREnvelope * *psExtent,* int *bForce*)

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

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

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **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

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

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

13.13.2.239 OGRFeatureH OGR_L_GetFeature (OGRLayerH *hLayer,* GIntBig *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 (and specialized implementations in drivers should make sure that they do not take into account 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 (with **OGR_L_GetNextFeature()** (p. **??**)) 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

hLayer	handle to the layer that owned the feature.
nFeatureId	the feature id of the feature to read.

Returns

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

13.13.2.240 GIntBig 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. **??**).

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

Parameters

hLayer	handle to the layer that owned the features.
bForce	Flag indicating whether the count should be computed even if it is expensive.

Returns

feature count, -1 if count not known.

13.13.2.241 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

hLayer	handle to the layer

Returns

fid column name.

13.13.2.242 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.

For layers with multiple geometry fields, this method only returns the geometry type of the first geometry column. For other columns, use OGR_GFld_GetNameRef(OGR_FD_GetGeomFieldDefn(OGR_L_GetLayerDefn(hLayer), i)).

This method is the same as the C++ method **OGRLayer::GetGeometryColumn()** (p. **??**)

Parameters

hLayer	handle to the layer

Returns

geometry column name.

13.13.2.243 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.

For layers with multiple geometry fields, this method only returns the geometry type of the first geometry column. For other columns, use OGR_GFld_GetType(OGR_FD_GetGeomFieldDefn(OGR_L_GetLayerDefn(hLayer), i)). For layers without any geometry field, this method returns wkbNone.

This function is the same as the C++ method **OGRLayer::GetGeomType()** (p. **??**).

Parameters

hLayer	handle to the layer.

Returns

the geometry type

Since

OGR 1.8.0

References OGR_GT_GetLinear(), OGR_GT_IsNonLinear(), OGRGetNonLinearGeometriesEnabledFlag(), and wkbUnknown.

13.13.2.244 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

hLayer	handle to the layer to get the schema information.

Returns

an handle to the feature definition.

13.13.2.245 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

hLayer	handle to the layer.

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

13.13.2.246 OGRFeatureH OGR_L_GetNextFeature (OGRLayerH *hLayer*)

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity 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.

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

This function is the same as the C++ method **OGRLayer::GetNextFeature()** (p. **??**).

Parameters

hLayer	handle to the layer from which feature are read.

Returns

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

13.13.2.247 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

hLayer	handle to the layer to get the spatial filter from.

Returns

an handle to the spatial filter geometry.

13.13.2.248 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

hLayer	handle to the layer to get the spatial reference from.

Returns

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

13.13.2.249 OGRErr OGR_L_Identity (OGRLayerH *pLayerInput,* OGRLayerH *pLayerMethod,* OGRLayerH *pLayerResult,* char ** *papszOptions,* GDALProgressFunc *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 **OGRLayer::Identity()** (p. **??**).

Parameters

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

Returns

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

Since

OGR 1.10

13.13.2.250 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

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

Returns

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

Since

OGR 1.10

13.13.2.251 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 existance that were obtained or created with the previous layer definition.

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

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

Not all drivers support this 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

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

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

13.13.2.252 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 existance that were obtained or created with the previous layer definition.

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

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

Not all drivers support this 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

hLayer	handle to the layer.
panMap	an array of GetLayerDefn()->**OGRFeatureDefn::GetFieldCount()** (p. **??**) elements which is a permutation of [0, GetLayerDefn()->**OGRFeatureDefn::GetFieldCount()** (p. **??**)-1].

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

13.13.2.253 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

hLayer	handle to the layer on which features are read.

13.13.2.254 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

hLayer	handle to the layer

Returns

OGRERR_NONE on success.

13.13.2.255 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_Reset↩ Reading()** (p. **??**)).

This function is the same as the C++ method **OGRLayer::SetAttributeFilter()** (p. **??**).

Parameters

hLayer	handle to the layer on which attribute query will be executed.
pszQuery	query in restricted SQL WHERE format, or NULL to clear the current query.

Returns

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

13.13.2.256 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

hLayer	handle to the layer to write the feature.
hFeat	the feature to write.

Returns

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

13.13.2.257 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

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

Returns

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

13.13.2.258 OGRErr OGR_L_SetNextByIndex (OGRLayerH *hLayer,* GIntBig *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

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

Returns

OGRERR_NONE on success or an error code.

13.13.2.259 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_GetNext↩ Feature()** (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

hLayer	handle to the layer on which to set the spatial filter.
hGeom	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.

13.13.2.260 void OGR_L_SetSpatialFilterEx (OGRLayerH *hLayer,* int *iGeomField,* 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_GetNext↩ Feature()** (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 geometry field definition it corresponds to (as returned by GetLayerDefn()->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->GetSpatial↩ Ref()). In the future this may be generalized.

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

This function is the same as the C++ method **OGRLayer::SetSpatialFilter** (p. **??**).

Parameters

hLayer	handle to the layer on which to set the spatial filter.
iGeomField	index of the geometry field on which the spatial filter operates.

hGeom	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.

Since

GDAL 1.11

13.13.2.261 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

hLayer	handle to the layer on which to set the spatial filter.
dfMinX	the minimum X coordinate for the rectangular region.
dfMinY	the minimum Y coordinate for the rectangular region.
dfMaxX	the maximum X coordinate for the rectangular region.
dfMaxY	the maximum Y coordinate for the rectangular region.

13.13.2.262 void OGR_L_SetSpatialFilterRectEx (OGRLayerH *hLayer,* int *iGeomField,* 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 as the geometry field definition it corresponds to (as returned by GetLayerDefn()->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->GetSpatialRef()). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (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

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

Since

GDAL 1.11

13.13.2.263 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.

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

This function is the same as the C++ method OGRLayer::StartTransaction().

Parameters

hLayer	handle to the layer

Returns

OGRERR_NONE on success.

13.13.2.264 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

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

Returns

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

Since

OGR 1.10

13.13.2.265 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

hLayer	handle to the layer

Returns

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

13.13.2.266 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 Create↩ Field(), otherwise FALSE.
- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using CreateGeomField(), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using DeleteField(), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using ReorderField() or ReorderFields(), otherwise FALSE.
- **OLCAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using AlterFieldDefn(), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the DeleteFeature() method is supported on this layer, otherwise FALSE.
- **OLCStringsAsUTF8** / "StringsAsUTF8": TRUE if values of OFTString fields are assured to be in UTF-8 format. If FALSE the encoding of fields is uncertain, though it might still be UTF-8.
- **OLCTransactions** / "Transactions": TRUE if the StartTransaction(), CommitTransaction() and Rollback↩ Transaction() methods work in a meaningful way, otherwise FALSE.
- **OLCCurveGeometries** / "CurveGeometries": TRUE if this layer supports writing curve geometries or may return such geometries. (GDAL 2.0).

This function is the same as the C++ method **OGRLayer::TestCapability()** (p. **??**).

Parameters

hLayer	handle to the layer to get the capability from.
pszCap	the name of the capability to test.

Returns

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

13.13.2.267 OGRErr OGR_L_Union (OGRLayerH *pLayerInput,* OGRLayerH *pLayerMethod,* OGRLayerH *pLayerResult,* char ** *papszOptions,* 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

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

Returns

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

Since

OGR 1.10

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

Note

If the schema of the result is set by user and contains fields that have the same name as a field in the method layer, then the attribute in the result feature the originates from the method layer will get the value from the feature of the method layer.
For best performance use the minimum amount of features in the method layer and copy it into a memory layer.
This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted.
- 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

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

Returns

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

Since

OGR 1.10

13.13.2.269 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

hSM	handle to the style manager.
hST	the style tool defining the part to add.

Returns

TRUE on success, FALSE on errors.

13.13.2.270 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

hSM	handle to the style manager.
pszStyleName	the name of the style to add.
pszStyleString	the style string to use, or NULL to use the style stored in the manager.

Returns

TRUE on success, FALSE on errors.

13.13.2.271 OGRStyleMgrH OGR_SM_Create (OGRStyleTableH *hStyleTable*)

OGRStyleMgr (p. **??**) factory.

This function is the same as the C++ method **OGRStyleMgr::OGRStyleMgr()** (p. **??**).

Parameters

hStyleTable	pointer to **OGRStyleTable** (p. **??**) or NULL if not working with a style table.

Returns

an handle to the new style manager object.

13.13.2.272 void OGR_SM_Destroy (OGRStyleMgrH *hSM*)

Destroy Style Manager.

This function is the same as the C++ method **OGRStyleMgr::~OGRStyleMgr()** (p. **??**).

Parameters

hSM	handle to the style manager to destroy.

13.13.2.273 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

hSM	handle to the style manager.
nPartId	the part number (0-based index).
pszStyleString	(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.

13.13.2.274 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

hSM	handle to the style manager.
pszStyleString	(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.

13.13.2.275 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

hSM	handle to the style manager.
hFeat	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.

13.13.2.276 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

hSM	handle to the style manager.
pszStyleString	the style string to use (can be NULL).

Returns

TRUE on success, FALSE on errors.

13.13.2.277 OGRStyleToolH OGR_ST_Create (OGRSTClassId *eClassId*)

OGRStyleTool (p. **??**) factory.

This function is a constructor for **OGRStyleTool** (p. **??**) derived classes.

Parameters

eClassId	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.

13.13.2.278 void OGR_ST_Destroy (OGRStyleToolH *hST*)

Destroy Style Tool.

Parameters

hST	handle to the style tool to destroy.

13.13.2.279 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

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
bValueIsNull	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.

13.13.2.280 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

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
bValueIsNull	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.

13.13.2.281 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

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)

bValueIsNull	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.

13.13.2.282 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

hST	handle to the style tool.
pszColor	the color to parse
pnRed	pointer to an int in which the red value will be returned
pnGreen	pointer to an int in which the green value will be returned
pnBlue	pointer to an int in which the blue value will be returned
pnAlpha	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.

13.13.2.283 const char* OGR_ST_GetStyleString (OGRStyleToolH *hST*)

Get the style string for this Style Tool.

Maps to the **OGRStyleTool** (p. **??**) subclasses' GetStyleString() methods.

Parameters

hST	handle to the style tool.

Returns

the style string for this style tool or "" if the hST is invalid.

13.13.2.284 OGRSTClassId OGR_ST_GetType (OGRStyleToolH *hST*)

Determine type of Style Tool.

Parameters

hST	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.

13.13.2.285 OGRSTUnitId OGR_ST_GetUnit (OGRStyleToolH *hST*)

Get Style Tool units.

Parameters

hST	handle to the style tool.

Returns

the style tool units.

13.13.2.286 void OGR_ST_SetParamDbl (OGRStyleToolH *hST,* int *eParam,* double *dfValue*)

Set Style Tool parameter value from a double.

Maps to the **OGRStyleTool** (p. **??**) subclasses' SetParamDbl() methods.

Parameters

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
dfValue	the new parameter value

13.13.2.287 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

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
nValue	the new parameter value

13.13.2.288 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

hST	handle to the style tool.
eParam	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
pszValue	the new parameter value

13.13.2.289 void OGR_ST_SetUnit (OGRStyleToolH *hST,* OGRSTUnitId *eUnit,* double *dfGroundPaperScale*)

Set Style Tool units.

This function is the same as OGRStyleTool::SetUnit()

Parameters

hST	handle to the style tool.
eUnit	the new unit.
dfGround↩ PaperScale	ground to paper scale factor.

13.13.2.290 int OGR_STBL_AddStyle (OGRStyleTableH *hStyleTable,* 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. This function is the same as the C++ method **OGRStyleTable::AddStyle()** (p. **??**).

Parameters

hStyleTable	handle to the style table.
pszName	the name the style to add.
pszStyleString	the style string to add.

Returns

TRUE on success, FALSE on error

13.13.2.291 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.

13.13.2.292 void OGR_STBL_Destroy (OGRStyleTableH *hSTBL*)

Destroy Style Table.

Parameters

hSTBL	handle to the style table to destroy.

13.13.2.293 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

hStyleTable	handle to the style table.
pszName	the name of the style string to find.

Returns

the style string matching the name or NULL if not found or error.

13.13.2.294 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

hStyleTable	handle to the style table.

Returns

the Name of the last style string or NULL on error.

13.13.2.295 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

hStyleTable	handle to the style table.

Returns

the next style string or NULL on error.

13.13.2.296 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

hStyleTable	handle to the style table.
pszFilename	the name of the file to load from.

Returns

TRUE on success, FALSE on error

13.13.2.297 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

hStyleTable	handle to the style table.

13.13.2.298 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

hStyleTable	handle to the style table.
pszFilename	the name of the file to save to.

Returns

TRUE on success, FALSE on error

13.13.2.299 OGRGeometryH OGRBuildPolygonFromEdges (OGRGeometryH *hLines,* int *bBestEffort,* int *bAutoClose,* double *dfTolerance,* OGRErr * *peErr*)

Build a ring from a bunch of arcs.

Parameters

hLines	handle to an **OGRGeometryCollection** (p. **??**) (or **OGRMultiLineString** (p. **??**)) containing the line string geometries to be built into rings.
bBestEffort	not yet implemented???.
bAutoClose	indicates if the ring should be close when first and last points of the ring are the same.
dfTolerance	tolerance into which two arcs are considered close enough to be joined.
peErr	OGRERR_NONE on success, or OGRERR_FAILURE on failure.

Returns

an handle to the new geometry, a polygon.

References OGRSimpleCurve::addPoint(), OGRCurvePolygon::addRingDirectly(), CPLCalloc(), CPLDebug(), CPLError(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometry↩Collection::getNumGeometries(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getX(), OGRSimple↩Curve::getY(), OGRSimpleCurve::getZ(), wkbFlatten, wkbGeometryCollection, wkbLineString, and wkbMultiLine↩String.

13.13.2.300 void OGRCleanupAll (void)

Cleanup all OGR related resources.

FIXME

13.13.2.301 OGRSFDriverH OGRGetDriver (int *iDriver*)

Fetch the indicated driver.

NOTE: Starting with GDAL 2.0, it is *NOT* safe to cast the returned handle to OGRSFDriver*. If a C++ object is needed, the handle should be cast to GDALDriver*.

Deprecated Use GDALGetDriver() in GDAL 2.0

Parameters

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

Returns

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

13.13.2.302 OGRSFDriverH OGRGetDriverByName (const char * *pszName*)

Fetch the indicated driver.

NOTE: Starting with GDAL 2.0, it is *NOT* safe to cast the returned handle to OGRSFDriver*. If a C++ object is needed, the handle should be cast to GDALDriver*.

Deprecated Use GDALGetDriverByName() in GDAL 2.0

Parameters

pszName	the driver name

Returns

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

13.13.2.303 int OGRGetDriverCount (void)

Fetch the number of registered drivers.

Deprecated Use GDALGetDriverCount() in GDAL 2.0

Returns

the drivers count.

13.13.2.304 int OGRGetNonLinearGeometriesEnabledFlag (void)

Get flag to enable/disable returning non-linear geometries in the C API.

return TRUE if non-linear geometries might be returned (default value is TRUE)

Since

GDAL 2.0

See also

OGRSetNonLinearGeometriesEnabledFlag() (p. **??**)

Referenced by OGR_F_GetGeometryRef(), OGR_F_GetGeomFieldRef(), OGR_FD_GetGeomType(), OGR_GFld_GetType(), and OGR_L_GetGeomType().

13.13.2.305 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.

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. **??**))

NOTE: Starting with GDAL 2.0, it is *NOT* safe to cast the returned handle to OGRDataSource∗. If a C++ object is needed, the handle should be cast to GDALDataset∗. Similarly, the returned OGRSFDriverH handle should be cast to GDALDriver∗, and NOT∗ OGRSFDriver∗.

Deprecated Use GDALOpenEx() in GDAL 2.0

Parameters

pszName	the name of the file, or data source to open.
bUpdate	FALSE for read-only access (the default) or TRUE for read-write access.
pahDriverList	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 a GDALDataset. This GDALDataset 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 );
```

13.13.2.306 int OGRRegisterAll (void)

Register all drivers.

Deprecated Use GDALAllRegister() in GDAL 2.0

13.13.2.307 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.

Deprecated Use GDALClose() in GDAL 2.0

Parameters

hDS	handle to the data source to release

Returns

OGRERR_NONE on success or an error code.

Referenced by OGRGeocodeDestroySession().

13.13.2.308 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.

13.13.2.309 void OGRSetNonLinearGeometriesEnabledFlag (int *bFlag*)

Set flag to enable/disable returning non-linear geometries in the C API.

This flag has only an effect on the **OGR_F_GetGeometryRef()** (p. **??**), **OGR_F_GetGeomFieldRef()** (p. **??**), **OGR_L_GetGeomType()** (p. **??**), **OGR_GFld_GetType()** (p. **??**) and **OGR_FD_GetGeomType()** (p. **??**) C API, and corresponding methods in the SWIG bindings. It is meant as making it simple for applications using the OGR C API not to have to deal with non-linear geometries, even if such geometries might be returned by drivers. In which case, they will be transformed into their closest linear geometry, by doing linear approximation, with **OGR_G_ForceTo()** (p. **??**).

Libraries should generally *not* use that method, since that could interfere with other libraries or applications.

Note that it *does* not affect the behaviour of the C++ API.

Parameters

bFlag	TRUE if non-linear geometries might be returned (default value). FALSE to ask for non-linear geometries to be approximated as linear geometries.

Returns

a point or NULL.

Since

GDAL 2.0

13.14 ogr_core.h File Reference

```
#include "cpl_port.h"
#include "gdal_version.h"
```

Classes

- class **OGREnvelope**
- class **OGREnvelope3D**
- union **OGRField**

Macros

- #define **wkbCurve** ((**OGRwkbGeometryType**)13)
- #define **wkbSurface** ((**OGRwkbGeometryType**)14)
- #define **wkb25DBit** 0x80000000
- #define **wkbFlatten**(x) **OGR_GT_Flatten**((**OGRwkbGeometryType**)(x))
- #define **wkbHasZ**(x) **OGR_GT_HasZ**(x)
- #define **wkbSetZ**(x) **OGR_GT_SetZ**(x)
- #define **ALTER_NAME_FLAG** 0x1
- #define **ALTER_TYPE_FLAG** 0x2
- #define **ALTER_WIDTH_PRECISION_FLAG** 0x4
- #define **ALTER_NULLABLE_FLAG** 0x8
- #define **ALTER_DEFAULT_FLAG** 0x10
- #define **ALTER_ALL_FLAG** (**ALTER_NAME_FLAG** | **ALTER_TYPE_FLAG** | **ALTER_WIDTH_PRECISI↩ ON_FLAG** | **ALTER_NULLABLE_FLAG** | **ALTER_DEFAULT_FLAG**)
- #define **OGR_F_VAL_NULL** 0x00000001
- #define **OGR_F_VAL_GEOM_TYPE** 0x00000002
- #define **OGR_F_VAL_WIDTH** 0x00000004
- #define **OGR_F_VAL_ALLOW_NULL_WHEN_DEFAULT** 0x00000008
- #define **OGR_F_VAL_ALL** 0xFFFFFFFF
- #define **OLMD_FID64** "OLMD_FID64"
- #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,
  wkbCircularString = 8, **wkbCompoundCurve** = 9, **wkbCurvePolygon** = 10, **wkbMultiCurve** = 11,
  wkbMultiSurface = 12, **wkbNone** = 100, **wkbLinearRing** = 101, **wkbCircularStringZ** = 1008,
  wkbCompoundCurveZ = 1009, **wkbCurvePolygonZ** = 1010, **wkbMultiCurveZ** = 1011, **wkbMulti↩ SurfaceZ** = 1012,
  wkbPoint25D = 0x80000001, **wkbLineString25D** = 0x80000002, **wkbPolygon25D** = 0x80000003, **wkb↩ MultiPoint25D** = 0x80000004,
  wkbMultiLineString25D = 0x80000005, **wkbMultiPolygon25D** = 0x80000006, **wkbGeometry↩ Collection25D** = 0x80000007 }
- enum **OGRwkbVariant** { **wkbVariantOldOgc**, **wkbVariantIso**, **wkbVariantPostGIS1** }
- 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,
  OFTInteger64 = 12, **OFTInteger64List** = 13 }
- enum **OGRFieldSubType** { **OFSTNone** = 0, **OFSTBoolean** = 1, **OFSTInt16** = 2, **OFSTFloat32** = 3 }
- 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.
- **OGRwkbGeometryType OGRMergeGeometryTypesEx** (**OGRwkbGeometryType** eMain, **OGRwkbGeometryType** eExtra, int bAllowPromotingToCurves)

  Find common geometry type.
- **OGRwkbGeometryType OGR_GT_Flatten** (**OGRwkbGeometryType** eType)

  Returns the 2D geometry type corresponding to the passed geometry type.
- **OGRwkbGeometryType OGR_GT_SetZ** (**OGRwkbGeometryType** eType)

  Returns the 3D geometry type corresponding to the passed geometry type.
- int **OGR_GT_HasZ** (**OGRwkbGeometryType** eType)

  Return if the geometry type is a 3D geometry type.
- int **OGR_GT_IsSubClassOf** (**OGRwkbGeometryType** eType, **OGRwkbGeometryType** eSuperType)

  Returns if a type is a subclass of another one.
- int **OGR_GT_IsCurve** (**OGRwkbGeometryType**)

  Return if a geometry type is an instance of Curve.
- int **OGR_GT_IsSurface** (**OGRwkbGeometryType**)

  Return if a geometry type is an instance of Surface.
- int **OGR_GT_IsNonLinear** (**OGRwkbGeometryType**)

  Return if a geometry type is a non-linear geometry type.
- **OGRwkbGeometryType OGR_GT_GetCollection** (**OGRwkbGeometryType** eType)

  Returns the collection type that can contain the passed geometry type.
- **OGRwkbGeometryType OGR_GT_GetCurve** (**OGRwkbGeometryType** eType)

  Returns the curve geometry type that can contain the passed geometry type.
- **OGRwkbGeometryType OGR_GT_GetLinear** (**OGRwkbGeometryType** eType)

  Returns the non-curve geometry type that can contain the passed geometry type.
- int CPL_STDCALL **GDALCheckVersion** (int nVersionMajor, int nVersionMinor, const char *pszCallingComponentName)

13.14.1 Detailed Description

Core portability services for cross-platform OGR code.

13.14.2 Macro Definition Documentation

13.14.2.1 #define ALTER_ALL_FLAG (ALTER_NAME_FLAG | ALTER_TYPE_FLAG | ALTER_WIDTH_PRECISION_FLAG | ALTER_NULLABLE_FLAG | ALTER_DEFAULT_FLAG)

Alter all parameters of field definition. Used by **OGR_L_AlterFieldDefn()** (p. **??**).

13.14.2.2 #define ALTER_DEFAULT_FLAG 0x10

Alter field DEFAULT value. Used by **OGR_L_AlterFieldDefn()** (p. **??**).

Since

GDAL 2.0

13.14.2.3 #define ALTER_NAME_FLAG 0x1

Alter field name. Used by **OGR_L_AlterFieldDefn()** (p. **??**).

13.14.2.4 #define ALTER_NULLABLE_FLAG 0x8

Alter field NOT NULL constraint. Used by **OGR_L_AlterFieldDefn()** (p. **??**).

Since

GDAL 2.0

13.14.2.5 #define ALTER_TYPE_FLAG 0x2

Alter field type. Used by **OGR_L_AlterFieldDefn()** (p. **??**).

13.14.2.6 #define ALTER_WIDTH_PRECISION_FLAG 0x4

Alter field width and precision. Used by **OGR_L_AlterFieldDefn()** (p. **??**).

13.14.2.7 #define GDAL_CHECK_VERSION(*pszCallingComponentName*) GDALCheckVersion(GDAL_VERSION_MAJOR, GDAL_VERSION_MINOR, pszCallingComponentName)

Helper macro for GDALCheckVersion

13.14.2.8 #define OGR_F_VAL_ALL 0xFFFFFFFF

Enable all validation tests. Used by **OGR_F_Validate()** (p. **??**).

Since

GDAL 2.0

13.14.2.9 #define OGR_F_VAL_ALLOW_NULL_WHEN_DEFAULT 0x00000008

Allow fields that are null when there's an associated default value. This can be used for drivers where the low-level layers will automatically set the field value to the associated default value. This flag only makes sense if OGR_F↩ _VAL_NULL is set too. Used by **OGR_F_Validate()** (p. **??**).

Since

GDAL 2.0

Referenced by OGRFeature::Validate().

13.14.2.10 #define OGR_F_VAL_GEOM_TYPE 0x00000002

Validate that geometries respect geometry column type. Used by **OGR_F_Validate()** (p. **??**).

Since

GDAL 2.0

Referenced by OGRFeature::Validate().

13.14.2.11 #define OGR_F_VAL_NULL 0x00000001

Validate that fields respect not-null constraints. Used by **OGR_F_Validate()** (p. **??**).

Since

GDAL 2.0

Referenced by OGRFeature::Validate().

13.14.2.12 #define OGR_F_VAL_WIDTH 0x00000004

Validate that (string) fields respect field width. Used by **OGR_F_Validate()** (p. **??**).

Since

GDAL 2.0

Referenced by OGRFeature::Validate().

13.14.2.13 #define OLMD_FID64 "OLMD_FID64"

Capability set to YES as metadata on a layer that has features with 64 bit identifiers.

Since

GDAL 2.0

13.14.2.14 #define wkb25DBit 0x80000000

Deprecated in GDAL 2.0. Use **wkbHasZ()** (p. **??**) or **wkbSetZ()** (p. **??**) instead

13.14.2.15 #define wkbCurve ((OGRwkbGeometryType)13)

Curve (abstract type). SF-SQL 1.2

Referenced by OGR_GT_IsCurve(), and OGR_GT_IsSubClassOf().

13.14.2.16 #define wkbFlatten(*x*) OGR_GT_Flatten((OGRwkbGeometryType)(x))

Return the 2D geometry type corresponding to the specified geometry type

Referenced by OGRGeometry::Centroid(), OGRGeometryCollection::closeRings(), OGRCurvePolygon::Contains(), OGRGeometryFactory::createGeometry(), OGRGeometry::dumpReadable(), OGRSimpleCurve::exportToWkb(),

OGRGeometryCollection::exportToWkb(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceTo↩ LineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGR↩ GeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRGeometryCollection↩ ::get_Area(), OGRGeometryCollection::get_Length(), OGRPolygon::getCurveGeometry(), OGRGeometry↩ ::getIsoGeometryType(), OGRMultiSurface::importFromWkt(), OGRMultiCurve::importFromWkt(), OGRPoint::↩ Intersects(), OGRCurvePolygon::Intersects(), OGR_G_AddGeometry(), OGR_G_AddGeometryDirectly(), OGR_↩ G_AddPoint(), OGR_G_AddPoint_2D(), OGR_G_Area(), OGR_G_Centroid(), OGR_G_GetGeometryCount(), O↩ GR_G_GetGeometryRef(), OGR_G_GetPoint(), OGR_G_GetPointCount(), OGR_G_GetPoints(), OGR_G_GetX(), OGR_G_GetY(), OGR_G_GetZ(), OGR_G_Length(), OGR_G_PointOnSurface(), OGR_G_RemoveGeometry(), OGR_G_SetPoint(), OGR_G_SetPoint_2D(), OGR_G_SetPointCount(), OGR_G_SetPoints(), OGR_GT_Get↩ Collection(), OGR_GT_GetCurve(), OGR_GT_GetLinear(), OGR_GT_IsNonLinear(), OGR_GT_IsSubClassOf(), OGRBuildPolygonFromEdges(), OGRGeometryTypeToName(), OGRMergeGeometryTypesEx(), OGRGeometry↩ Factory::organizePolygons(), OGRGeometry::Polygonize(), and OGRPoint::Within().

13.14.2.17 #define wkbHasZ(*x*) OGR_GT_HasZ(x)

Return if the geometry type is a 3D geometry type

Since

GDAL 2.0

Referenced by OGRSimpleCurve::exportToWkb(), OGRGeometryCollection::exportToWkb(), OGR_GT_Get↩ Collection(), OGR_GT_GetCurve(), OGR_GT_GetLinear(), and OGRFeature::Validate().

13.14.2.18 #define wkbSetZ(*x*) OGR_GT_SetZ(x)

Return the 3D geometry type corresponding to the specified geometry type.

Since

GDAL 2.0

Referenced by OGR_GT_GetCollection(), OGR_GT_GetCurve(), OGR_GT_GetLinear(), and OGRFeature::↩ Validate().

13.14.2.19 #define wkbSurface ((OGRwkbGeometryType)14)

Surface (abstract type). SF-SQL 1.2

Referenced by OGR_GT_IsSubClassOf(), and OGR_GT_IsSurface().

13.14.3 Typedef Documentation

13.14.3.1 typedef enum ogr_style_tool_param_brush_id OGRSTBrushParam

List of parameters for use with **OGRStyleBrush** (p. **??**).

13.14.3.2 typedef enum ogr_style_tool_class_id OGRSTClassId

OGRStyleTool (p. **??**) derived class types (returned by GetType()).

13.14.3.3 typedef enum ogr_style_tool_param_label_id OGRSTLabelParam

List of parameters for use with **OGRStyleLabel** (p. **??**).

13.14.3.4 typedef enum ogr_style_tool_param_pen_id OGRSTPenParam

List of parameters for use with **OGRStylePen** (p. **??**).

13.14.3.5 typedef enum ogr_style_tool_param_symbol_id OGRSTSymbolParam

List of parameters for use with **OGRStyleSymbol** (p. **??**).

13.14.3.6 typedef enum ogr_style_tool_units_id OGRSTUnitId

List of units supported by OGRStyleTools.

13.14.4 Enumeration Type Documentation

13.14.4.1 enum ogr_style_tool_class_id

OGRStyleTool (p. **??**) derived class types (returned by GetType()).

13.14.4.2 enum ogr_style_tool_param_brush_id

List of parameters for use with **OGRStyleBrush** (p. **??**).

13.14.4.3 enum ogr_style_tool_param_label_id

List of parameters for use with **OGRStyleLabel** (p. **??**).

13.14.4.4 enum ogr_style_tool_param_pen_id

List of parameters for use with **OGRStylePen** (p. **??**).

13.14.4.5 enum ogr_style_tool_param_symbol_id

List of parameters for use with **OGRStyleSymbol** (p. **??**).

13.14.4.6 enum ogr_style_tool_units_id

List of units supported by OGRStyleTools.

13.14.4.7 enum OGRFieldSubType

List of field subtypes. A subtype represents a hint, a restriction of the main type, that is not strictly necessary to consult. This list is likely to be extended in the future ... avoid coding applications based on the assumption that all field types can be known. Most subtypes only make sense for a restricted set of main types.

Since

GDAL 2.0

Enumerator

OFSTNone No subtype. This is the default value

OFSTBoolean Boolean integer. Only valid for OFTInteger and OFTIntegerList.

OFSTInt16 Signed 16-bit integer. Only valid for OFTInteger and OFTIntegerList.

OFSTFloat32 Single precision (32 bit) floatint point. Only valid for OFTReal and OFTRealList.

13.14.4.8 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

OFTInteger64 Single 64bit integer

OFTInteger64List List of 64bit integers

13.14.4.9 enum OGRJustification

Display justification for field values.

13.14.4.10 enum OGRwkbGeometryType

List of well known binary geometry types. These are used within the BLOBs but are also returned from **OGR↩ Geometry::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

wkbCircularString one or more circular arc segments connected end to end, ISO SQL/MM Part 3. GDAL >= 2.0

wkbCompoundCurve sequence of contiguous curves, ISO SQL/MM Part 3. GDAL >= 2.0

wkbCurvePolygon planar surface, defined by 1 exterior boundary and zero or more interior boundaries, that are curves. ISO SQL/MM Part 3. GDAL >= 2.0

wkbMultiCurve GeometryCollection of Curves, ISO SQL/MM Part 3. GDAL >= 2.0

wkbMultiSurface GeometryCollection of Surfaces, ISO SQL/MM Part 3. GDAL >= 2.0

wkbNone non-standard, for pure attribute records

wkbLinearRing non-standard, just for createGeometry()

wkbCircularStringZ wkbCircularString with Z component. ISO SQL/MM Part 3. GDAL >= 2.0

wkbCompoundCurveZ wkbCompoundCurve with Z component. ISO SQL/MM Part 3. GDAL >= 2.0

wkbCurvePolygonZ wkbCurvePolygon with Z component. ISO SQL/MM Part 3. GDAL >= 2.0

wkbMultiCurveZ wkbMultiCurve with Z component. ISO SQL/MM Part 3. GDAL >= 2.0

wkbMultiSurfaceZ wkbMultiSurface with Z component. ISO SQL/MM Part 3. GDAL >= 2.0

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

13.14.4.11 enum OGRwkbVariant

Output variants of WKB we support.

99-402 was a short-lived extension to SFSQL 1.1 that used a high-bit flag to indicate the presence of Z coordiantes in a WKB geometry.

SQL/MM Part 3 and SFSQL 1.2 use offsets of 1000 (Z), 2000 (M) and 3000 (ZM) to indicate the present of higher dimensional coordinates in a WKB geometry. Reference: `09-009_Committee_Draft_↩ ISOIEC_CD_13249-3_SQLMM_Spatial.pdf`, ISO/IEC JTC 1/SC 32 N 1820, ISO/IEC CD 13249-3:201x(E), Date: 2009-01-16. The codes are also found in §8.2.3 of `OGC 06-103r4 "OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture", v1.2.1`

Enumerator

wkbVariantOldOgc Old-style 99-402 extended dimension (Z) WKB types

wkbVariantIso SFSQL 1.2 and ISO SQL/MM Part 3 extended dimension (Z&M) WKB types

wkbVariantPostGIS1 PostGIS 1.X has different codes for CurvePolygon, MultiCurve and MultiSurface

13.14.5 Function Documentation

13.14.5.1 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 primarly intented for external plugins.

Parameters

nVersionMajor	Major version to be tested against
nVersionMinor	Minor version to be tested against
pszCalling↩ Component↩ Name	If not NULL, in case of version mismatch, the method will issue a failure mentionning the name of the calling component.

13.14.5.2 OGRwkbGeometryType OGR_GT_Flatten (OGRwkbGeometryType *eType*)

Returns the 2D geometry type corresponding to the passed geometry type.

This function is intended to work with geometry types as old-style 99-402 extended dimension (Z) WKB types, as well as with newer SFSQL 1.2 and ISO SQL/MM Part 3 extended dimension (Z&M) WKB types.

Parameters

eType	Input geometry type

Returns

2D geometry type corresponding to the passed geometry type.

Since

GDAL 2.0

13.14.5.3 OGRwkbGeometryType OGR_GT_GetCollection (OGRwkbGeometryType *eType*)

Returns the collection type that can contain the passed geometry type.

Handled conversions are : wkbNone->wkbNone, wkbPoint -> wkbMultiPoint, wkbLineString->wkbMultiLine↩ String, wkbPolygon->wkbMultiPolygon, wkbCircularString->wkbMultiCurve, wkbCompoundCurve->wkbMulti↩ Curve, wkbCurvePolygon->wkbMultiSurface. In other cases, wkbUnknown is returned

Passed Z flag is preserved.

Parameters

eType	Input geometry type

Returns

the collection type that can contain the passed geometry type or wkbUnknown

Since

GDAL 2.0

References OGR_GT_IsCurve(), OGR_GT_IsSurface(), wkbFlatten, wkbHasZ, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbNone, wkbPoint, wkbPolygon, wkb↩ SetZ, and wkbUnknown.

Referenced by OGRGeometryFactory::forceTo().

13.14.5.4 OGRwkbGeometryType OGR_GT_GetCurve (OGRwkbGeometryType *eType*)

Returns the curve geometry type that can contain the passed geometry type.

Handled conversions are : wkbPolygon -> wkbCurvePolygon, wkbLineString->wkbCompoundCurve, wkbMulti↩ Polygon->wkbMultiSurface and wkbMultiLineString->wkbMultiCurve. In other cases, the passed geometry is returned.

Passed Z flag is preserved.

Parameters

eType	Input geometry type

Returns

the curve type that can contain the passed geometry type

Since

GDAL 2.0

References wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbHasZ, wkbLineString, wkbMultiCurve, wkb↩ MultiLineString, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, and wkbSetZ.

Referenced by OGRGeometryCollection::getCurveGeometry().

13.14.5.5 OGRwkbGeometryType OGR_GT_GetLinear (OGRwkbGeometryType *eType*)

Returns the non-curve geometry type that can contain the passed geometry type.

Handled conversions are : wkbCurvePolygon -> wkbPolygon, wkbCircularString->wkbLineString, wkb↩ CompoundCurve->wkbLineString, wkbMultiSurface->wkbMultiPolygon and wkbMultiCurve->wkbMultiLineString. In other cases, the passed geometry is returned.

Passed Z flag is preserved.

Parameters

eType	Input geometry type

Returns

the non-curve type that can contain the passed geometry type

Since

GDAL 2.0

References OGR_GT_IsCurve(), OGR_GT_IsSurface(), wkbFlatten, wkbHasZ, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, and wkbSetZ.

Referenced by OGRGeometryCollection::getLinearGeometry(), OGR_F_GetGeometryRef(), OGR_F_GetGeom↩ FieldRef(), OGR_FD_GetGeomType(), OGR_GFld_GetType(), and OGR_L_GetGeomType().

13.14.5.6 int OGR_GT_HasZ (OGRwkbGeometryType *eType*)

Return if the geometry type is a 3D geometry type.

Parameters

eType	Input geometry type

Returns

TRUE if the geometry type is a 3D geometry type.

Since

GDAL 2.0

Referenced by OGR_GT_SetZ().

13.14.5.7 int OGR_GT_IsCurve (OGRwkbGeometryType *eGeomType*)

Return if a geometry type is an instance of Curve.

Such geometry type are wkbLineString, wkbCircularString, wkbCompoundCurve and their 3D variant.

Parameters

eGeomType	the geometry type

Returns

TRUE if the geometry type is an instance of Curve

Since

GDAL 2.0

< Curve (abstract type). SF-SQL 1.2

References OGR_GT_IsSubClassOf(), and wkbCurve.

Referenced by OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToPolygon(), OGRGeometry↩Collection::get_Area(), OGRGeometryCollection::get_Length(), OGR_G_AddGeometry(), OGR_G_Add↩GeometryDirectly(), OGR_G_Area(), OGR_G_GetPointCount(), OGR_G_Length(), OGR_G_Value(), OGR_↩GT_GetCollection(), OGR_GT_GetLinear(), and OGRMergeGeometryTypesEx().

13.14.5.8 int OGR_GT_IsNonLinear (OGRwkbGeometryType *eGeomType*)

Return if a geometry type is a non-linear geometry type.

Such geometry type are wkbCircularString, wkbCompoundCurve, wkbCurvePolygon, wkbMultiCurve, wkbMulti↩Surface and their 3D variant.

Parameters

eGeomType	the geometry type

Returns

TRUE if the geometry type is a non-linear geometry type.

Since

GDAL 2.0

References wkbCircularString, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbMultiCurve, and wkb↩MultiSurface.

Referenced by OGR_F_GetGeometryRef(), OGR_F_GetGeomFieldRef(), OGR_FD_GetGeomType(), OGR_G↩Fld_GetType(), and OGR_L_GetGeomType().

13.14.5.9 int OGR_GT_IsSubClassOf (OGRwkbGeometryType *eType,* OGRwkbGeometryType *eSuperType*)

Returns if a type is a subclass of another one.

Parameters

eType	Type.
eSuperType	Super type

Returns

TRUE if eType is a subclass of eSuperType.

Since

GDAL 2.0

< Curve (abstract type). SF-SQL 1.2

< Surface (abstract type). SF-SQL 1.2

References wkbCircularString, wkbCompoundCurve, wkbCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, wkbSurface, and wkbUnknown.

Referenced by OGRGeometryFactory::forceTo(), OGRGeometryCollection::get_Area(), OGRGeometryCollection::get_Length(), OGR_G_AddGeometry(), OGR_G_AddGeometryDirectly(), OGR_G_Area(), OGR_G_GetGeometryCount(), OGR_G_GetGeometryRef(), OGR_G_Length(), OGR_G_RemoveGeometry(), OGR_GT_IsCurve(), OGR_GT_IsSurface(), and OGRMergeGeometryTypesEx().

13.14.5.10 int OGR_GT_IsSurface (OGRwkbGeometryType *eGeomType*)

Return if a geometry type is an instance of Surface.

Such geometry type are wkbCurvePolygon and wkbPolygon and their 3D variant.

Parameters

eGeomType	the geometry type

Returns

TRUE if the geometry type is an instance of Surface

Since

GDAL 2.0

< Surface (abstract type). SF-SQL 1.2

References OGR_GT_IsSubClassOf(), and wkbSurface.

Referenced by OGRGeometryCollection::get_Area(), OGR_G_Area(), OGR_GT_GetCollection(), and OGR_GT_GetLinear().

13.14.5.11 OGRwkbGeometryType OGR_GT_SetZ (OGRwkbGeometryType *eType*)

Returns the 3D geometry type corresponding to the passed geometry type.

Parameters

eType	Input geometry type

Returns

3D geometry type corresponding to the passed geometry type.

Since

GDAL 2.0

References OGR_GT_HasZ(), wkbGeometryCollection, wkbNone, and wkbUnknown.

13.14.5.12 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

eType	the geometry type.

Returns

internal human readable string, or NULL on failure.

References wkbCircularString, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbNone, wkbPoint, wkbPolygon, and wkbUnknown.

Referenced by OGRFeature::Validate().

13.14.5.13 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 encounted will establish the preliminary type.

Parameters

eMain	the first input geometry type.
eExtra	the second input geometry type.

Returns

the merged geometry type.

References OGRMergeGeometryTypesEx().

13.14.5.14 OGRwkbGeometryType OGRMergeGeometryTypesEx (OGRwkbGeometryType *eMain,* OGRwkbGeometryType *eExtra,* int *bAllowPromotingToCurves*)

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 encounted will establish the preliminary type.

If bAllowPromotingToCurves is set to TRUE, mixing Polygon and CurvePolygon will return CurvePolygon. Mixing LineString, CircularString, CompoundCurve will return CompoundCurve. Mixing MultiPolygon and MultiSurface will return MultiSurface. Mixing MultiCurve and MultiLineString will return MultiCurve.

Parameters

eMain	the first input geometry type.
eExtra	the second input geometry type.
bAllow↩ PromotingTo↩ Curves	determine if promotion to curve type must be done.

Returns

the merged geometry type.

Since

GDAL 2.0

References OGR_GT_IsCurve(), OGR_GT_IsSubClassOf(), wkbCompoundCurve, wkbFlatten, wkbGeometry↩ Collection, wkbNone, and wkbUnknown.

Referenced by OGRMergeGeometryTypes().

13.15 ogr_feature.h File Reference

```
#include "ogr_geometry.h"
#include "ogr_featurestyle.h"
#include "cpl_atomic_ops.h"
```

Classes

- class **OGRFieldDefn**
- class **OGRGeomFieldDefn**
- class **OGRFeatureDefn**
- class **OGRFeature**
- class **OGRFeatureQuery**

13.15.1 Detailed Description

Simple feature classes.

13.16 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**

13.16.1 Detailed Description

Simple feature style classes.

13.17 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 **papsz↩ StructuredQuery, 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.

13.17.1 Detailed Description

C API for geocoding client.

13.17.2 Function Documentation

13.17.2.1 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 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. 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 fellow 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: additionnal parameters for the GET request.

Parameters

hSession	the geocoding session handle.
pszQuery	the string to geocode.
papsz↩ StructuredQuery	unused for now. Must be NULL.
papszOptions	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 **OGR↩ GeocodeFreeResult()** (p. **??**).

Since

GDAL 1.10

References CPLError(), and CPLEscapeString().

13.17.2.2 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 occurence 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 occurence 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

papszOptions	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().

13.17.2.3 void OGRGeocodeDestroySession (OGRGeocodingSessionH *hSession*)

Destroys a session handle for geocoding requests.

Parameters

hSession	the handle to destroy.

Since

GDAL 1.10

References OGRReleaseDataSource().

Referenced by OGRGeocodeCreateSession().

13.17.2.4 void OGRGeocodeFreeResult (OGRLayerH *hLayer*)

Destroys the result of a geocoding request.

Parameters

hLayer	the layer returned by **OGRGeocode()** (p. **??**) or **OGRGeocodeReverse()** (p. **??**) to destroy.

Since

GDAL 1.10

13.17.2.5 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

hSession	the geocoding session handle.
dfLon	the longitude.
dfLat	the latitude.
papszOptions	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 **OGR↩ GeocodeFreeResult()** (p. **??**).

Since

GDAL 1.10

References CPLError().

13.18 ogr_geometry.h File Reference

```
#include "ogr_core.h"
#include "ogr_spatialref.h"
```

Classes

- class **OGRRawPoint**
- class **OGRGeometry**
- class **OGRPoint**
- class **OGRPointIterator**
- class **OGRCurve**
- class **OGRSimpleCurve**
- class **OGRLineString**
- class **OGRLinearRing**
- class **OGRCircularString**
- class **OGRCurveCollection**
- class **OGRCompoundCurve**
- class **OGRSurface**
- class **OGRCurvePolygon**
- class **OGRPolygon**
- class **OGRGeometryCollection**
- class **OGRMultiSurface**
- class **OGRMultiPolygon**
- class **OGRMultiPoint**
- class **OGRMultiCurve**
- class **OGRMultiLineString**
- class **OGRGeometryFactory**

13.18.1 Detailed Description

Simple feature geometry classes.

13.19 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)

13.19.1 Detailed Description

Coordinate systems services.

13.19.2 Function Documentation

13.19.2.1 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

poSource	source spatial reference system.
poTarget	target spatial reference system.

Returns

NULL on failure or a ready to use transformation object.

References CPLError().

Referenced by OCTNewCoordinateTransformation(), and OGRGeometry::transformTo().

13.20 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 **OSRImportFromOzi** (OGRSpatialReferenceH, const char *const *)
- 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 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, OGRSpatial↩ ReferenceH hHorizSRS, OGRSpatialReferenceH hVertSRS)

  Setup a compound coordinate system.

- OGRErr **OSRSetGeogCS** (OGRSpatialReferenceH hSRS, const char *pszGeogName, const char *psz↩ DatumName, const char *pszEllipsoidName, double dfSemiMajor, double dfInvFlattening, const char *psz↩ PMName, double dfPMOffset, const char *pszUnits, double dfConvertToRadians)

  Set geographic coordinate system.

- OGRErr **OSRSetVertCS** (OGRSpatialReferenceH hSRS, const char *pszVertCSName, const char *psz↩ VertDatumName, 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 *psz↩ Authority, 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, O↩ GRErr *)

  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 df↩ Default, 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, OGRAxis↩ Orientation *peOrientation)

  Fetch the orientation of one axis.
- OGRErr **OSRSetACEA** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenter↩ Lat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetAE** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double df↩ FalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetBonne** (OGRSpatialReferenceH hSRS, double dfStandardParallel, double dfCentral↩ Meridian, 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 df↩ FalseEasting, 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 **OSRSetEquirectangular** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenter↩ Long, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEquirectangular2** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenter↩ Long, 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 df↩ Azimuth, 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 df↩ Long1, 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 df↩ CenterLong, 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 dfCenter↩ Lat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMC** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double df↩ FalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMercator** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMollweide** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalse↩ Easting, double dfFalseNorthing)
- OGRErr **OSRSetNZMG** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetOS** (OGRSpatialReferenceH hSRS, double dfOriginLat, double dfCMeridian, double df↩ Scale, 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 df↩ Scale, 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 df↩ Scale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMVariant** (OGRSpatialReferenceH hSRS, const char ∗pszVariantName, double df↩ CenterLat, 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 df↩ FalseNorthing)
- OGRErr **OSRSetQSC** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong)
- double **OSRCalcInvFlattening** (double dfSemiMajor, double dfSemiMinor)

  Compute inverse flattening from semi-major and semi-minor axis.
- double **OSRCalcSemiMinorFromInvFlattening** (double dfSemiMajor, double dfInvFlattening)

  Compute semi-minor axis from semi-major axis and inverse flattening.
- void **OSRCleanup** (void)

  Cleanup cached SRS related memory.
- OGRCoordinateTransformationH CPL_STDCALL **OCTNewCoordinateTransformation** (OGRSpatial↩ ReferenceH hSourceSRS, OGRSpatialReferenceH hTargetSRS)
- void CPL_STDCALL **OCTDestroyCoordinateTransformation** (OGRCoordinateTransformationH)

  OGRCoordinateTransformation *(*p. **??***) destructor.*
- char ∗∗ **OPTGetProjectionMethods** (void)
- char ∗∗ **OPTGetParameterList** (const char ∗pszProjectionMethod, char ∗∗ppszUserName)
- int **OPTGetParameterInfo** (const char ∗pszProjectionMethod, const char ∗pszParameterName, char ∗∗ppszUserName, char ∗∗ppszType, double ∗pdfDefaultValue)

13.20.1 Detailed Description

C spatial reference system services and defines.

See also: **ogr_spatialref.h** (p. **??**)

13.20.2 Function Documentation

13.20.2.1 void CPL_STDCALL OCTDestroyCoordinateTransformation (OGRCoordinateTransformationH *hCT*)

OGRCoordinateTransformation (p. **??**) destructor.

This function is the same as **OGRCoordinateTransformation::DestroyCT()** (p. **??**)

Parameters

hCT	the object to delete

13.20.2.2 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

hSourceSRS	source spatial reference system.
hTargetSRS	target spatial reference system.

Returns

NULL on failure or a ready to use transformation object.

References OGRCreateCoordinateTransformation().

13.20.2.3 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

pszProjection↩ Method	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 "↩ Tranverse_Mercator".
pszParameter↩ Name	name of the parameter to fetch information about. This is the internal name such as "central↩ _meridian" (SRS_PP_CENTRAL_MERIDIAN).

ppszUserName	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.
ppszType	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.
pdfDefaultValue	location at which to put the default value for this parameter. The pointer may be NULL.

Returns

TRUE if parameter found, or FALSE otherwise.

References CPLAtof().

13.20.2.4 char** OPTGetParameterList (const char * *pszProjectionMethod,* char ** *ppszUserName*)

Fetch the parameters for a given projection method.

Parameters

pszProjection↩ Method	internal name of projection methods to fetch the parameters for, such as "Transverse_↩ Mercator" (SRS_PT_TRANSVERSE_MERCATOR).
ppszUserName	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().

13.20.2.5 char** OPTGetProjectionMethods (void)

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.

13.20.2.6 OGRErr OSRAutoIdentifyEPSG (OGRSpatialReferenceH *hSRS*)

Set EPSG authority info if possible.

This function is the same as **OGRSpatialReference::AutoIdentifyEPSG()** (p. **??**).

13.20.2.7 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().

13.20.2.8 double OSRCalcInvFlattening (double *dfSemiMajor,* double *dfSemiMinor*)

Compute inverse flattening from semi-major and semi-minor axis.

Parameters

dfSemiMajor	Semi-major axis length.
dfSemiMinor	Semi-minor axis length.

Returns

inverse flattening, or 0 if both axis are equal.

Since

GDAL 2.0

References CPLError().

Referenced by OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), and OGR↩SpatialReference::importFromUSGS().

13.20.2.9 double OSRCalcSemiMinorFromInvFlattening (double *dfSemiMajor,* double *dfInvFlattening*)

Compute semi-minor axis from semi-major axis and inverse flattening.

Parameters

dfSemiMajor	Semi-major axis length.
dfInvFlattening	Inverse flattening or 0 for sphere.

Returns

semi-minor axis

Since

GDAL 2.0

References CPLError().

Referenced by OGRSpatialReference::exportToPCI(), and OGRSpatialReference::GetSemiMinor().

13.20.2.10 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.

13.20.2.11 OGRSpatialReferenceH CPL_STDCALL OSRClone (OGRSpatialReferenceH *hSRS*)

Make a duplicate of this **OGRSpatialReference** (p. **??**).

This function is the same as **OGRSpatialReference::Clone()** (p. **??**)

13.20.2.12 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. **??**).

13.20.2.13 OGRErr OSRCopyGeogCSFrom (OGRSpatialReferenceH *hSRS,* OGRSpatialReferenceH *hSrcSRS*)

Copy GEOGCS from another **OGRSpatialReference** (p. **??**).

This function is the same as **OGRSpatialReference::CopyGeogCSFrom()** (p. **??**)

13.20.2.14 int OSRDereference (OGRSpatialReferenceH *hSRS*)

Decrements the reference count by one.

This function is the same as **OGRSpatialReference::Dereference()** (p. **??**)

13.20.2.15 void CPL_STDCALL OSRDestroySpatialReference (OGRSpatialReferenceH *hSRS*)

OGRSpatialReference (p. **??**) destructor.

This function is the same as **OGRSpatialReference::~OGRSpatialReference()** (p. **??**) and **OGRSpatialReference::DestroySpatialReference()** (p. **??**)

Parameters

hSRS	the object to delete

13.20.2.16 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().

13.20.2.17 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

13.20.2.18 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. **??**).

13.20.2.19 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. **??**)

13.20.2.20 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. **??**).

13.20.2.21 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. **??**).

13.20.2.22 OGRErr CPL_STDCALL OSRExportToProj4 (OGRSpatialReferenceH *hSRS*, char ** *ppszReturn*)

Export coordinate system in PROJ.4 format.

This function is the same as **OGRSpatialReference::exportToProj4()** (p. **??**).

13.20.2.23 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. **??**).

13.20.2.24 OGRErr CPL_STDCALL OSRExportToWkt (OGRSpatialReferenceH *hSRS*, char ** *ppszReturn*)

Convert this SRS into WKT format.

This function is the same as **OGRSpatialReference::exportToWkt()** (p. **??**).

13.20.2.25 OGRErr OSRExportToXML (OGRSpatialReferenceH *hSRS*, char ** *ppszRawXML*, const char * *pszDialect*)

Export coordinate system in XML format.

This function is the same as **OGRSpatialReference::exportToXML()** (p. **??**).

13.20.2.26 OGRErr OSRFixup (OGRSpatialReferenceH *hSRS*)

Fixup as needed.

This function is the same as **OGRSpatialReference::Fixup()** (p. **??**).

13.20.2.27 OGRErr OSRFixupOrdering (OGRSpatialReferenceH *hSRS*)

Correct parameter ordering to match CT Specification.

This function is the same as **OGRSpatialReference::FixupOrdering()** (p. **??**).

13.20.2.28 double OSRGetAngularUnits (OGRSpatialReferenceH *hSRS*, char ** *ppszName*)

Fetch angular geographic coordinate system units.

This function is the same as **OGRSpatialReference::GetAngularUnits()** (p. **??**)

13.20.2.29 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. **??**)

13.20.2.30 const char* OSRGetAuthorityCode (OGRSpatialReferenceH *hSRS*, const char * *pszTargetKey*)

Get the authority code for a node.

This function is the same as **OGRSpatialReference::GetAuthorityCode()** (p. **??**).

13.20.2.31 const char* OSRGetAuthorityName (OGRSpatialReferenceH *hSRS*, const char * *pszTargetKey*)

Get the authority name for a node.

This function is the same as **OGRSpatialReference::GetAuthorityName()** (p. **??**).

13.20.2.32 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. **??**)

13.20.2.33 double OSRGetInvFlattening (OGRSpatialReferenceH *hSRS*, OGRErr * *pnErr*)

Get spheroid inverse flattening.

This function is the same as **OGRSpatialReference::GetInvFlattening()** (p. **??**)

13.20.2.34 double OSRGetLinearUnits (OGRSpatialReferenceH *hSRS*, char ** *ppszName*)

Fetch linear projection units.

This function is the same as **OGRSpatialReference::GetLinearUnits()** (p. **??**)

13.20.2.35 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. **??**)

13.20.2.36 double OSRGetPrimeMeridian (OGRSpatialReferenceH *hSRS*, char ** *ppszName*)

Fetch prime meridian info.

This function is the same as **OGRSpatialReference::GetPrimeMeridian()** (p. **??**)

13.20.2.37 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. **??**)

13.20.2.38 double OSRGetSemiMajor (OGRSpatialReferenceH *hSRS*, OGRErr * *pnErr*)

Get spheroid semi major axis.

This function is the same as **OGRSpatialReference::GetSemiMajor()** (p. **??**)

13.20.2.39 double OSRGetSemiMinor (OGRSpatialReferenceH *hSRS,* OGRErr * *pnErr*)

Get spheroid semi minor axis.

This function is the same as **OGRSpatialReference::GetSemiMinor()** (p. **??**)

13.20.2.40 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

13.20.2.41 OGRErr OSRGetTOWGS84 (OGRSpatialReferenceH *hSRS,* double * *padfCoeff,* int *nCoeffCount*)

Fetch TOWGS84 parameters, if available.

This function is the same as **OGRSpatialReference::GetTOWGS84()** (p. **??**).

13.20.2.42 int OSRGetUTMZone (OGRSpatialReferenceH *hSRS,* int * *pbNorth*)

Get utm zone information.

This is the same as the C++ method **OGRSpatialReference::GetUTMZone()** (p. **??**)

13.20.2.43 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. **??**).

13.20.2.44 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. **??**).

13.20.2.45 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. **??**).

13.20.2.46 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. **??**)

13.20.2.47 OGRErr OSRImportFromMICoordSys (OGRSpatialReferenceH *hSRS,* const char * *pszCoordSys*)

Import Mapinfo style CoordSys definition.

This method is the equivalent of the C++ method **OGRSpatialReference::importFromMICoordSys** (p. **??**)

13.20.2.48 OGRErr OSRImportFromOzi (OGRSpatialReferenceH *hSRS,* const char *const * *papszLines*)

Import coordinate system from OziExplorer projection definition.

This function will import projection definition in style, used by OziExplorer software.

Note: another version of this function with a different signature existed in GDAL 1.X.

Parameters

hSRS	spatial reference object.
papszLines	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 2.0

13.20.2.49 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. **??**).

13.20.2.50 OGRErr OSRImportFromProj4 (OGRSpatialReferenceH *hSRS,* const char * *pszProj4*)

Import PROJ.4 coordinate string.

This function is the same as **OGRSpatialReference::importFromProj4()** (p. **??**).

13.20.2.51 OGRErr OSRImportFromUrl (OGRSpatialReferenceH *hSRS,* const char * *pszUrl*)

Set spatial reference from a URL.

This function is the same as **OGRSpatialReference::importFromUrl()** (p. **??**)

13.20.2.52 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. **??**).

13.20.2.53 OGRErr OSRImportFromWkt (OGRSpatialReferenceH *hSRS,* char ** *ppszInput*)

Import from WKT string.

This function is the same as **OGRSpatialReference::importFromWkt()** (p. **??**).

13.20.2.54 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. **??**).

13.20.2.55 int OSRIsCompound (OGRSpatialReferenceH *hSRS*)

Check if the coordinate system is compound.

This function is the same as **OGRSpatialReference::IsCompound()** (p. **??**).

13.20.2.56 int OSRIsGeocentric (OGRSpatialReferenceH *hSRS*)

Check if geocentric coordinate system.

This function is the same as **OGRSpatialReference::IsGeocentric()** (p. **??**).

Since

OGR 1.9.0

13.20.2.57 int OSRIsGeographic (OGRSpatialReferenceH *hSRS*)

Check if geographic coordinate system.

This function is the same as **OGRSpatialReference::IsGeographic()** (p. **??**).

13.20.2.58 int OSRIsLocal (OGRSpatialReferenceH *hSRS*)

Check if local coordinate system.

This function is the same as **OGRSpatialReference::IsLocal()** (p. **??**).

13.20.2.59 int OSRIsProjected (OGRSpatialReferenceH *hSRS*)

Check if projected coordinate system.

This function is the same as **OGRSpatialReference::IsProjected()** (p. **??**).

13.20.2.60 int OSRIsSame (OGRSpatialReferenceH *hSRS1*, OGRSpatialReferenceH *hSRS2*)

Do these two spatial references describe the same system ?

This function is the same as **OGRSpatialReference::IsSame()** (p. **??**).

13.20.2.61 int OSRIsSameGeogCS (OGRSpatialReferenceH *hSRS1*, OGRSpatialReferenceH *hSRS2*)

Do the GeogCS'es match?

This function is the same as **OGRSpatialReference::IsSameGeogCS()** (p. **??**).

13.20.2.62 int OSRIsSameVertCS (OGRSpatialReferenceH *hSRS1*, OGRSpatialReferenceH *hSRS2*)

Do the VertCS'es match?

This function is the same as **OGRSpatialReference::IsSameVertCS()** (p. **??**).

13.20.2.63 int OSRIsVertical (OGRSpatialReferenceH *hSRS*)

Check if vertical coordinate system.

This function is the same as **OGRSpatialReference::IsVertical()** (p. **??**).

Since

OGR 1.8.0

13.20.2.64 OGRErr OSRMorphFromESRI (OGRSpatialReferenceH *hSRS*)

Convert in place from ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphFromESRI()** (p. **??**)

13.20.2.65 OGRErr OSRMorphToESRI (OGRSpatialReferenceH *hSRS*)

Convert in place to ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphToESRI()** (p. **??**)

13.20.2.66 OGRSpatialReferenceH CPL_STDCALL OSRNewSpatialReference (const char * *pszWKT*)

Constructor.

This function is the same as OGRSpatialReference::OGRSpatialReference()

References OGRSpatialReference::importFromWkt().

13.20.2.67 int OSRReference (OGRSpatialReferenceH *hSRS*)

Increments the reference count by one.

This function is the same as **OGRSpatialReference::Reference()** (p. **??**)

13.20.2.68 void OSRRelease (OGRSpatialReferenceH *hSRS*)

Decrements the reference count by one, and destroy if zero.

This function is the same as **OGRSpatialReference::Release()** (p. **??**)

13.20.2.69 OGRErr OSRSetACEA (OGRSpatialReferenceH *hSRS,* double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Albers Conic Equal Area

13.20.2.70 OGRErr OSRSetAE (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Azimuthal Equidistant

13.20.2.71 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. **??**)

13.20.2.72 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. **??**)

13.20.2.73 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. **??**).

13.20.2.74 OGRErr OSRSetBonne (OGRSpatialReferenceH *hSRS*, double *dfStandardParallel*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Bonne

13.20.2.75 OGRErr OSRSetCEA (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cylindrical Equal Area

13.20.2.76 OGRErr OSRSetCompoundCS (OGRSpatialReferenceH *hSRS*, const char * *pszName*, OGRSpatialReferenceH *hHorizSRS*, OGRSpatialReferenceH *hVertSRS*)

Setup a compound coordinate system.

This function is the same as **OGRSpatialReference::SetCompoundCS()** (p. **??**)

13.20.2.77 OGRErr OSRSetCS (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cassini-Soldner

13.20.2.78 OGRErr OSRSetEC (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equidistant Conic

13.20.2.79 OGRErr OSRSetEckert (OGRSpatialReferenceH *hSRS*, int *nVariation*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert I-VI

13.20.2.80 OGRErr OSRSetEckertIV (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert IV

13.20.2.81 OGRErr OSRSetEckertVI (OGRSpatialReferenceH *hSRS,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Eckert VI

13.20.2.82 OGRErr OSRSetEquirectangular (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equirectangular

13.20.2.83 OGRErr OSRSetEquirectangular2 (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfPseudoStdParallel1,* double *dfFalseEasting,* double *dfFalseNorthing*)

Equirectangular generalized form

13.20.2.84 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. **??**)

13.20.2.85 OGRErr OSRSetGaussSchreiberTMercator (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Gauss Schreiber Transverse Mercator

13.20.2.86 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

13.20.2.87 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. **??**)

13.20.2.88 OGRErr OSRSetGEOS (OGRSpatialReferenceH *hSRS,* double *dfCentralMeridian,* double *dfSatelliteHeight,* double *dfFalseEasting,* double *dfFalseNorthing*)

GEOS - Geostationary Satellite View

13.20.2.89 OGRErr OSRSetGH (OGRSpatialReferenceH *hSRS,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Goode Homolosine

13.20.2.90 OGRErr OSRSetGnomonic (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gnomonic

13.20.2.91 OGRErr OSRSetGS (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gall Stereograpic

13.20.2.92 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. **??**)

13.20.2.93 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. **??**)

13.20.2.94 OGRErr OSRSetIGH (OGRSpatialReferenceH *hSRS*)

Interrupted Goode Homolosine

13.20.2.95 OGRErr OSRSetIWMPolyconic (OGRSpatialReferenceH *hSRS*, double *dfLat1*, double *dfLat2*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

International Map of the World Polyconic

13.20.2.96 OGRErr OSRSetKrovak (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfAzimuth*, double *dfPseudoStdParallelLat*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Krovak Oblique Conic Conformal

13.20.2.97 OGRErr OSRSetLAEA (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Azimuthal Equal-Area

13.20.2.98 OGRErr OSRSetLCC (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic

13.20.2.99 OGRErr OSRSetLCC1SP (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic 1SP

13.20.2.100 OGRErr OSRSetLCCB (OGRSpatialReferenceH *hSRS,* double *dfStdP1,* double *dfStdP2,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Lambert Conformal Conic (Belgium)

13.20.2.101 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. **??**)

13.20.2.102 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. **??**)

13.20.2.103 OGRErr OSRSetLocalCS (OGRSpatialReferenceH *hSRS,* const char * *pszName*)

Set the user visible LOCAL_CS name.

This function is the same as **OGRSpatialReference::SetLocalCS()** (p. **??**)

13.20.2.104 OGRErr OSRSetMC (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Miller Cylindrical

13.20.2.105 OGRErr OSRSetMercator (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Mercator

13.20.2.106 OGRErr OSRSetMollweide (OGRSpatialReferenceH *hSRS,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Mollweide

13.20.2.107 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. **??**)

13.20.2.108 OGRErr OSRSetNZMG (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

New Zealand Map Grid

13.20.2.109 OGRErr OSRSetOM (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfAzimuth,* double *dfRectToSkew,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Oblique Mercator (aka HOM (variant B)

13.20.2.110 OGRErr OSRSetOrthographic (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Orthographic

13.20.2.111 OGRErr OSRSetOS (OGRSpatialReferenceH *hSRS,* double *dfOriginLat,* double *dfCMeridian,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Oblique Stereographic

13.20.2.112 OGRErr OSRSetPolyconic (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Polyconic

13.20.2.113 OGRErr OSRSetProjCS (OGRSpatialReferenceH *hSRS,* const char * *pszName*)

Set the user visible PROJCS name.

This function is the same as **OGRSpatialReference::SetProjCS()** (p. **??**)

13.20.2.114 OGRErr OSRSetProjection (OGRSpatialReferenceH *hSRS,* const char * *pszProjection*)

Set a projection name.

This function is the same as **OGRSpatialReference::SetProjection()** (p. **??**)

13.20.2.115 OGRErr OSRSetProjParm (OGRSpatialReferenceH *hSRS,* const char * *pszParmName,* double *dfValue*)

Set a projection parameter value.

This function is the same as **OGRSpatialReference::SetProjParm()** (p. **??**)

13.20.2.116 OGRErr OSRSetPS (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Polar Stereographic

13.20.2.117 OGRErr OSRSetQSC (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong*)

Quadrilateralized Spherical Cube

13.20.2.118 OGRErr OSRSetRobinson (OGRSpatialReferenceH *hSRS,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Robinson

13.20.2.119 OGRErr OSRSetSinusoidal (OGRSpatialReferenceH *hSRS,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Sinusoidal

13.20.2.120 OGRErr OSRSetSOC (OGRSpatialReferenceH *hSRS,* double *dfLatitudeOfOrigin,* double *dfCentralMeridian,* double *dfFalseEasting,* double *dfFalseNorthing*)

Swiss Oblique Cylindrical

13.20.2.121 OGRErr OSRSetStatePlane (OGRSpatialReferenceH *hSRS,* int *nZone,* int *bNAD83*)

Set State Plane projection definition.

This function is the same as **OGRSpatialReference::SetStatePlane()** (p. **??**).

13.20.2.122 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. **??**).

13.20.2.123 OGRErr OSRSetStereographic (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfScale,* double *dfFalseEasting,* double *dfFalseNorthing*)

Stereographic

13.20.2.124 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

13.20.2.125 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. **??**).

13.20.2.126 OGRErr OSRSetTMG (OGRSpatialReferenceH *hSRS,* double *dfCenterLat,* double *dfCenterLong,* double *dfFalseEasting,* double *dfFalseNorthing*)

Tunesia Mining Grid

13.20.2.127 OGRErr OSRSetTMSO (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator (South Oriented)

13.20.2.128 OGRErr OSRSetTMVariant (OGRSpatialReferenceH *hSRS*, const char * *pszVariantName*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator variant

13.20.2.129 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. **??**).

13.20.2.130 OGRErr OSRSetUTM (OGRSpatialReferenceH *hSRS*, int *nZone*, int *bNorth*)

Set UTM projection definition.

This is the same as the C++ method **OGRSpatialReference::SetUTM()** (p. **??**)

13.20.2.131 OGRErr OSRSetVDG (OGRSpatialReferenceH *hSRS*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

VanDerGrinten

13.20.2.132 OGRErr OSRSetVertCS (OGRSpatialReferenceH *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

13.20.2.133 OGRErr OSRSetWagner (OGRSpatialReferenceH *hSRS*, int *nVariation*, double *dfFalseEasting*, double *dfFalseNorthing*)

Wagner I – VII

13.20.2.134 OGRErr OSRSetWellKnownGeogCS (OGRSpatialReferenceH *hSRS*, const char * *pszName*)

Set a GeogCS based on well known name.

This function is the same as **OGRSpatialReference::SetWellKnownGeogCS()** (p. **??**)

13.20.2.135 OGRErr OSRStripCTParms (OGRSpatialReferenceH *hSRS*)

Strip OGC CT Parameters.

This function is the same as **OGRSpatialReference::StripCTParms()** (p. **??**).

13.20.2.136 OGRErr OSRValidate (OGRSpatialReferenceH *hSRS*)

Validate SRS tokens.

This function is the same as the C++ method **OGRSpatialReference::Validate()** (p. **??**).

13.21 ograpispy.h File Reference

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

13.21.1 Detailed Description

OGR C API spy.

If GDAL is compiled with OGRAPISPY_ENABLED defined (which is the case for a DEBUG build), a mechanism to trace calls to the OGR *C* API is available (calls to the C++ API will not be traced)

Provided there is compile-time support, the mechanism must also be enabled at runtime by setting the OGR_API_SPY_FILE configuration option to a file where the calls to the OGR C API will be dumped (stdout and stderr are recognized as special strings to name the standard output and error files). The traced calls are outputted as a OGR Python script.

Only calls that may have side-effects to the behaviour of drivers are traced.

If a file-based datasource is open in update mode, a snapshot of its initial state is stored in a 'snapshot' directory, and then a copy of it is made as the working datasource. That way, the generated script can be executed in a reproducible way. The path for snapshots is the current working directory by default, and can be changed by setting the OGR_API_SPY_SNAPSHOT_PATH configuration option. If it is set to NO, the snapshot feature will be disabled. The reliability of snapshoting relies on if the dataset correctly implements GetFileList() (for multi-file datasources)

Since

GDAL 2.0

13.22 ogrsf_frmts.h File Reference

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

Classes

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

Functions

- void **OGRRegisterAll** ()

  Register all drivers.

13.22.1 Detailed Description

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

13.22.2 Function Documentation

13.22.2.1 void OGRRegisterAll (void)

Register all drivers.

Deprecated Use GDALAllRegister() in GDAL 2.0