

OGR



# Contents



# Chapter 1

## 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.

### 1.1 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.

With GDAL 2.3 and C++11:

```

for( auto& poFeature: poLayer )
{

```

With GDAL 2.3 and C:

```

OGR_FOR_EACH_FEATURE_BEGIN(hFeature, hLayer)
{

```

If using older GDAL versions, 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.

With GDAL < 2.3 and C++ :

```
OGRFeature *poFeature;

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

With GDAL < 2.3 and 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.

With GDAL 2.3 and C++11:

```
for( auto&& oField: *poFeature )
{
    switch( oField.GetType() )
    {
        case OFTInteger:
            printf( "%d,", oField.GetInteger() );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB ", ", oField.GetInteger64() );
            break;
        case OFTReal:
            printf( "%.3f,", oField.GetDouble() );
            break;
        case OFTString:
            printf( "%s,", oField.GetString() );
            break;
        default:
            printf( "%s,", oField.GetAsString() );
            break;
    }
}
```

With GDAL < 2.3 and C++ :

```
OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
for( int iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
{
    OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

    switch( poFieldDefn->GetType() )
    {
        case OFTInteger:
            printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB ", ", poFeature->GetFieldAsInteger64( iField ) );
            break;
        case OFTReal:
            printf( "%.3f,", poFeature->GetFieldAsDouble( iField ) );
            break;
        case OFTString:
            printf( "%s,", poFeature->GetFieldAsString( iField ) );
            break;
        default:
            printf( "%s,", poFeature->GetFieldAsString( iField ) );
            break;
    }
}
```

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 );

    switch( OGR_Fld_GetType(hFieldDefn) )
    {
        case OFTInteger:
            printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB " ", OGR_F_GetFieldAsInteger64( hFeature, iField ) );
            break;
        case OFTReal:
            printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField ) );
            break;
        case OFTString:
            printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField ) );
            break;
        default:
            printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField ) );
            break;
    }
}

```

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 )
{
    #if GDAL_VERSION_NUM >= GDAL_COMPUTE_VERSION(2,3,0)
        OGRPoint *poPoint = poGeometry->toPoint();
    #else
        OGRPoint *poPoint = (OGRPoint *) poGeometry;
    #endif

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

```

In C :

```

OGRGeometryH hGeometry;

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

```



The **wkbFlatten()** (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 = poGeometry->toPoint();
        OGRPoint *poPoint = (OGRPoint *) poGeometry;
    }
    printf( "%.3f,%.3f\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,%.3f\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 delete the return geometry.

With GDAL 2.3 and C++11, the looping over features is simply terminated by a closing curly bracket.

```
}

```

With GDAL 2.3 and C, the looping over features is simply terminated by the following.

```
}
OGR_FOR_EACH_FEATURE_END(hFeature)

```

For GDAL < 2.3, as 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 issues.

In C/C++ :

```
GDALClose( poDS );
}

```

All together our program looks like this.

With GDAL 2.3 and C++11 :

```
#include "ogr_sfrmts.h"

int main()
{
    GDALAllRegister();

    GDALDatasetUniquePtr poDS(GDALDataset::Open( "point.shp", GDAL_OF_VECTOR));
    if( poDS == nullptr )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    for( const OGRLayer* poLayer: poDS->GetLayers() )
    {
        for( const auto& poFeature: *poLayer )
        {
            for( const auto& oField: *poFeature )
            {
                switch( oField.GetType() )
                {
                    case OFTInteger:
                        printf( "%d,", oField.GetInteger() );
                        break;
                    case OFTInteger64:
                        printf( CPL_FRMT_GIB " ", oField.GetInteger64() );
                        break;
                    case OFTReal:

```

```

        printf( "%.3f,", oField.GetDouble() );
        break;
    case OFTString:
        printf( "%s,", oField.GetString() );
        break;
    default:
        printf( "%s,", oField.GetAsString() );
        break;
    }
}

const OGRGeometry *poGeometry = poFeature->GetGeometryRef();
if( poGeometry != nullptr
    && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
{
    const OGRPoint *poPoint = poGeometry->toPoint();

    printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );
}
else
{
    printf( "no point geometry\n" );
}
}
}
return 0;
}

```

In C++ :

```

#include "ogrsgf_frmts.h"

int main()
{
    GDALAllRegister();

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

    OGRLayer *poLayer = poDS->GetLayerByName( "point" );
    OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();

    poLayer->ResetReading();
    OGRFeature *poFeature;
    while( (poFeature = poLayer->GetNextFeature()) != NULL )
    {
        for( int iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
        {
            OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

            switch( poFieldDefn->GetType() )
            {
                case OFTInteger:
                    printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
                    break;
                case OFTInteger64:
                    printf( CPL_FRMT_GIB "%d,", poFeature->GetFieldAsInteger64( iField ) );
                    break;
                case OFTReal:
                    printf( "%.3f,", poFeature->GetFieldAsDouble( iField ) );
                    break;
                case OFTString:
                    printf( "%s,", poFeature->GetFieldAsString( iField ) );
                    break;
                default:
                    printf( "%s,", poFeature->GetFieldAsString( iField ) );
                    break;
            }
        }

        OGRGeometry *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;
    OGRFeatureDefnH hFDefn;

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

    hLayer = GDALDatasetGetLayerByName( hDS, "point" );
    hFDefn = OGR_L_GetLayerDefn(hLayer);

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

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

            switch( OGR_Fld_GetType( hFieldDefn ) )
            {
                case OFTInteger:
                    printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
                    break;
                case OFTInteger64:
                    printf( CPL_FRMT_GIB " ", OGR_F_GetFieldAsInteger64( hFeature, iField ) );
                    break;
                case OFTReal:
                    printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField ) );
                    break;
                case OFTString:
                    printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField ) );
                    break;
                default:
                    printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField ) );
                    break;
            }
        }

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

        OGR_F_Destroy( hFeature );
    }

    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

```

## 1.2 Writing To OGR

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

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

In C++ :

```

#include "ogr_sfrmts.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    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 snippet 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 "ogr_sfrmts.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 2

# 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 modeled 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 modeled on those from the interface specifications.

### 2.1 Class Overview

- **Geometry** (`ogr_geometry.h`): The geometry classes (**OGRGeometry** (p. ??), etc) encapsulate the OpenGIS model vector data as well as providing some geometry operations, and translation to/from well known binary and text format. A geometry includes a spatial reference system (projection).
- **Spatial Reference** (`ogr_spatialref.h`): An **OGRSpatialReference** (p. ??) encapsulates the definition of a projection and datum.
- **Feature** (`ogr_feature.h`): The **OGRFeature** (p. ??) encapsulates the definition of a whole feature, that is a geometry and a set of attributes.
- **Feature Class Definition** (`ogr_feature.h`): The **OGRFeatureDefn** (p. ??) class captures the schema (set of field definitions) for a group of related features (normally a whole layer).
- **Layer** (`ogr_sfcmts.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**.

## 2.2 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 modeled in the simple features abstract model and SFCOM are not modeled 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.

### 2.2.1 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.

## 2.3 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.



## 2.4 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 modeled 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).

## 2.5 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.

## 2.6 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.

## 2.7 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 instantiated 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 3

# OGR Driver Implementation Tutorial

### 3.1 Overall Approach

In general new formats are added to OGR by implementing format specific drivers with instantiating a GDALDriver and subclasses of GDALDataset and **OGRLayer** (p. ??). The GDALDriver instance is registered with the GDALDriverManager 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.

### 3.2 Contents

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

### 3.3 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()
{
    if( GDALGetDriverByName("SPF") != NULL )
        return;

    GDALDriver *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 data-sources 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->eAccess == GA_Update) )
    {
        delete poDS;
        return NULL;
    }

    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.

### 3.4 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 `TestCapability()` 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; }
    OGRSPFLayer      *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 its work off to the OGRSPFLayer constructor if it believes the file is of the desired format.

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

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

```

int OGRSPFDataSource::Open( const char *pszFilename, int bUpdate )
{
    if( bUpdate )
    {
        CPLError( CE_Failure, CPLE_OpenFailed,
            "Update access not supported by the SPF driver." );
        return FALSE;
    }

    // Create a corresponding layer.
    nLayers = 1;
    papoLayers = static_cast<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.

```

OGRSPFLayer *OGRSPFDataSource::GetLayer( int iLayer )
{
    if( iLayer < 0 || iLayer >= nLayers )
        return NULL;

    return papoLayers[iLayer];
}

```

### 3.5 Read Only Layer

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

```
class OGRSPFLayer : public OGRLayer
{
    OGRFeatureDefn      *poFeatureDefn;
    FILE                *fp;
    int                 nNextFID;

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

    void                ResetReading();
    OGRFeature *        GetNextFeature();

    OGRFeatureDefn *    GetLayerDefn() { return poFeatureDefn; }

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

The layer constructor is responsible for initialization. The most important initialization is setting up the **OGRFeatureDefn** (p. ??) for the layer. This defines the list of fields and their types, the geometry type and the coordinate system for the layer. In the SPF format the set of fields is fixed - a single string field and we have no coordinate system info to set.

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

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

    poFeatureDefn = new OGRFeatureDefn(CPLGetBasename(pszFilename));
    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 = CPLReadLineL(fp);

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

        const double dfX = atof(pszLine);

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

        const double dfY = atof(pszLine);

        pszLine = strstr(pszLine, "|");

        const char *pszName = NULL;
        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 **OGRLayer** (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 its feature id will result in many calls to `GetNextFeature()` until 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 4

# 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.

### 4.1 SELECT

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

```
SELECT * FROM polylayer
```

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

This slightly more sophisticated form still pulls all features from the layer but the schema will only contain the `EAS_ID` and `PROP_VALUE` attributes. Any other attributes would be discarded.

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

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

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

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

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

### 4.1.1 General syntax

The general syntax of a SELECT statement is:

```
SELECT [fields] FROM layer_name [JOIN ...] [WHERE ...] [ORDER BY ...] [LIMIT ...] [OFFSET ...]
```

### 4.1.2 Field List Operators

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

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

```
SELECT DISTINCT areacode FROM polylayer
```

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

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

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

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

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

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

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

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

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

#### 4.1.2.1 Functions

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

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

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

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'
```

#### 4.1.2.2 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
```

#### 4.1.2.3 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]`.

#### 4.1.2.4 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 surrounded with double-quote characters. e.g. `WHERE "from" = 5`.

### 4.1.3 WHERE

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

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

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

The value argument to the **LIKE** operator is a pattern against which the value string is matched. In this pattern percent (%) matches any number of characters, and underscore ( \_ ) matches any one character. An optional `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
NON 1P0	%NON%	Yes
L4C 5E2	%NON%	No

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

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

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

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

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

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

#### 4.1.4 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 >=.

#### 4.1.5 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.

### 4.1.6 LIMIT and OFFSET

Starting with GDAL 2.2, the **LIMIT** clause can be used to limit the number of features returned. For example

```
SELECT * FROM poly LIMIT 5
```

The **OFFSET** clause can be used to skip the first features of the result set. The value after OFFSET is the number of features skipped. For example, to skip the first 3 features from the result set:

```
SELECT * FROM poly OFFSET 3
```

Both clauses can be combined:

```
SELECT * FROM poly LIMIT 5 OFFSET 3
```

### 4.1.7 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 mentioned 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.

#### 4.1.8 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.

## 4.2 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] [()]]*
```

### 4.2.1 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.

## 4.3 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.

### 4.3.1 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
```

### 4.3.2 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'
```

### 4.3.3 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%'
```



### 4.3.4 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
```

### 4.3.5 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%'
```

## 4.4 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
```

### 4.4.1 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 they will accelerate is a simple "field = value" query.

## 4.5 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
```

## 4.6 ALTER TABLE

(OGR  $\geq$  1.9.0)

The following OGR SQL ALTER TABLE commands can be used.

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

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

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

## 4.7 DROP TABLE

(OGR  $\geq$  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
```

## 4.8 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::ReleaseResultSet()` method when no longer needed. Failure to release it before the datasource is destroyed may result in a crash.

## 4.9 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 5

# SQLite SQL dialect

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

The SQLite dialect may be used with any OGR datasource, like the OGR SQL dialect. It is available through the `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/UPDATE/DELETE statements can also be run.

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

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

### 5.1 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.

### 5.1.1 Delimited identifiers

If names of layers or attributes are reserved keywords in SQL like 'FROM' or they begin with a number or underscore they must be handled as "delimited identifiers" and enclosed between double quotation marks in queries. Double quotes can be used even when they are not strictly needed.

```
SELECT "p"."geometry", "p"."FROM", "p"."3D" FROM "poly" p
```

When SQL statements are used in the command shell and the statement itself is put between double quotes, the internal double quotes must be escaped with \

```
ogrinfo p.shp -sql "SELECT geometry \"FROM\", \"3D\" FROM p"
```

### 5.1.2 Geometry field

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

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

```
SELECT EAS_ID, GEOMETRY FROM poly
```

returns:

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

```
SELECT * FROM poly
```

returns:

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

### 5.1.3 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%'
```

### 5.1.4 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
```

### 5.1.5 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')
```

### 5.1.6 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))
```

### 5.1.7 OGR compression functions

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

**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. ??).

### 5.1.7.1 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'
```

### 5.1.8 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.

### 5.1.9 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 6

# OGR Projections Tutorial

### 6.1 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 modeled 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.

You may also consult the [OGC WKT Coordinate System Issues page](#).

### 6.2 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 its 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.

## 6.3 Defining a Projected Coordinate System

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

```
OGRSpatialReference oSRS;

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

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

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

```
PROJCS["UTM 17 (WGS84) in northern hemisphere.",
  GEOGCS["WGS 84",
    DATUM["WGS_1984",
      SPHEROID["WGS 84",6378137,298.257223563,
        AUTHORITY["EPSG",7030]],
      TOWGS84[0,0,0,0,0,0,0],
      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()`.

## 6.4 Querying Coordinate System

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

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

## 6.5 Coordinate Transformation

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

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

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

poCT = OGRCreateCoordinateTransformation( &oSourceSRS,
                                          &oTargetSRS );

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

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

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

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

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

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

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

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

poLatLong = oUTM.CloneGeogCS();

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

...

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

## 6.6 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):

```

## 6.7 Internal Implementation

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

## Chapter 7

# Deprecated List

### Member **CPL\_LSBINT16PTR** (p. ??) (x)

Use rather CPL\_LSBSINT16PTR or CPL\_LSBUINT16PTR for explicit signedness.

### Member **CPL\_LSBINT32PTR** (p. ??) (x)

Use rather CPL\_LSBSINT32PTR or CPL\_LSBUINT32PTR for explicit signedness.

### Member **OGR\_Dr\_CopyDataSource** (p. ??) (OGRSFDriverH, OGRDataSourceH, const char \*, char \*\*) CPL\_WARN\_UNUSED\_RESULT ↩

Use GDALCreateCopy() in GDAL 2.0

### Member **OGR\_Dr\_CreateDataSource** (p. ??) (OGRSFDriverH, const char \*, char \*\*) CPL\_WARN\_UNUSED\_RESULT ↩

Use GDALCreate() in GDAL 2.0

### Member **OGR\_Dr\_DeleteDataSource** (p. ??) (OGRSFDriverH, const char \*)

Use GDALDeleteDataset() in GDAL 2

### Member **OGR\_Dr\_Open** (p. ??) (OGRSFDriverH, const char \*, int) CPL\_WARN\_UNUSED\_RESULT

Use GDALOpenEx() in GDAL 2.0

### Member **OGR\_Dr\_TestCapability** (p. ??) (OGRSFDriverH, const char \*)

Use GDALGetMetadataItem(hDriver, GDAL\_DCAP\_CREATE) in GDAL 2.0

### Member **OGR\_DS\_CopyLayer** (p. ??) (OGRDataSourceH, OGRLayerH, const char \*, char \*\*)

Use GDALDatasetCopyLayer() in GDAL 2.0

### Member **OGR\_DS\_CreateLayer** (p. ??) (OGRDataSourceH, const char \*, OGRSpatialReferenceH, OGRwkbGeometryType, char \*\*) ↩

Use GDALDatasetCreateLayer() in GDAL 2.0

### Member **OGR\_DS\_DeleteLayer** (p. ??) (OGRDataSourceH, int)

Use GDALDatasetDeleteLayer() in GDAL 2.0

### Member **OGR\_DS\_Destroy** (p. ??) (OGRDataSourceH)

Use GDALClose() in GDAL 2.0

### Member **OGR\_DS\_ExecuteSQL** (p. ??) (OGRDataSourceH, const char \*, OGRGeometryH, const char \*)

Use GDALDatasetExecuteSQL() in GDAL 2.0

### Member **OGR\_DS\_GetDriver** (p. ??) (OGRDataSourceH)

Use GDALGetDatasetDriver() in GDAL 2.0

### Member **OGR\_DS\_GetLayer** (p. ??) (OGRDataSourceH, int)

Use GDALDatasetGetLayer() in GDAL 2.0

**Member OGR\_DS\_GetLayerByName** (p. ??) (OGRDataSourceH, const char \*)

Use GDALDatasetGetLayerByName() in GDAL 2.0

**Member OGR\_DS\_GetLayerCount** (p. ??) (OGRDataSourceH)

Use GDALDatasetGetLayerCount() in GDAL 2.0

**Member OGR\_DS\_GetName** (p. ??) (OGRDataSourceH)

Use GDALGetDescription() in GDAL 2.0

**Member OGR\_DS\_ReleaseResultSet** (p. ??) (OGRDataSourceH, OGRLayerH)

Use GDALDatasetReleaseResultSet() in GDAL 2.0

**Member OGR\_DS\_TestCapability** (p. ??) (OGRDataSourceH, const char \*)

Use GDALDatasetTestCapability() in GDAL 2.0

**Member OGR\_G\_GetCoordinateDimension** (p. ??) (OGRGeometryH)

use OGR\_G\_CoordinateDimension() (p. ??), OGR\_G\_Is3D() (p. ??), or OGR\_G\_IsMeasured() (p. ??).

**Member OGR\_G\_SetCoordinateDimension** (p. ??) (OGRGeometryH, int)

use OGR\_G\_Set3D() (p. ??) or OGR\_G\_SetMeasured() (p. ??).

**Member OGR\_SRSNode::importFromWkt** (p. ??) (char \*\*)

GDAL 2.3. Use importFromWkt(const char\*\*) (p. ??) instead.

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

**Member OGRGeometry::getCoordinateDimension** (p. ??) () const

use CoordinateDimension() (p. ??).

**Member OGRGeometry::importFromWkt** (p. ??) (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

in GDAL 2.3

**Member OGRGeometry::setCoordinateDimension** (p. ??) (int nDimension)

use set3D() (p. ??) or setMeasured() (p. ??).

**Member OGRGeometryFactory::createFromWkt** (p. ??) (char \*\*ppszInput, OGRSpatialReference (p. ??) \*poSRS, OGRGeometry (p. ??) \*\*ppoGeom)

in GDAL 2.3

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

Use GDALGetDriver() in GDAL 2.0

**Member OGRGetDriverByName** (p. ??) (const char \*)

Use GDALGetDriverByName() in GDAL 2.0

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

Use GDALGetDriverCount() in GDAL 2.0

**Member OGROpen** (p. ??) (const char \*, int, OGRSFDriverH \*) CPL\_WARN\_UNUSED\_RESULT

Use GDALOpenEx() in GDAL 2.0

**Member OGROpenShared** (p. ??) (const char \*, int, OGRSFDriverH \*) CPL\_WARN\_UNUSED\_RESULT

Use GDALOpenEx() in GDAL 2.0

**Member OGRRegisterAll** (p. ??) (void)

Use GDALAllRegister() in GDAL 2.0

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

Use GDALClose() in GDAL 2.0

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



---

**Class OGRSFDriverRegistrar (p. ??)****Member OGRSpatialReference::GetAngularUnits (p. ??) (const char \*\*=nullptr) const**

GDAL 2.3.0. Use **GetAngularUnits(const char\*\*) const** (p. ??).

**Member OGRSpatialReference::GetLinearUnits (p. ??) (char \*\*) const CPL\_WARN\_DEPRECATED("Use GetLinearUnits( const char\*\*) instead")**

GDAL 2.3.0. Use **GetLinearUnits(const char\*\*) const** (p. ??).

**Member OGRSpatialReference::GetPrimeMeridian (p. ??) (const char \*\*=nullptr) const**

GDAL 2.3.0. Use **GetPrimeMeridian(const char\*\*) const** (p. ??).

**Member OGRSpatialReference::GetTargetLinearUnits (p. ??) (const char \*pszTargetKey, const char \*\*ppszRetName=nullptr) const**

GDAL 2.3.0. Use **GetTargetLinearUnits(const char\*, const char\*\*) const** (p. ??).

**Member OGRSpatialReference::importFromWkt (p. ??) (char \*\*)**

GDAL 2.3. Use **importFromWkt(const char\*\*) (p. ??)** or **importFromWkt(const char\*) (p. ??)**

**Member OGRSpatialReference::~OGRSpatialReference (p. ??) ()****Member wkb25DBit (p. ??)**

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



## Chapter 8

# Hierarchical Index

### 8.1 Class Hierarchy

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

_CPLHashSet . . . . .	??
_CPLList . . . . .	??
_CPLLock . . . . .	??
_CPLQuadTree . . . . .	??
_CPLSpawnedProcess . . . . .	??
_CPLSpinLock . . . . .	??
_MutexLinkedElt . . . . .	??
_OGRGeocodingSessionHS . . . . .	??
_QuadTreeNode . . . . .	??
_sPolyExtended . . . . .	??
OGRFeature::ConstFieldIterator . . . . .	??
CPL_SHA1Context . . . . .	??
CPLErrorContext . . . . .	??
CPLHTTPErrorBuffer . . . . .	??
CPLHTTPResult . . . . .	??
CPLHTTPResultWithLimit . . . . .	??
CPLJSONDocument . . . . .	??
CPLJSONObject . . . . .	??
CPLJSONArray . . . . .	??
CPLLockHolder . . . . .	??
CPLMimePart . . . . .	??
CPLMutexHolder . . . . .	??
CPLODBCDriverInstaller . . . . .	??
CPLODBCSession . . . . .	??
CPLODBCStatement . . . . .	??
CPLRectObj . . . . .	??
CPLSharedFileInfo . . . . .	??
CPLSharedFileInfoExtra . . . . .	??
CPLStdCallThreadInfo . . . . .	??
CPLStringList . . . . .	??
CPLVirtualMem . . . . .	??
CPLWorkerThreadPool . . . . .	??
CPLXMLNode . . . . .	??
CPLZip . . . . .	??
ctb . . . . .	??

curfile_info . . . . .	??
CurlProcessData . . . . .	??
DefaultCSVFileNameTLS . . . . .	??
errHandler . . . . .	??
exception	
OGRFeature::FieldNotFoundException . . . . .	??
OGRFeature::FieldValue . . . . .	??
file_in_zip_read_info_s . . . . .	??
FindFileTLS . . . . .	??
GDALDataset	
OGRDataSource . . . . .	??
OGRDataSourceWithTransaction . . . . .	??
GDALDriver	
OGRSFDriver . . . . .	??
GDALMajorObject	
OGRLayer . . . . .	??
GDALScaledProgressInfo . . . . .	??
GOA2Manager . . . . .	??
IOGRConstGeometryVisitor . . . . .	??
OGRDefaultConstGeometryVisitor . . . . .	??
IOGRGeometryVisitor . . . . .	??
OGRDefaultGeometryVisitor . . . . .	??
IOGRTransactionBehaviour . . . . .	??
LinearUnitsStruct . . . . .	??
linkedlist_data_s . . . . .	??
linkedlist_datablock_internal_s . . . . .	??
OGR_SRSNode . . . . .	??
OGRAttrIndex	
OGRMIAAttrIndex . . . . .	??
OGRCoordinateTransformation . . . . .	??
OGRProj4CT . . . . .	??
OGRFeature . . . . .	??
OGRFeatureDefn . . . . .	??
OGRField . . . . .	??
OGRFieldDefn . . . . .	??
OGRGeometry . . . . .	??
OGRCurve . . . . .	??
OGRCompoundCurve . . . . .	??
OGRSimpleCurve . . . . .	??
OGRCircularString . . . . .	??
OGRLineString . . . . .	??
OGRLinearRing . . . . .	??
OGRGeometryCollection . . . . .	??
OGRMultiCurve . . . . .	??
OGRMultiLineString . . . . .	??
OGRMultiPoint . . . . .	??
OGRMultiSurface . . . . .	??
OGRMultiPolygon . . . . .	??
OGRPoint . . . . .	??
OGRSurface . . . . .	??
OGRCurvePolygon . . . . .	??
OGRPolygon . . . . .	??
OGRTriangle . . . . .	??
OGRPolyhedralSurface . . . . .	??
OGRTriangulatedSurface . . . . .	??
OGRGeometryFactory . . . . .	??

OGRGeomFieldDefn . . . . .	??
OGRLayerAttrIndex	
OGRMILayerAttrIndex . . . . .	??
OGRLayerDecorator	
OGRLayerWithTransaction . . . . .	??
OGRPointIterator . . . . .	??
OGRCompoundCurvePointIterator . . . . .	??
OGRSimpleCurvePointIterator . . . . .	??
OGRProj4Datum . . . . .	??
OGRProj4PM . . . . .	??
OGRRawPoint . . . . .	??
OGRSFDriverRegistrar . . . . .	??
OGRSpatialReference . . . . .	??
OGRStyleMgr . . . . .	??
OGRStyleTable . . . . .	??
OGRStyleTool . . . . .	??
osr_cs_wkt_tokens . . . . .	??
ParseContext . . . . .	??
PCIDatums . . . . .	??
OGRFeature::FieldValue::Private . . . . .	??
OGRLayer::FeatureIterator::Private . . . . .	??
OGRFeature::ConstFieldIterator::Private . . . . .	??
OGRCurve::ConstIterator::Private . . . . .	??
OGRSimpleCurve::Iterator::Private . . . . .	??
OGRSimpleCurve::ConstIterator::Private . . . . .	??
OGRLayer::Private . . . . .	??
SFRegion . . . . .	??
StackContext . . . . .	??
string	
CPLString . . . . .	??
TupleEnvVarOptionName . . . . .	??
unique_ptr	
CPLXMLTreeCloser . . . . .	??
unz_file_info_internal_s . . . . .	??
unz_s . . . . .	??
VSIDIR . . . . .	??
VSIErrorContext . . . . .	??
VSIFilesystemHandler	
VSI SparseFileFilesystemHandler . . . . .	??
VSI SubFileFilesystemHandler . . . . .	??
VSIReadDirRecursiveTask . . . . .	??
VSVirtualHandle . . . . .	??
VSI SparseFileHandle . . . . .	??
VSI SubFileHandle . . . . .	??
yyalloc . . . . .	??
zip_internal . . . . .	??



## Chapter 9

# Class Index

### 9.1 Class List

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

<b>_CPLHashSet</b>	??
<b>_CPLList</b>	??
<b>_CPLLock</b>	??
<b>_CPLQuadTree</b>	??
<b>_CPLSpawnedProcess</b>	??
<b>_CPLSpinLock</b>	??
<b>_MutexLinkedElt</b>	??
<b>_OGRGeocodingSessionHS</b>	??
<b>_QuadTreeNode</b>	??
<b>_sPolyExtended</b>	??
<b>OGRFeature::ConstFieldIterator</b>	??
<b>CPL_SHA1Context</b>	??
<b>CPLErrorContext</b>	??
<b>CPLHTTPErrorBuffer</b>	??
<b>CPLHTTPResult</b>	??
<b>CPLHTTPResultWithLimit</b>	??
<b>CPLJSONArray</b>	
The JSONArray class JSON array from JSONDocument	??
<b>CPLJSONDocument</b>	
Wrapper class around json-c library	??
<b>CPLJSONObject</b>	
The <b>CPLJSONArray</b> (p. ??) class holds JSON object from <b>CPLJSONDocument</b> (p. ??)	??
<b>CPLLockHolder</b>	??
<b>CPLMimePart</b>	??
<b>CPLMutexHolder</b>	??
<b>CPLODBCDriverInstaller</b>	??
<b>CPLODBCSession</b>	??
<b>CPLODBCStatement</b>	??
<b>CPLRectObj</b>	??
<b>CPLSharedFileInfo</b>	??
<b>CPLSharedFileInfoExtra</b>	??
<b>CPLStdCallThreadInfo</b>	??
<b>CPLString</b>	
Convenient string class based on std::string	??
<b>CPLStringList</b>	
String list class designed around our use of C "char*" string lists	??

CPLVirtualMem	??
CPLWorkerThreadPool	??
CPLXMLNode	??
CPLXMLTreeCloser	??
CPLZip	??
ctb	??
curfile_info	??
CurlProcessData	??
DefaultCSVFileNameTLS	??
errHandler	??
OGRFeature::FieldNotFoundException	??
OGRFeature::FieldValue	??
file_in_zip_read_info_s	??
FindFileTLS	??
GDALScaledProgressInfo	??
GOA2Manager	??
IOGRConstGeometryVisitor	??
IOGRGeometryVisitor	??
IOGRTransactionBehaviour	??
LinearUnitsStruct	??
linkedlist_data_s	??
linkedlist_datablock_internal_s	??
OGR_SRSNode	??
OGRCircularString	??
OGRCompoundCurve	??
OGRCompoundCurvePointIterator	??
OGRCoordinateTransformation	??
OGRCurve	??
OGRCurvePolygon	??
OGRDataSource	??
OGRDataSourceWithTransaction	??
OGRDefaultConstGeometryVisitor	??
OGRDefaultGeometryVisitor	??
OGRFeature	??
OGRFeatureDefn	??
OGRField	??
OGRFieldDefn	??
OGRGeometry	??
OGRGeometryCollection	??
OGRGeometryFactory	??
OGRGeomFieldDefn	??
OGRLayer	??
OGRLayerWithTransaction	??
OGRLinearRing	??
OGRLineString	??
OGRMAttrIndex	??
OGRMILayerAttrIndex	??
OGRMultiCurve	??
OGRMultiLineString	??
OGRMultiPoint	??
OGRMultiPolygon	??
OGRMultiSurface	??
OGRPoint	??
OGRPointIterator	??
OGRPolygon	??
OGRPolyhedralSurface	??
OGRProj4CT	??
OGRProj4Datum	??



OGRProj4PM	??
OGRRawPoint	??
OGRSFDriver	??
OGRSFDriverRegistrar	??
OGRSimpleCurve	??
OGRSimpleCurvePointIterator	??
OGRSpatialReference	??
OGRStyleMgr	??
OGRStyleTable	??
OGRStyleTool	??
OGRSurface	??
OGRTriangle	??
OGRTriangulatedSurface	??
osr_cs_wkt_tokens	??
ParseContext	??
PCIDatums	??
OGRFeature::FieldValue::Private	??
OGRLayer::FeatureIterator::Private	??
OGRFeature::ConstFieldIterator::Private	??
OGRCurve::ConstIterator::Private	??
OGRSimpleCurve::Iterator::Private	??
OGRSimpleCurve::ConstIterator::Private	??
OGRLayer::Private	??
SFRegion	??
StackContext	??
TupleEnvVarOptionName	??
unz_file_info_internal_s	??
unz_s	??
VSIDIR	??
VSIErrorContext	??
VSIReadDirRecursiveTask	??
VSI SparseFileFilesystemHandler	??
VSI SparseFileHandle	??
VSI SubFileFilesystemHandler	??
VSI SubFileHandle	??
VSI VirtualHandle	??
yyalloc	??
zip_internal	??



## Chapter 10

# File Index

### 10.1 File List

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

<b>cpl_alibaba_oss.h</b>	??
<b>cpl_atomic_ops.h</b>	??
<b>cpl_aws.h</b>	??
<b>cpl_azure.h</b>	??
<b>cpl_config.h</b>	??
<b>cpl_config_extras.h</b>	??
<b>cpl_conv.h</b>	??
<b>cpl_cpu_features.h</b>	??
<b>cpl_csv.h</b>	??
<b>cpl_error.h</b>	??
<b>cpl_google_cloud.h</b>	??
<b>cpl_hash_set.h</b>	??
<b>cpl_http.h</b>	??
<b>cpl_json.h</b>	??
<b>cpl_json_header.h</b>	??
<b>cpl_json_streaming_parser.h</b>	??
<b>cpl_list.h</b>	??
<b>cpl_md5.h</b>	??
<b>cpl_mem_cache.h</b>	??
<b>cpl_minixml.h</b>	??
<b>cpl_minizip_ioapi.h</b>	??
<b>cpl_minizip_unzip.h</b>	??
<b>cpl_minizip_zip.h</b>	??
<b>cpl_multiproc.h</b>	??
<b>cpl_odbc.h</b>	??
<b>cpl_port.h</b>	??
<b>cpl_progress.h</b>	??
<b>cpl_quad_tree.h</b>	??
<b>cpl_sha1.h</b>	??
<b>cpl_sha256.h</b>	??
<b>cpl_spawn.h</b>	??
<b>cpl_string.h</b>	??
<b>cpl_swift.h</b>	??
<b>cpl_time.h</b>	??
<b>cpl_virtualmem.h</b>	??

cpl_vsi.h	??
cpl_vsi_error.h	??
cpl_vsi_virtual.h	??
cpl_vsil_curl_priv.h	??
cpl_worker_thread_pool.h	??
cplkeywordparser.h	??
gdal_csv.h	??
ogr_api.h	??
ogr_attrind.h	??
ogr_core.h	??
ogr_expat.h	??
ogr_feature.h	??
ogr_featurestyle.h	??
ogr_gensql.h	??
ogr_geo_utils.h	??
ogr_geocoding.h	??
ogr_geometry.h	??
ogr_geos.h	??
ogr_libs.h	??
ogr_p.h	??
ogr_sfcgal.h	??
ogr_spatialref.h	??
ogr_srs_api.h	??
ogr_srs_esri_names.h	??
ogr_xerces.h	??
ogr_xerces_headers.h	??
ograpispy.h	??
ogreditablelayer.h	??
ogremulatedtransaction.h	??
ogrgeomediageometry.h	??
ogrlayerdecorator.h	??
ogrlayerpool.h	??
ogrmutexdatasource.h	??
ogrmutexedlayer.h	??
ogrpgogeometry.h	??
ogrsf_frmts.h	??
ogrunionlayer.h	??
ogrwarpedlayer.h	??
osr_cs_wkt.h	??
osr_cs_wkt_parser.h	??
swq.h	??

# Chapter 11

## Class Documentation

### 11.1 `_CPLHashSet` Struct Reference

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

- `cpl_hash_set.cpp`

### 11.2 `_CPLList` Struct Reference

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

#### Public Attributes

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

#### 11.2.1 Detailed Description

List element structure.

#### 11.2.2 Member Data Documentation

##### 11.2.2.1 `pData`

```
void* _CPLList::pData
```

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

Referenced by `CPLHashSetForeach()`, `CPLHashSetInsert()`, `CPLListAppend()`, `CPLListGetData()`, `CPLListInsert()`, `CPLWorkerThreadPool::SubmitJob()`, and `CPLWorkerThreadPool::SubmitJobs()`.

### 11.2.2.2 psNext

```
struct _CPLList* _CPLList::psNext
```

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

Referenced by CPLHashSetForeach(), CPLHashSetInsert(), CPLListAppend(), CPLListCount(), CPLListDestroy(), CPLListGet(), CPLListGetLast(), CPLListGetNext(), CPLListInsert(), CPLListRemove(), CPLWorkerThreadPool::SubmitJob(), and CPLWorkerThreadPool::SubmitJobs().

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

- `cpl_list.h`

## 11.3 \_CPLLock Struct Reference

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

- `cpl_multiproc.cpp`

## 11.4 \_CPLQuadTree Struct Reference

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

- `cpl_quad_tree.cpp`

## 11.5 \_CPLSpawnedProcess Struct Reference

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

- `cpl_spawn.cpp`

## 11.6 \_CPLSpinLock Struct Reference

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

- `cpl_multiproc.cpp`

## 11.7 \_MutexLinkedElit Struct Reference

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

- `cpl_multiproc.cpp`

## 11.8 \_OGRGeocodingSessionHS Struct Reference

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

- ogr\_geocoding.cpp

## 11.9 \_QuadTreeNode Struct Reference

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

- cpl\_quad\_tree.cpp

## 11.10 \_sPolyExtended Struct Reference

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

- ogrgeometryfactory.cpp

## 11.11 OGRFeature::ConstFieldIterator Class Reference

```
#include <ogr_feature.h>
```

### Classes

- struct **Private**

### Friends

- class **OGRFeature**

### 11.11.1 Detailed Description

Field value iterator class.

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

- **ogr\_feature.h**

## 11.12 CPL\_SHA1Context Struct Reference

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

- `cpl_sha1.cpp`

## 11.13 CPL\_ErrorContext Struct Reference

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

- `cpl_error.cpp`

## 11.14 CPLHTTPErrorBuffer Class Reference

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

- `cpl_http.cpp`

## 11.15 CPLHTTPResult Struct Reference

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

### Public Attributes

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

### 11.15.1 Detailed Description

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

### 11.15.2 Member Data Documentation



#### 11.15.2.1 nDataAlloc

```
int CPLHTTPResult::nDataAlloc
```

Allocated size of the pabyData buffer

#### 11.15.2.2 nDataLen

```
int CPLHTTPResult::nDataLen
```

Length of the pabyData buffer

#### 11.15.2.3 nMimePartCount

```
int CPLHTTPResult::nMimePartCount
```

Number of parts in a multipart message

Referenced by CPLHTTPDestroyResult(), and CPLHTTPParseMultipartMime().

#### 11.15.2.4 nStatus

```
int CPLHTTPResult::nStatus
```

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

Referenced by CPLHTTPFetchEx(), and CPLJSONDocument::LoadUrl().

#### 11.15.2.5 pabyData

```
GByte* CPLHTTPResult::pabyData
```

Buffer with downloaded data

Referenced by CPLHTTPDestroyResult(), and GOA2GetRefreshToken().

#### 11.15.2.6 papszHeaders

```
char** CPLHTTPResult::papszHeaders
```

Headers returned

Referenced by CPLHTTPDestroyResult().

### 11.15.2.7 pasMimePart

**CPLMimePart\*** CPLHTTPResult::pasMimePart

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

Referenced by CPLHTTPDestroyResult().

### 11.15.2.8 pszContentType

char\* CPLHTTPResult::pszContentType

Content-Type of the response

Referenced by CPLHTTPDestroyResult(), and CPLHTTPParseMultipartMime().

### 11.15.2.9 pszErrBuf

char\* CPLHTTPResult::pszErrBuf

Error message from curl, or NULL

Referenced by CPLHTTPDestroyResult(), and CPLHTTPFetchEx().

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

- **cpl\_http.h**

## 11.16 CPLHTTPResultWithLimit Class Reference

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

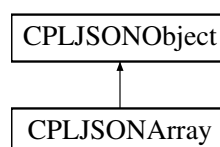
- **cpl\_http.cpp**

## 11.17 CPLJSONArray Class Reference

The JSONArray class JSON array from JSONDocument.

```
#include <cpl_json.h>
```

Inheritance diagram for CPLJSONArray:



## Public Member Functions

- int **Size** () const
- void **Add** (const **CPLJSONObject** &oValue)
- void **Add** (const std::string &osValue)
- void **Add** (const char \*pszValue)
- void **Add** (double dfValue)
- void **Add** (int nValue)
- void **Add** ( **GInt64** nValue)
- void **Add** (bool bValue)
- **CPLJSONObject** **operator[]** (int nIndex)
- const **CPLJSONObject** **operator[]** (int nIndex) const

## Friends

- class **CPLJSONObject**
- class **CPLJSONDocument**

## Additional Inherited Members

### 11.17.1 Detailed Description

The JSONArray class JSON array from JSONDocument.

### 11.17.2 Member Function Documentation

#### 11.17.2.1 Add() [1/7]

```
void CPLJSONArray::Add (
    const CPLJSONObject & oValue )
```

Add json object to array.

#### Parameters

<i>oValue</i>	Json array.
---------------	-------------

#### Since

GDAL 2.3

**11.17.2.2 Add()** [2/7]

```
void CPLJSONArray::Add (
    const std::string & osValue )
```

Add value to array

**Parameters**

<i>osValue</i>	Value to add.
----------------	---------------

**Since**

GDAL 2.3

**11.17.2.3 Add()** [3/7]

```
void CPLJSONArray::Add (
    const char * pszValue )
```

Add value to array

**Parameters**

<i>pszValue</i>	Value to add.
-----------------	---------------

**Since**

GDAL 2.3

**11.17.2.4 Add()** [4/7]

```
void CPLJSONArray::Add (
    double dfValue )
```

Add value to array

**Parameters**

<i>dfValue</i>	Value to add.
----------------	---------------

**Since**

GDAL 2.3

#### 11.17.2.5 Add() [5/7]

```
void CPLJSONArray::Add (
    int nValue )
```

Add value to array

##### Parameters

<i>nValue</i>	Value to add.
---------------	---------------

Since

GDAL 2.3

#### 11.17.2.6 Add() [6/7]

```
void CPLJSONArray::Add (
    GInt64 nValue )
```

Add value to array

##### Parameters

<i>nValue</i>	Value to add.
---------------	---------------

Since

GDAL 2.3

#### 11.17.2.7 Add() [7/7]

```
void CPLJSONArray::Add (
    bool bValue )
```

Add value to array

##### Parameters

<i>bValue</i>	Value to add.
---------------	---------------

Since

GDAL 2.3

#### 11.17.2.8 `operator[]()` [1/2]

```
CPLJSONObject CPLJSONArray::operator[] (
    int nIndex )
```

Get array item by index.

Parameters

<code>nIndex</code>	Item index.
---------------------	-------------

Returns

Json object.

Since

GDAL 2.3

References `CPLSPrintf()`.

#### 11.17.2.9 `operator[]()` [2/2]

```
const CPLJSONObject CPLJSONArray::operator[] (
    int nIndex ) const
```

Get array const item by index.

Parameters

<code>nIndex</code>	Item index.
---------------------	-------------

Returns

Json object.

Since

GDAL 2.3

References `CPLSPrintf()`.

## 11.17.2.10 Size()

```
int CPLJSONArray::Size ( ) const
```

Get array size.

## Returns

Array size.

## Since

GDAL 2.3

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

- **cpl\_json.h**
- **cpl\_json.cpp**

## 11.18 CPLJSONDocument Class Reference

The **CPLJSONDocument** (p. ??) class Wrapper class around json-c library.

```
#include <cpl_json.h>
```

### Public Member Functions

- bool **Save** (const std::string &osPath)
- std::string **SaveAsString** ()
- **CPLJSONObject** **GetRoot** ()
- bool **Load** (const std::string &osPath)
- bool **LoadMemory** (const std::string &osStr)
- bool **LoadMemory** (const **GByte** \*pabyData, int nLength=-1)
- bool **LoadChunks** (const std::string &osPath, size\_t nChunkSize=16384, GDALProgressFunc pfnProgress=NULL, void \*pProgressArg=NULL)
- bool **LoadUrl** (const std::string &osUrl, char \*\*papszOptions, GDALProgressFunc pfnProgress=NULL, void \*pProgressArg=NULL)

### 11.18.1 Detailed Description

The **CPLJSONDocument** (p. ??) class Wrapper class around json-c library.

### 11.18.2 Member Function Documentation

### 11.18.2.1 GetRoot()

```
CPLJSONObject CPLJSONDocument::GetRoot ( )
```

Get json document root object

#### Returns

**CPLJSONObject** (p. ??) class instance

#### Since

GDAL 2.3

Referenced by OGRGeometryFactory::createFromGeoJson().

### 11.18.2.2 Load()

```
bool CPLJSONDocument::Load (
    const std::string & osPath )
```

Load json document from file by provided path

#### Parameters

<i>osPath</i>	Path to json file.
---------------	--------------------

#### Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

#### Since

GDAL 2.3

References CPLError(), and VSILIngestFile().

### 11.18.2.3 LoadChunks()

```
bool CPLJSONDocument::LoadChunks (
    const std::string & osPath,
    size_t nChunkSize = 16384,
    GDALProgressFunc pfnProgress = nullptr,
    void * pProgressArg = nullptr )
```

Load json document from file using small chunks of data.



## Parameters

<i>osPath</i>	Path to json document file.
<i>nChunkSize</i>	Chunk size.
<i>pfnProgress</i>	a function to report progress of the json data loading.
<i>pProgressArg</i>	application data passed into progress function.

## Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

## Since

GDAL 2.3

References CPLError(), and VSISatL().

## 11.18.2.4 LoadMemory() [1/2]

```
bool CPLJSONDocument::LoadMemory (
    const std::string & osStr )
```

Load json document from memory buffer.

## Parameters

<i>osStr</i>	String
--------------	--------

## Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

## Since

GDAL 2.3

References LoadMemory().

Referenced by OGRGeometryFactory::createFromGeoJson(), and LoadMemory().

## 11.18.2.5 LoadMemory() [2/2]

```
bool CPLJSONDocument::LoadMemory (
    const GByte * pabyData,
    int nLength = -1 )
```

Load json document from memory buffer.

**Parameters**

<i>pabyData</i>	Buffer.data.
<i>nLength</i>	Buffer size.

**Returns**

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

**Since**

GDAL 2.3

References CPLError().

**11.18.2.6 LoadUrl()**

```
bool CPLJSONDocument::LoadUrl (
    const std::string & osUrl,
    char ** papszOptions,
    GDALProgressFunc pfnProgress = nullptr,
    void * pProgressArg = nullptr )
```

Load json document from web.

**Parameters**

<i>osUrl</i>	Url to json document.
<i>papszOptions</i>	Option list as a NULL-terminated array of strings. May be NULL. The available keys are same for CPLHTTPFetch method. Additional key JSON_DEPTH define json parse depth. Default is 10.
<i>pfnProgress</i>	a function to report progress of the json data loading.
<i>pProgressArg</i>	application data passed into progress function.

**Returns**

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

**Since**

GDAL 2.3

References CPLError(), CPLHTTPDestroyResult(), CPLHTTPFetchEx(), CSLFetchNameValueDef(), and CPLHTTPResult::nStatus.

## 11.18.2.7 Save()

```
bool CPLJSONDocument::Save (
    const std::string & osPath )
```

Save json document at specified path

## Parameters

<i>osPath</i>	Path to save json document
---------------	----------------------------

## Returns

true on success. If error occurred it can be received using CPLGetLastErrorMsg method.

## Since

GDAL 2.3

References CPLError(), and VSIFOpenL().

## 11.18.2.8 SaveAsString()

```
std::string CPLJSONDocument::SaveAsString ( )
```

Return the json document as a serialized string.

## Returns

serialized document.

## Since

GDAL 2.3

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

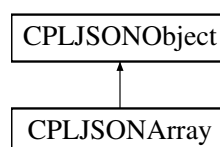
- **cpl\_json.h**
- cpl\_json.cpp

## 11.19 CPLJSONObject Class Reference

The **CPLJSONArray** (p. ??) class holds JSON object from **CPLJSONDocument** (p. ??).

```
#include <cpl_json.h>
```

Inheritance diagram for CPLJSONObject:



## Public Types

- enum **Type**
- enum **PrettyFormat** { **Plain**, **Spaced**, **Pretty** }

## Public Member Functions

- void **Add** (const std::string &osName, const std::string &osValue)
- void **Add** (const std::string &osName, const char \*pszValue)
- void **Add** (const std::string &osName, double dfValue)
- void **Add** (const std::string &osName, int nValue)
- void **Add** (const std::string &osName, **GInt64** nValue)
- void **Add** (const std::string &osName, const **CPLJSONArray** &oValue)
- void **Add** (const std::string &osName, const **CPLJSONObject** &oValue)
- void **Add** (const std::string &osName, bool bValue)
- void **AddNull** (const std::string &osName)
- void **Set** (const std::string &osName, const std::string &osValue)
- void **Set** (const std::string &osName, const char \*pszValue)
- void **Set** (const std::string &osName, double dfValue)
- void **Set** (const std::string &osName, int nValue)
- void **Set** (const std::string &osName, **GInt64** nValue)
- void **Set** (const std::string &osName, bool bValue)
- void **SetNull** (const std::string &osName)
- std::string **GetString** (const std::string &osName, const std::string &osDefault="") const
- double **GetDouble** (const std::string &osName, double dfDefault=0.0) const
- int **GetInteger** (const std::string &osName, int nDefault=0) const
- **GInt64** **GetLong** (const std::string &osName, **GInt64** nDefault=0) const
- bool **GetBool** (const std::string &osName, bool bDefault=false) const
- std::string **ToString** (const std::string &osDefault="") const
- double **ToDouble** (double dfDefault=0.0) const
- int **ToInteger** (int nDefault=0) const
- **GInt64** **ToLong** (**GInt64** nDefault=0) const
- bool **ToBool** (bool bDefault=false) const
- **CPLJSONArray** **ToArray** () const
- std::string **Format** (enum **PrettyFormat** eFormat) const
- void **Delete** (const std::string &osName)
- **CPLJSONArray** **GetArray** (const std::string &osName) const
- **CPLJSONObject** **GetObj** (const std::string &osName) const
- **CPLJSONObject** **operator[]** (const std::string &osName) const
- enum **Type** **GetType** () const
- std::vector< **CPLJSONObject** > **GetChildren** () const  
*Get json object children.*
- bool **IsValid** () const
- void **Deinit** ()

## Friends

- class **CPLJSONArray**
- class **CPLJSONDocument**

### 11.19.1 Detailed Description

The **CPLJSONArray** (p. ??) class holds JSON object from **CPLJSONDocument** (p. ??).

### 11.19.2 Member Enumeration Documentation

#### 11.19.2.1 PrettyFormat

enum **CPLJSONObject::PrettyFormat**

Json object format to string options

Enumerator

Plain	No extra whitespace or formatting applied.
Spaced	Minimal whitespace inserted.
Pretty	Formatted output.

#### 11.19.2.2 Type

enum **CPLJSONObject::Type**

Json object types

### 11.19.3 Member Function Documentation

#### 11.19.3.1 Add() [1/8]

```
void CPLJSONObject::Add (
    const std::string & osName,
    const std::string & osValue )
```

Add new key - value pair to json object.

Parameters

<i>osName</i>	Key name.
<i>osValue</i>	String value.

Since

GDAL 2.3

References IsValid().

Referenced by Set().

### 11.19.3.2 Add() [2/8]

```
void CPLJSONObject::Add (
    const std::string & osName,
    const char * pszValue )
```

Add new key - value pair to json object.

Parameters

<i>osName</i>	Key name.
<i>pszValue</i>	String value.

Since

GDAL 2.3

References IsValid().

### 11.19.3.3 Add() [3/8]

```
void CPLJSONObject::Add (
    const std::string & osName,
    double dfValue )
```

Add new key - value pair to json object.

Parameters

<i>osName</i>	Key name.
<i>dfValue</i>	Double value.

Since

GDAL 2.3

References IsValid().

**11.19.3.4 Add()** [4/8]

```
void CPLJSONObject::Add (
    const std::string & osName,
    int nValue )
```

Add new key - value pair to json object.

**Parameters**

<i>osName</i>	Key name.
<i>nValue</i>	Integer value.

**Since**

GDAL 2.3

References IsValid().

**11.19.3.5 Add()** [5/8]

```
void CPLJSONObject::Add (
    const std::string & osName,
    GInt64 nValue )
```

Add new key - value pair to json object.

**Parameters**

<i>osName</i>	Key name.
<i>nValue</i>	Long value.

**Since**

GDAL 2.3

References IsValid().

**11.19.3.6 Add()** [6/8]

```
void CPLJSONObject::Add (
    const std::string & osName,
    const CPLJSONArray & oValue )
```

Add new key - value pair to json object.

## Parameters

<i>osName</i>	Key name.
<i>oValue</i>	Array value.

## Since

GDAL 2.3

References IsValid().

## 11.19.3.7 Add() [7/8]

```
void CPLJSONObject::Add (
    const std::string & osName,
    const  CPLJSONObject & oValue )
```

Add new key - value pair to json object.

## Parameters

<i>osName</i>	Key name.
<i>oValue</i>	Json object value.

## Since

GDAL 2.3

References IsValid().

## 11.19.3.8 Add() [8/8]

```
void CPLJSONObject::Add (
    const std::string & osName,
    bool bValue )
```

Add new key - value pair to json object.

## Parameters

<i>osName</i>	Key name.
<i>bValue</i>	Boolean value.



Since

GDAL 2.3

References IsValid().

#### 11.19.3.9 AddNull()

```
void CPLJSONObject::AddNull (
    const std::string & osName )
```

Add new key - null pair to json object.

Parameters

<i>osName</i>	Key name.
---------------	-----------

Since

GDAL 2.3

References IsValid().

Referenced by SetNull().

#### 11.19.3.10 Deinit()

```
void CPLJSONObject::Deinit ( )
```

Decrement reference counter and make pointer NULL. A json object will become invalid.

Since

GDAL 2.3

#### 11.19.3.11 Delete()

```
void CPLJSONObject::Delete (
    const std::string & osName )
```

Delete json object by key.

**Parameters**

<i>osName</i>	Key name.
---------------	-----------

**Since**

GDAL 2.3

References IsValid().

Referenced by Set(), and SetNull().

**11.19.3.12 Format()**

```
std::string CPLJSONObject::Format (
    enum PrettyFormat eFormat ) const
```

Stringify object to json format.

**Parameters**

<i>eFormat</i>	Format type,
----------------	--------------

**Returns**

A string in JSON format.

**Since**

GDAL 2.3

References Pretty, and Spaced.

**11.19.3.13 GetArray()**

```
CPLJSONArray CPLJSONObject::GetArray (
    const std::string & osName ) const
```

Get value by key.

**Parameters**

<i>osName</i>	Key name.
---------------	-----------

**Returns**

Json array object.

**Since**

GDAL 2.3

References IsValid().

**11.19.3.14 GetBool()**

```
bool CPLJSONObject::GetBool (
    const std::string & osName,
    bool bDefault = false ) const
```

Get value by key.

**Parameters**

<i>osName</i>	Key name.
<i>bDefault</i>	Default value.

**Returns**

Boolean value.

**Since**

GDAL 2.3

References GetObj().

**11.19.3.15 GetChildren()**

```
std::vector< CPLJSONObject > CPLJSONObject::GetChildren ( ) const
```

Get json object children.

This function is useful when keys is not know and need to iterate over json object items and get keys and values.

**Returns**

Array of **CPLJSONObject** (p. ??) class instance.

**Since**

GDAL 2.3

### 11.19.3.16 GetDouble()

```
double CPLJSONObject::GetDouble (
    const std::string & osName,
    double dfDefault = 0.0 ) const
```

Get value by key.

#### Parameters

<i>osName</i>	Key name.
<i>dfDefault</i>	Default value.

#### Returns

Double value.

#### Since

GDAL 2.3

References GetObj().

### 11.19.3.17 GetInteger()

```
int CPLJSONObject::GetInteger (
    const std::string & osName,
    int nDefault = 0 ) const
```

Get value by key.

#### Parameters

<i>osName</i>	Key name.
<i>nDefault</i>	Default value.

#### Returns

Integer value.

#### Since

GDAL 2.3

References GetObj().

### 11.19.3.18 GetLong()

```
GInt64 CPLJSONObject::GetLong (
    const std::string & osName,
    GInt64 nDefault = 0 ) const
```

Get value by key.

#### Parameters

<i>osName</i>	Key name.
<i>nDefault</i>	Default value.

#### Returns

Long value.

#### Since

GDAL 2.3

References GetObj().

### 11.19.3.19 GetObj()

```
CPLJSONObject CPLJSONObject::GetObj (
    const std::string & osName ) const
```

Get value by key.

#### Parameters

<i>osName</i>	Key name.
---------------	-----------

#### Returns

Json object.

#### Since

GDAL 2.3

References IsValid().

Referenced by GetBool(), GetDouble(), GetInteger(), GetLong(), GetString(), and operator[]().

### 11.19.3.20 GetString()

```
std::string CPLJSONObject::GetString (
    const std::string & osName,
    const std::string & osDefault = "" ) const
```

Get value by key.

#### Parameters

<i>osName</i>	Key name.
<i>osDefault</i>	Default value.

#### Returns

String value.

#### Since

GDAL 2.3

References GetObj().

### 11.19.3.21 GetType()

```
CPLJSONObject::Type CPLJSONObject::GetType ( ) const
```

Get json object type.

#### Returns

Json object type.

#### Since

GDAL 2.3

### 11.19.3.22 IsValid()

```
bool CPLJSONObject::IsValid ( ) const
```

Check if json object valid.

#### Returns

true if json object valid.

#### Since

GDAL 2.3

Referenced by Add(), AddNull(), OGRGeometryFactory::createFromGeoJson(), Delete(), GetArray(), and GetObj().

## 11.19.3.23 operator[]()

```
CPLJSONObject CPLJSONObject::operator[] (
    const std::string & osName ) const
```

Get value by key.

## Parameters

<i>osName</i>	Key name.
---------------	-----------

## Returns

Json object.

## Since

GDAL 2.3

References GetObj().

## 11.19.3.24 Set() [1/6]

```
void CPLJSONObject::Set (
    const std::string & osName,
    const std::string & osValue )
```

Change value by key.

## Parameters

<i>osName</i>	Key name.
<i>osValue</i>	String value.

## Since

GDAL 2.3

References Add(), and Delete().

## 11.19.3.25 Set() [2/6]

```
void CPLJSONObject::Set (
    const std::string & osName,
    const char * pszValue )
```

Change value by key.

**Parameters**

<i>osName</i>	Key name.
<i>pszValue</i>	String value.

**Since**

GDAL 2.3

References Add(), and Delete().

**11.19.3.26 Set()** [3/6]

```
void CPLJSONObject::Set (
    const std::string & osName,
    double dfValue )
```

Change value by key.

**Parameters**

<i>osName</i>	Key name.
<i>dfValue</i>	Double value.

**Since**

GDAL 2.3

References Add(), and Delete().

**11.19.3.27 Set()** [4/6]

```
void CPLJSONObject::Set (
    const std::string & osName,
    int nValue )
```

Change value by key.

**Parameters**

<i>osName</i>	Key name.
<i>nValue</i>	Integer value.



Since

GDAL 2.3

References Add(), and Delete().

#### 11.19.3.28 Set() [ 5 / 6 ]

```
void CPLJSONObject::Set (
    const std::string & osName,
    GInt64 nValue )
```

Change value by key.

Parameters

<i>osName</i>	Key name.
<i>nValue</i>	Long value.

Since

GDAL 2.3

References Add(), and Delete().

#### 11.19.3.29 Set() [ 6 / 6 ]

```
void CPLJSONObject::Set (
    const std::string & osName,
    bool bValue )
```

Change value by key.

Parameters

<i>osName</i>	Key name.
<i>bValue</i>	Boolean value.

Since

GDAL 2.3

References Add(), and Delete().

### 11.19.3.30 SetNull()

```
void CPLJSONObject::SetNull (
    const std::string & osName )
```

Change value by key.

#### Parameters

<i>osName</i>	Key name.
---------------	-----------

#### Since

GDAL 2.3

References AddNull(), and Delete().

### 11.19.3.31 ToArray()

```
CPLJSONArray CPLJSONObject::ToArray ( ) const
```

Get value.

#### Returns

Array

#### Since

GDAL 2.3

### 11.19.3.32 ToBool()

```
bool CPLJSONObject::ToBool (
    bool bDefault = false ) const
```

Get value.

#### Parameters

<i>bDefault</i>	Default value.
-----------------	----------------

**Returns**

Boolean value.

**Since**

GDAL 2.3

**11.19.3.33 ToDouble()**

```
double CPLJSONObject::ToDouble (
    double dfDefault = 0.0 ) const
```

**Get value****Parameters**

<i>dfDefault</i>	Default value.
------------------	----------------

**Returns**

Double value.

**Since**

GDAL 2.3

**11.19.3.34 ToInteger()**

```
int CPLJSONObject::ToInteger (
    int nDefault = 0 ) const
```

**Get value.****Parameters**

<i>nDefault</i>	Default value.
-----------------	----------------

**Returns**

Integer value.

**Since**

GDAL 2.3

### 11.19.3.35 ToLong()

```
GInt64 CPLJSONObject::ToLong (
    GInt64 nDefault = 0 ) const
```

Get value.

#### Parameters

<i>nDefault</i>	Default value.
-----------------	----------------

#### Returns

Long value.

#### Since

GDAL 2.3

### 11.19.3.36 ToString()

```
std::string CPLJSONObject::ToString (
    const std::string & osDefault = "" ) const
```

Get value.

#### Parameters

<i>osDefault</i>	Default value.
------------------	----------------

#### Returns

String value.

#### Since

GDAL 2.3

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

- **cpl\_json.h**
- **cpl\_json.cpp**

## 11.20 CPLLockHolder Class Reference

```
#include <cpl_multiproc.h>
```

## Public Member Functions

- **CPLLockHolder** ( **CPLLock** \*\*phSpin, CPLLockType eType, const char \*pszFile=\_\_FILE\_\_, int nLine=\_\_LINE\_\_ )
- **CPLLockHolder** ( **CPLLock** \*hSpin, const char \*pszFile=\_\_FILE\_\_, int nLine=\_\_LINE\_\_ )

### 11.20.1 Detailed Description

Object to hold a lock

### 11.20.2 Constructor & Destructor Documentation

#### 11.20.2.1 CPLLockHolder() [1/2]

```
CPLLockHolder::CPLLockHolder (
    CPLLock ** phSpin,
    CPLLockType eType,
    const char * pszFile = __FILE__,
    int nLine = __LINE__ )
```

Instantiates the lock if not already done.

#### 11.20.2.2 CPLLockHolder() [2/2]

```
CPLLockHolder::CPLLockHolder (
    CPLLock * hSpin,
    const char * pszFile = __FILE__,
    int nLine = __LINE__ )
```

This variant assumes the lock has already been created. If not, it will be a no-op

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

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

## 11.21 CPLMimePart Struct Reference

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

## Public Attributes

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

### 11.21.1 Detailed Description

Describe a part of a multipart message

### 11.21.2 Member Data Documentation

#### 11.21.2.1 nDataLen

```
int CPLMimePart::nDataLen
```

Buffer length

#### 11.21.2.2 pabyData

```
GByte* CPLMimePart::pabyData
```

Buffer with data of the part

#### 11.21.2.3 papszHeaders

```
char** CPLMimePart::papszHeaders
```

NULL terminated array of headers

Referenced by CPLHTTPDestroyResult().

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

- **cpl\_http.h**

## 11.22 CPLMutexHolder Class Reference

```
#include <cpl_multiproc.h>
```

### Public Member Functions

- **CPLMutexHolder** (void \*\*phMutex, double dfWaitInSeconds=1000.0, const char \*pszFile=\_\_FILE\_\_, int nLine=\_\_LINE\_\_, int nOptions=0)
- **CPLMutexHolder** (void \*hMutex, double dfWaitInSeconds=1000.0, const char \*pszFile=\_\_FILE\_\_, int nLine=\_\_LINE\_\_)

### 11.22.1 Detailed Description

Object to hold a mutex

### 11.22.2 Constructor & Destructor Documentation

#### 11.22.2.1 CPLMutexHolder() [1/2]

```
CPLMutexHolder::CPLMutexHolder (
    void ** phMutex,
    double dfWaitInSeconds = 1000.0,
    const char * pszFile = __FILE__,
    int nLine = __LINE__,
    int nOptions = 0 )
```

Instantiates the mutex if not already done.

#### 11.22.2.2 CPLMutexHolder() [2/2]

```
CPLMutexHolder::CPLMutexHolder (
    void * hMutex,
    double dfWaitInSeconds = 1000.0,
    const char * pszFile = __FILE__,
    int nLine = __LINE__ )
```

This variant assumes the mutex has already been created. If not, it will be a no-op

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

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

## 11.23 CPODBCDriverInstaller Class Reference

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

### Public Member Functions

- int **InstallDriver** (const char \*pszDriver, const char \*pszPathIn, WORD fRequest=ODBC\_INSTALL\_COM↵  
LETE)
- int **RemoveDriver** (const char \*pszDriverName, int fRemoveDSN=FALSE)
- int **GetUsageCount** () const
- const char \* **GetPathOut** () const
- const char \* **GetLastError** () const
- DWORD **GetLastErrorCode** () const

### 11.23.1 Detailed Description

A class providing functions to install or remove ODBC driver.

### 11.23.2 Member Function Documentation

#### 11.23.2.1 GetLastError()

```
const char* CPLODBCDriverInstaller::GetLastError ( ) const [inline]
```

If InstallDriver returns FALSE, then GetLastError then error message can be obtained by calling this function. Internally, it calls ODBC's SQLInstallerError function.

#### 11.23.2.2 GetLastErrorCode()

```
DWORD CPLODBCDriverInstaller::GetLastErrorCode ( ) const [inline]
```

If InstallDriver returns FALSE, then GetLastErrorCode then error code can be obtained by calling this function. Internally, it calls ODBC's SQLInstallerError function. See ODBC API Reference for possible error flags.

#### 11.23.2.3 GetPathOut()

```
const char* CPLODBCDriverInstaller::GetPathOut ( ) const [inline]
```

Path of the target directory where the driver should be installed. For details, see ODBC API Reference and lpsz↔PathOut parameter of SQLInstallDriverEx

#### 11.23.2.4 GetUsageCount()

```
int CPLODBCDriverInstaller::GetUsageCount ( ) const [inline]
```

The usage count of the driver after this function has been called

#### 11.23.2.5 InstallDriver()

```
int CPLODBCDriverInstaller::InstallDriver (
    const char * pszDriver,
    const char * pszPathIn,
    WORD fRequest = ODBC_INSTALL_COMPLETE )
```

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



## Parameters

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

## Returns

TRUE indicates success, FALSE if it fails.

References CPL\_UNUSED, CPLAssert, CPLDebug(), and CPLMalloc().

## 11.23.2.6 RemoveDriver()

```
int CPLODBCDriverInstaller::RemoveDriver (
    const char * pszDriverName,
    int fRemoveDSN = FALSE )
```

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

## Parameters

<i>pszDriverName</i>	- The name of the driver as registered in the Odbcinst.ini key of the system information.
<i>fRemoveDSN</i>	- TRUE: Remove DSNs associated with the driver specified in 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()** (p. ??).

References CPLAssert.

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

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

## 11.24 CPLODBCSession Class Reference

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

## Public Member Functions

- **CPLODBCSession** ()
- **~CPLODBCSession** ()
- int **EstablishSession** (const char \*pszDSN, const char \*pszUserid, const char \*pszPassword)
- const char \* **GetLastError** ()
- int **ClearTransaction** ()
- int **BeginTransaction** ()
- int **CommitTransaction** ()
- int **RollbackTransaction** ()
- int **IsInTransaction** ()
- int **CloseSession** ()
- int **Failed** (int, HSTMT=nullptr)
- HDBC **GetConnection** ()
- HENV **GetEnvironment** ()

### 11.24.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

### 11.24.2 Constructor & Destructor Documentation

#### 11.24.2.1 CPLODBCSession()

```
CPLODBCSession::CPLODBCSession ( )
```

Constructor

#### 11.24.2.2 ~CPLODBCSession()

```
CPLODBCSession::~~CPLODBCSession ( )
```

Destructor

References `CloseSession()`.

### 11.24.3 Member Function Documentation

#### 11.24.3.1 BeginTransaction()

```
int CPLODBCSession::BeginTransaction ( )
```

Begin transaction

References Failed().

#### 11.24.3.2 ClearTransaction()

```
int CPLODBCSession::ClearTransaction ( )
```

Clear transaction

References Failed().

Referenced by CPLODBCStatement::ExecuteSQL(), CPLODBCStatement::GetColumns(), CPLODBCStatement::GetPrimaryKeys(), and CPLODBCStatement::GetTables().

#### 11.24.3.3 CloseSession()

```
int CPLODBCSession::CloseSession ( )
```

Close session

References CPLError(), and IsInTransaction().

Referenced by EstablishSession(), and ~CPLODBCSession().

#### 11.24.3.4 CommitTransaction()

```
int CPLODBCSession::CommitTransaction ( )
```

Commit transaction

References Failed().

#### 11.24.3.5 EstablishSession()

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

Connect to database and logon.

**Parameters**

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

**Returns**

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

References `CloseSession()`, `CPLDebug()`, and `Failed()`.

**11.24.3.6 Failed()**

```
int CPODBCSession::Failed (
    int nRetCode,
    HSTMT hStmt = nullptr )
```

Test if a return code indicates failure, return TRUE if that is the case. Also update error text.

References `RollbackTransaction()`.

Referenced by `BeginTransaction()`, `ClearTransaction()`, `CommitTransaction()`, and `EstablishSession()`.

**11.24.3.7 GetConnection()**

```
HDBC CPODBCSession::GetConnection ( ) [inline]
```

Return connection handle

Referenced by `CPODBCStatement::CPODBCStatement()`.

**11.24.3.8 GetEnvironment()**

```
HENV CPODBCSession::GetEnvironment ( ) [inline]
```

Return `GetEnvironment` handle

#### 11.24.3.9 GetLastError()

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

Returns the last ODBC error message.

##### Returns

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

#### 11.24.3.10 IsInTransaction()

```
int CPODBCSession::IsInTransaction ( ) [inline]
```

Returns whether a transaction is active

Referenced by CloseSession(), CPODBCStatement::ExecuteSQL(), CPODBCStatement::GetColumns(), CPODBCStatement::GetPrimaryKeys(), and CPODBCStatement::GetTables().

#### 11.24.3.11 RollbackTransaction()

```
int CPODBCSession::RollbackTransaction ( )
```

Rollback transaction

Referenced by Failed().

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

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

## 11.25 CPODBCStatement Class Reference

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

## Public Member Functions

- **CPLDBCStatement** ( **CPLDBCSession** \*)
- **~CPLDBCStatement** ()
- **HSTMT GetStatement** ()
- **void Clear** ()
- **void AppendEscaped** (const char \*)
- **void Append** (const char \*)
- **void Append** (int)
- **void Append** (double)
- **int Appendf** (const char \*,...)
- **const char \* GetCommand** ()
- **int ExecuteSQL** (const char \*==nullptr)
- **int Fetch** (int nOrientation=SQL\_FETCH\_NEXT, int nOffset=0)
- **void ClearColumnData** ()
- **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 \*==nullptr)
- **const char \* GetColData** (const char \*, const char \*==nullptr)
- **int GetColDataLength** (int)
- **int GetRowCountAffected** ()
- **int GetColumns** (const char \*pszTable, const char \*pszCatalog=nullptr, const char \*pszSchema=nullptr)
- **int GetPrimaryKeys** (const char \*pszTable, const char \*pszCatalog=nullptr, const char \*pszSchema=nullptr)
- **int GetTables** (const char \*pszCatalog=nullptr, const char \*pszSchema=nullptr)
- **void DumpResult** (FILE \*fp, int bShowSchema=FALSE)
- **int CollectResultsInfo** ()

## Static Public Member Functions

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

### 11.25.1 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.

### 11.25.2 Constructor & Destructor Documentation

## 11.25.2.1 CPODBCStatement()

```
CPODBCStatement::CPODBCStatement (
    CPODBCSession * poSession ) [explicit]
```

Constructor

References CPODBCSession::GetConnection().

## 11.25.2.2 ~CPODBCStatement()

```
CPODBCStatement::~~CPODBCStatement ( )
```

Destructor

References Clear().

## 11.25.3 Member Function Documentation

## 11.25.3.1 Append() [1/3]

```
void CPODBCStatement::Append (
    const char * pszText )
```

Append text to internal command.

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

Parameters

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

References CPLRealloc(), and VSIMalloc().

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

## 11.25.3.2 Append() [2/3]

```
void CPODBCStatement::Append (
    int nValue )
```

Append to internal command.

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

**Parameters**

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

References [Append\(\)](#).

**11.25.3.3 Append()** [3/3]

```
void CPLODBCStatement::Append (
    double dfValue )
```

Append to internal command.

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

**Parameters**

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

References [Append\(\)](#).

**11.25.3.4 AppendEscaped()**

```
void CPLODBCStatement::AppendEscaped (
    const char * pszText )
```

Append text to internal command.

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

**Parameters**

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

References [Append\(\)](#), [CPLFree](#), and [VSIMalloc\(\)](#).

**11.25.3.5 Appendf()**

```
int CPLODBCStatement::Appendf (
    const char * pszFormat,
    ... )
```



Append to internal command.

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

#### Parameters

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

#### Returns

FALSE if formatting fails due to result being too large.

References Append().

#### 11.25.3.6 Clear()

```
void CPLODBCStatement::Clear ( )
```

Clear internal command text and result set definitions.

References ClearColumnData(), CPLFree, and CSLDestroy().

Referenced by ExecuteSQL(), and ~CPLODBCStatement().

#### 11.25.3.7 ClearColumnData()

```
void CPLODBCStatement::ClearColumnData ( )
```

ClearColumnData

References CPLFree.

Referenced by Clear(), and Fetch().

#### 11.25.3.8 CollectResultsInfo()

```
int CPLODBCStatement::CollectResultsInfo ( )
```

CollectResultsInfo

References CPLCalloc(), CPLDebug(), and CPLStrdup().

Referenced by ExecuteSQL(), GetPrimaryKeys(), and GetTables().

#### 11.25.3.9 DumpResult()

```
void CPLODBCStatement::DumpResult (
    FILE * fp,
    int bShowSchema = FALSE )
```

Dump resultset to file.

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

## Parameters

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

References [Fetch\(\)](#), [GetColCount\(\)](#), [GetColData\(\)](#), [GetColName\(\)](#), [GetColNullable\(\)](#), [GetColPrecision\(\)](#), [GetColSize\(\)](#), [GetColType\(\)](#), and [GetTypeName\(\)](#).

11.25.3.10 **ExecuteSQL()**

```
int CPLODBCStatement::ExecuteSQL (
    const char * pszStatement = nullptr )
```

Execute an SQL statement.

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

## Parameters

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

## Returns

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

References [Append\(\)](#), [Clear\(\)](#), [CPLODBCSession::ClearTransaction\(\)](#), [CollectResultsInfo\(\)](#), and [CPLODBCSession::IsInTransaction\(\)](#).

11.25.3.11 **Fetch()**

```
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 occurred if **Fetch()** (p. ??) fails.

**Parameters**

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

**Returns**

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

References ClearColumnData(), and CPLError().

Referenced by DumpResult().

**11.25.3.12 GetColColumnDef()**

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

Fetch a column default value.

Returns the default value of a column.

**Parameters**

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

**Returns**

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

**11.25.3.13 GetColCount()**

```
int CPLODBCStatement::GetColCount ( )
```

Fetch the resultset column count.

**Returns**

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

Referenced by DumpResult().

**11.25.3.14 GetColData()** [1/2]

```
const char * CPODBCStatement::GetColData (
    int iCol,
    const char * pszDefault = nullptr )
```

Fetch column data.

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

**Parameters**

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

**Returns**

pointer to internal column data or NULL on failure.

Referenced by DumpResult(), and GetColData().

**11.25.3.15 GetColData()** [2/2]

```
const char * CPODBCStatement::GetColData (
    const char * pszColName,
    const char * pszDefault = nullptr )
```

Fetch column data.

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

**Parameters**

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

**Returns**

pointer to internal column data or NULL on failure.

References GetColData(), and GetColId().

**11.25.3.16 GetColDataLength()**

```
int CPODBCStatement::GetColDataLength (
    int iCol )
```

GetColDataLength

### 11.25.3.17 GetColId()

```
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

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

#### Returns

the column index, or -1 if not found.

References EQUAL.

Referenced by GetColData().

### 11.25.3.18 GetColName()

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

Fetch a column name.

#### Parameters

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

#### Returns

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

Referenced by DumpResult().

### 11.25.3.19 GetColNullable()

```
short CPLODBCStatement::GetColNullable (
    int iCol )
```

Fetch the column nullability.

**Parameters**

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

**Returns**

TRUE if the column may contains or FALSE otherwise.

Referenced by DumpResult().

**11.25.3.20 GetColPrecision()**

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

Fetch the column precision.

**Parameters**

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

**Returns**

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

Referenced by DumpResult().

**11.25.3.21 GetColSize()**

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

Fetch the column width.

**Parameters**

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

**Returns**

column width, zero for unknown width columns.

Referenced by DumpResult().

## 11.25.3.22 GetColType()

```
short CPODBCStatement::GetColType (
    int iCol )
```

Fetch a column data type.

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

## Parameters

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

## Returns

type code or -1 if the column is illegal.

Referenced by DumpResult().

## 11.25.3.23 GetColTypeName()

```
const char * CPODBCStatement::GetColTypeName (
    int iCol )
```

Fetch a column data type name.

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

## Parameters

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

## Returns

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

## 11.25.3.24 GetColumns()

```
int CPODBCStatement::GetColumns (
    const char * pszTable,
    const char * pszCatalog = nullptr,
    const char * pszSchema = nullptr )
```

Fetch column definitions for a table.

The SQLColumn() method is used to fetch the definitions for the columns of a table (or other queryable 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

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

#### Returns

TRUE on success or FALSE on failure.

References CPLODBCSession::ClearTransaction(), CPLAlloc(), CPLStrdup(), and CPLODBCSession::IsInTransaction().

#### 11.25.3.25 GetCommand()

```
const char* CPLODBCStatement::GetCommand ( ) [inline]
```

Return statement string

#### 11.25.3.26 GetPrimaryKeys()

```
int CPLODBCStatement::GetPrimaryKeys (
    const char * pszTable,
    const char * pszCatalog = nullptr,
    const char * pszSchema = nullptr )
```

Fetch primary keys for a table.

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

#### Parameters

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

#### Returns

TRUE on success or FALSE on failure.



References CPODBCSession::ClearTransaction(), CollectResultsInfo(), and CPODBCSession::IsInTransaction().

### 11.25.3.27 GetRowCountAffected()

```
int CPODBCStatement::GetRowCountAffected ( )
```

GetRowCountAffected

### 11.25.3.28 GetStatement()

```
HSTMT CPODBCStatement::GetStatement ( ) [inline]
```

Return statement handle

### 11.25.3.29 GetTables()

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

Fetch tables in database.

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

#### Parameters

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

#### Returns

TRUE on success or FALSE on failure.

References CPODBCSession::ClearTransaction(), CollectResultsInfo(), CPLDebug(), and CPODBCSession::IsInTransaction().

### 11.25.3.30 GetTypeMapping()

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

Get appropriate C data type for SQL column type.

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

**Parameters**

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

**Returns**

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

**11.25.3.31 GetTypeNames()**

```
CPLString CPODBCStatement::GetTypeNames (
    int nTypeCode ) [static]
```

Get name for SQL column type.

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

**Parameters**

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

**Returns**

internal string, "UNKNOWN" if code not recognised.

References CPLString::Printf().

Referenced by DumpResult().

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

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

**11.26 CPLRectObj Struct Reference**

```
#include <cpl_quad_tree.h>
```

**Public Attributes**

- double **minx**
- double **miny**
- double **maxx**
- double **maxy**

### 11.26.1 Detailed Description

Describe a rectangle

### 11.26.2 Member Data Documentation

#### 11.26.2.1 maxx

```
double CPLRectObj::maxx
```

Maximum x

#### 11.26.2.2 maxy

```
double CPLRectObj::maxy
```

Maximum y

#### 11.26.2.3 minx

```
double CPLRectObj::minx
```

Minimum x

#### 11.26.2.4 miny

```
double CPLRectObj::miny
```

Minimum y

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

- `cpl_quad_tree.h`

## 11.27 CPLSharedFileInfo Struct Reference

```
#include <cpl_conv.h>
```

## Public Attributes

- FILE \* **fp**
- int **nRefCount**
- int **bLarge**
- char \* **pszFilename**
- char \* **pszAccess**

### 11.27.1 Detailed Description

Information on a shared file

### 11.27.2 Member Data Documentation

#### 11.27.2.1 bLarge

```
int CPLSharedFileInfo::bLarge
```

Whether fp must be interpreted as VSIFILE\*

#### 11.27.2.2 fp

```
FILE* CPLSharedFileInfo::fp
```

File pointer

Referenced by CPLCloseShared().

#### 11.27.2.3 nRefCount

```
int CPLSharedFileInfo::nRefCount
```

Reference counter

#### 11.27.2.4 pszAccess

```
char* CPLSharedFileInfo::pszAccess
```

Access mode

### 11.27.2.5 pszFilename

```
char* CPLSharedFileInfo::pszFilename
```

Filename

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

- `cpl_conv.h`

## 11.28 CPLSharedFileInfoExtra Struct Reference

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

- `cpl_conv.cpp`

## 11.29 CPLStdCallThreadInfo Struct Reference

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

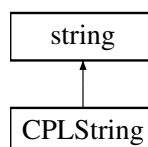
- `cpl_multiproc.cpp`

## 11.30 CPLString Class Reference

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

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

Inheritance diagram for CPLString:



## Public Member Functions

- **CPLStringT** (void)
- **CPLStringT** (const std::string &oStr)
- **CPLStringT** (const char \*pszStr)
- **CPLStringT** (const char \*pszStr, size\_t n)
- **operator const char \*** (void) const
- char & **operator[]** (std::string::size\_type i)
- const char & **operator[]** (std::string::size\_type i) const
- char & **operator[]** (int i)
- const char & **operator[]** (int i) const
- void **Clear** ()
- void **Seize** (char \*pszValue)
- CPLSTRING\_DLL **CPLString** & **Printf** (const char \*pszFormat,...)
- CPLSTRING\_DLL **CPLString** & **vPrintf** (const char \*pszFormat, va\_list args)
- CPLSTRING\_DLL **CPLString** & **FormatC** (double dfValue, const char \*pszFormat=NULLPTR)
- CPLSTRING\_DLL **CPLString** & **Trim** ()
- CPLSTRING\_DLL **CPLString** & **Recode** (const char \*pszSrcEncoding, const char \*pszDstEncoding)
- CPLSTRING\_DLL **CPLString** & **replaceAll** (const std::string &osBefore, const std::string &osAfter)
- CPLSTRING\_DLL **CPLString** & **replaceAll** (const std::string &osBefore, char chAfter)
- CPLSTRING\_DLL **CPLString** & **replaceAll** (char chBefore, const std::string &osAfter)
- CPLSTRING\_DLL **CPLString** & **replaceAll** (char chBefore, char chAfter)
- CPLSTRING\_DLL size\_t **ifind** (const std::string &str, size\_t pos=0) const
- CPLSTRING\_DLL size\_t **ifind** (const char \*s, size\_t pos=0) const
- CPLSTRING\_DLL **CPLString** & **toupper** (void)
- CPLSTRING\_DLL **CPLString** & **tolower** (void)
- CPLSTRING\_DLL bool **endsWith** (const std::string &osStr) const

### 11.30.1 Detailed Description

Convenient string class based on std::string.

### 11.30.2 Member Function Documentation

#### 11.30.2.1 Clear()

```
void CPLString::Clear ( ) [inline]
```

Clear the string

Referenced by Seize().

#### 11.30.2.2 CPLStringT() [1/4]

```
CPLString::CPLStringT (
    void ) [inline]
```

Constructor

## 11.30.2.3 CPLStringT() [2/4]

```
CPLString::CPLStringT (
    const std::string & oStr ) [inline]
```

Constructor

## 11.30.2.4 CPLStringT() [3/4]

```
CPLString::CPLStringT (
    const char * pszStr ) [inline]
```

Constructor

## 11.30.2.5 CPLStringT() [4/4]

```
CPLString::CPLStringT (
    const char * pszStr,
    size_t n ) [inline]
```

Constructor

## 11.30.2.6 endsWith()

```
bool CPLString::endsWith (
    const std::string & osStr ) const
```

Returns whether the string ends with another string

Parameters

<i>osStr</i>	other string.
--------------	---------------

Returns

true if the string ends wit osStr.

## 11.30.2.7 FormatC()

```
CPLString & CPLString::FormatC (
    double dfValue,
    const char * pszFormat = nullptr )
```

Format double in C locale.

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

**Parameters**

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

**Returns**

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

References CPLsprintf().

**11.30.2.8 ifind()** [1/2]

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

Case insensitive find() alternative.

**Parameters**

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

**Returns**

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

**Since**

GDAL 1.9.0

Referenced by CPLURLAddKVP(), and CPLURLGetValue().

**11.30.2.9 ifind()** [2/2]

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

Case insensitive find() alternative.



**Parameters**

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

**Returns**

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

**Since**

GDAL 1.9.0

References `EQUALN`, and `tolower()`.

**11.30.2.10 operator const char \*()**

```
CPLString::operator const char * (
    void ) const [inline]
```

Return string as zero terminated character array

**11.30.2.11 operator[]() [1/4]**

```
char& CPLString::operator[] (
    std::string::size_type i ) [inline]
```

Return character at specified index

**11.30.2.12 operator[]() [2/4]**

```
const char& CPLString::operator[] (
    std::string::size_type i ) const [inline]
```

Return character at specified index

**11.30.2.13 operator[]() [3/4]**

```
char& CPLString::operator[] (
    int i ) [inline]
```

Return character at specified index

**11.30.2.14 operator[]()** [4/4]

```
const char& CPLString::operator[] (
    int i ) const [inline]
```

Return character at specified index

**11.30.2.15 Printf()**

```
CPLString & CPLString::Printf (
    const char * pszFormat,
    ... )
```

Assign the content of the string using sprintf()

References vPrintf().

Referenced by CPLGenerateTempFilename(), OGRSimpleCurve::exportToWkt(), CPLODBCStatement::GetType↵Name(), GOA2GetAuthorizationURL(), GOA2GetRefreshToken(), and OGRSpatialReference::SetVertCS().

**11.30.2.16 Recode()**

```
CPLString & CPLString::Recode (
    const char * pszSrcEncoding,
    const char * pszDstEncoding )
```

Recode the string

References CPL\_ENC\_UTF8, CPLFree, and CPLRecode().

**11.30.2.17 replaceAll()** [1/4]

```
CPLString & CPLString::replaceAll (
    const std::string & osBefore,
    const std::string & osAfter )
```

Replace all occurrences of osBefore with osAfter.

Referenced by replaceAll().

**11.30.2.18 replaceAll()** [2/4]

```
CPLString & CPLString::replaceAll (
    const std::string & osBefore,
    char chAfter )
```

Replace all occurrences of *osBefore* with *chAfter*.

References `replaceAll()`.

**11.30.2.19 replaceAll()** [3/4]

```
CPLString & CPLString::replaceAll (
    char chBefore,
    const std::string & osAfter )
```

Replace all occurrences of *chBefore* with *osAfter*.

References `replaceAll()`.

**11.30.2.20 replaceAll()** [4/4]

```
CPLString & CPLString::replaceAll (
    char chBefore,
    char chAfter )
```

Replace all occurrences of *chBefore* with *chAfter*.

References `replaceAll()`.

**11.30.2.21 Seize()**

```
void CPLString::Seize (
    char * pszValue ) [inline]
```

Assign specified string and take ownership of it (assumed to be allocated with **CPLMalloc()** (p. ??)). NULL can be safely passed to clear the string.

References `Clear()`, and `CPLFree`.

Referenced by `GOA2GetAuthorizationURL()`, and `GOA2GetRefreshToken()`.

#### 11.30.2.22 tolower()

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

Convert to lower case in place.

Referenced by ifind().

#### 11.30.2.23 toupper()

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

Convert to upper case in place.

Referenced by OGRFeatureDefn::ComputeMapForSetFrom().

#### 11.30.2.24 Trim()

```
CPLString & CPLString::Trim ( )
```

Trim white space.

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

##### Returns

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

#### 11.30.2.25 vPrintf()

```
CPLString & CPLString::vPrintf (
    const char * pszFormat,
    va_list args )
```

Assign the content of the string using vsprintf()

References CPLError(), CPLMalloc(), and CPLvsprintf().

Referenced by CPLOPrintf(), CPLOvPrintf(), CPLVASprintf(), CSLAppendPrintf(), Printf(), and VSIFPrintfL().

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

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

## 11.31 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=TRUE)
- **CPLStringList** ( **CSLConstList** papszList)
- **CPLStringList** (const **CPLStringList** &oOther)  
*Copy constructor.*
- **CPLStringList & Clear** ()
- int **size** () const
- int **Count** () const
- bool **empty** () 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 **FindString** (const char \*pszTarget) const
- int **PartialFindString** (const char \*pszNeedle) const
- int **FindName** (const char \*pszName) const
- bool **FetchBool** (const char \*pszKey, bool bDefault) 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=TRUE)
- **CPLStringList & operator=** (char \*\*papszListIn)
- **CPLStringList & operator=** (const **CPLStringList** &oOther)
- **CPLStringList & operator=** ( **CSLConstList** papszListIn)
- char \* **operator[]** (int i)
- char \* **operator[]** (size\_t i)
- const char \* **operator[]** (int i) const
- const char \* **operator[]** (size\_t i) const
- const char \* **operator[]** (const char \*pszKey) const
- char \*\* **List** ()
- **CSLConstList List** () const
- char \*\* **StealList** ()
- **CPLStringList & Sort** ()
- int **IsSorted** () const
- **operator char \*\*** (void)
- **operator CSLConstList** (void) const

### 11.31.1 Detailed Description

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

## 11.31.2 Constructor & Destructor Documentation

### 11.31.2.1 CPLStringList() [1/2]

```
CPLStringList::CPLStringList (
    char ** papszListIn,
    int bTakeOwnership = TRUE )
```

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

#### Parameters

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

References Assign().

### 11.31.2.2 CPLStringList() [2/2]

```
CPLStringList::CPLStringList (
    CSLConstList papszListIn )
```

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

The input list is copied.

#### Parameters

<i>papszListIn</i>	the NULL terminated list of strings to ingest.
--------------------	--

References Assign(), and CSLDuplicate().

## 11.31.3 Member Function Documentation

### 11.31.3.1 AddNameValue()

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

Add a name=value entry to the list.

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

#### Parameters

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

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

Referenced by SetNameValue().

#### 11.31.3.2 AddString()

```
CPLStringList & CPLStringList::AddString (
    const char * pszNewString )
```

Add a string to the list.

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

#### Parameters

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

References AddStringDirectly(), and CPLStrdup().

Referenced by CSLTokenizeString2(), GOA2GetAccessTokenFromCloudEngineVM(), GOA2GetRefreshToken(), OGRFeature::FieldValue::operator=(), and VSIReadDirRecursive().

#### 11.31.3.3 AddStringDirectly()

```
CPLStringList & CPLStringList::AddStringDirectly (
    char * pszNewString )
```

Add a string to the list.

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

#### Parameters

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

References Count().

Referenced by AddNameValue(), and AddString().

#### 11.31.3.4 Assign()

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

Assign a list of strings.

##### Parameters

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

##### Returns

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

References Clear().

Referenced by CPLStringList(), CSLTokenizeString2(), and operator=().

#### 11.31.3.5 Clear()

```
CPLStringList & CPLStringList::Clear ( )
```

Clear the string list.

References CSLDestroy().

Referenced by Assign().

#### 11.31.3.6 Count()

```
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().



### 11.31.3.7 empty()

```
bool CPLStringList::empty ( ) const [inline]
```

Return whether the list is empty.

### 11.31.3.8 FetchBool()

```
bool CPLStringList::FetchBool (
    const char * pszKey,
    bool bDefault ) const
```

Check for boolean key value.

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

#### Parameters

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

#### Returns

true or false

References CPLTestBool(), and FetchNameValue().

Referenced by FetchBoolean().

### 11.31.3.9 FetchBoolean()

```
int CPLStringList::FetchBoolean (
    const char * pszKey,
    int bDefault ) const
```

DEPRECATED: Check for boolean key value.

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

#### Parameters

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

**Returns**

TRUE or FALSE

References FetchBool().

**11.31.3.10 FetchNameValue()**

```
const char * CPLStringList::FetchNameValue (
    const char * pszName ) const
```

Fetch value associated with this key name.

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

**Parameters**

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

**Returns**

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

References CPLAssert, and FindName().

Referenced by FetchBool(), and FetchNameValueDef().

**11.31.3.11 FetchNameValueDef()**

```
const char * CPLStringList::FetchNameValueDef (
    const char * pszName,
    const char * pszDefault ) const
```

Fetch value associated with this key name.

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

**Parameters**

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

**Returns**

the corresponding value or the passed default if not found.

References `FetchNameValue()`.

#### 11.31.3.12 FindName()

```
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()** (p. ??) or **PartialFindString()** (p. ??) for searches not based on name=value pairs.

##### Parameters

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

##### Returns

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

References `CSLFindName()`, `EQUALN`, and `IsSorted()`.

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

#### 11.31.3.13 FindString()

```
int CPLStringList::FindString (
    const char * pszTarget ) const [inline]
```

Return index of `pszTarget` in the list, or -1

References `CSLFindString()`.

#### 11.31.3.14 InsertString()

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

Insert into the list at identified location.

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

## Parameters

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

References CPLStrdup().

## 11.31.3.15 InsertStringDirectly()

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

Insert into the list at identified location.

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

## Parameters

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

References Count(), and CPLError().

Referenced by AddNameValue().

## 11.31.3.16 IsSorted()

```
int CPLStringList::IsSorted ( ) const [inline]
```

Returns whether the list is sorted

Referenced by AddNameValue(), and FindName().

## 11.31.3.17 List() [1/2]

```
char** CPLStringList::List ( ) [inline]
```

Return list. Ownership remains to the object

Referenced by CSLTokenizeString2(), GOA2Manager::GetBearer(), and OGRFeature::FieldValue::operator=().

**11.31.3.18 List()** [2/2]

```
CSLConstList CPLStringList::List ( ) const [inline]
```

Return list. Ownership remains to the object

**11.31.3.19 operator char \*\*()**

```
CPLStringList::operator char ** (
    void ) [inline]
```

Return lists

**11.31.3.20 operator CSLConstList()**

```
CPLStringList::operator CSLConstList (
    void ) const [inline]
```

Return lists

**11.31.3.21 operator=()** [1/3]

```
CPLStringList & CPLStringList::operator= (
    char ** papszListIn ) [inline]
```

Assignment operator

**11.31.3.22 operator=()** [2/3]

```
CPLStringList & CPLStringList::operator= (
    const CPLStringList & oOther )
```

Assignment operator

References Assign().

**11.31.3.23 operator=()** [3/3]

```
CPLStringList& CPLStringList::operator= (
    CSLConstList papszListIn )
```

Assignment operator

**11.31.3.24 operator[]()** [1/5]

```
char * CPLStringList::operator[] (
    int i )
```

Return string at specified index

Fetch entry "i".

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

**Parameters**

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

**Returns**

selected entry in the list.

References Count().

**11.31.3.25 operator[]()** [2/5]

```
char* CPLStringList::operator[] (
    size_t i ) [inline]
```

Return string at specified index

**11.31.3.26 operator[]()** [3/5]

```
const char * CPLStringList::operator[] (
    int i ) const
```

Return string at specified index

References Count().

**11.31.3.27 operator[]()** [4/5]

```
const char* CPLStringList::operator[] (
    size_t i ) const [inline]
```

Return string at specified index

**11.31.3.28 operator[]()** [5/5]

```
const char* CPLStringList::operator[] (
    const char * pszKey ) const [inline]
```

Return value corresponding to pszKey, or nullptr

**11.31.3.29 PartialFindString()**

```
int CPLStringList::PartialFindString (
    const char * pszNeedle ) const [inline]
```

Return index of pszTarget in the list (using partial search), or -1

References CSLPartialFindString().

**11.31.3.30 SetNameValue()**

```
CPLStringList & CPLStringList::SetNameValue (
    const char * pszKey,
    const char * pszValue )
```

Set name=value entry in the list.

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

**Parameters**

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

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

**11.31.3.31 size()**

```
int CPLStringList::size ( ) const [inline]
```

Return size of list

**11.31.3.32 Sort()**

```
CPLStringList & CPLStringList::Sort ( )
```

Sort the entries in the list and mark list sorted.

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

References Count().

### 11.31.3.33 StealList()

```
char ** CPLStringList::StealList ( )
```

Seize ownership of underlying string array.

This method is similar to **List()** (p. ??), 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

## 11.32 CPLVirtualMem Struct Reference

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

- cpl\_virtualmem.cpp

## 11.33 CPLWorkerThreadPool Class Reference

```
#include <cpl_worker_thread_pool.h>
```

### Public Member Functions

- **CPLWorkerThreadPool** ()
- **~CPLWorkerThreadPool** ()
- bool **Setup** (int nThreads, CPLThreadFunc pfnInitFunc, void \*\*pasInitData)
- bool **SubmitJob** (CPLThreadFunc pfnFunc, void \*pData)
- bool **SubmitJobs** (CPLThreadFunc pfnFunc, const std::vector< void \*> &apData)
- void **WaitCompletion** (int nMaxRemainingJobs=0)
- int **GetThreadCount** () const

### 11.33.1 Detailed Description

Pool of worker threads



## 11.33.2 Constructor & Destructor Documentation

### 11.33.2.1 CPLWorkerThreadPool()

```
CPLWorkerThreadPool::CPLWorkerThreadPool ( )
```

Instantiate a new pool of worker threads.

The pool is in an uninitialized state after this call. The **Setup()** (p. ??) method must be called.

### 11.33.2.2 ~CPLWorkerThreadPool()

```
CPLWorkerThreadPool::~~CPLWorkerThreadPool ( )
```

Destroys a pool of worker threads.

Any still pending job will be completed before the destructor returns.

References CPLListDestroy(), and WaitCompletion().

## 11.33.3 Member Function Documentation

### 11.33.3.1 GetThreadCount()

```
int CPLWorkerThreadPool::GetThreadCount ( ) const [inline]
```

Return the number of threads setup

### 11.33.3.2 Setup()

```
bool CPLWorkerThreadPool::Setup (
    int nThreads,
    CPLThreadFunc pfnInitFunc,
    void ** pasInitData )
```

Setup the pool.

#### Parameters

<i>nThreads</i>	Number of threads to launch
<i>pfnInitFunc</i>	Initialization function to run in each thread. May be NULL
<i>pasInitData</i>	Array of initialization data. Its length must be nThreads, or it should be NULL.

**Returns**

true if initialization was successful.

References CPLAssert.

**11.33.3.3 SubmitJob()**

```
bool CPLWorkerThreadPool::SubmitJob (
    CPLThreadFunc pfnFunc,
    void * pData )
```

Queue a new job.

**Parameters**

<i>pfnFunc</i>	Function to run for the job.
<i>pData</i>	User data to pass to the job function.

**Returns**

true in case of success.

References CPLAssert, CPLDebug(), CPLFree, \_CPLList::pData, \_CPLList::pNext, VSI\_MALLOC\_VERBOSE, and VSIFree().

**11.33.3.4 SubmitJobs()**

```
bool CPLWorkerThreadPool::SubmitJobs (
    CPLThreadFunc pfnFunc,
    const std::vector< void *> & apData )
```

Queue several jobs

**Parameters**

<i>pfnFunc</i>	Function to run for the job.
<i>apData</i>	User data instances to pass to the job function.

**Returns**

true in case of success.

References CPLAssert, CPLDebug(), CPLFree, \_CPLList::pData, \_CPLList::pNext, VSI\_MALLOC\_VERBOSE, and VSIFree().

## 11.33.3.5 WaitCompletion()

```
void CPLWorkerThreadPool::WaitCompletion (
    int nMaxRemainingJobs = 0 )
```

Wait for completion of part or whole jobs.

## Parameters

<i>nMaxRemainingJobs</i>	Maximum number of pendings jobs that are allowed in the queue after this method has completed. Might be 0 to wait for all jobs.
--------------------------	---

Referenced by `~CPLWorkerThreadPool()`.

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

- `cpl_worker_thread_pool.h`
- `cpl_worker_thread_pool.cpp`

## 11.34 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.*

## 11.34.1 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 hierarchical tree structure for a document can be represented as a tree of **CPLXMLNode** (p. ??) structures.

## 11.34.2 Member Data Documentation

### 11.34.2.1 eType

**CPLXMLNodeType** CPLXMLNode::eType

Node type.

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

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

### 11.34.2.2 psChild

**struct CPLXMLNode\*** CPLXMLNode::psChild

Child node.

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

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

### 11.34.2.3 psNext

**struct CPLXMLNode\*** CPLXMLNode::psNext

Next sibling.

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

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

## 11.34.2.4 pszValue

```
char* CPLXMLNode::pszValue
```

Node value.

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

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

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

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

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

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

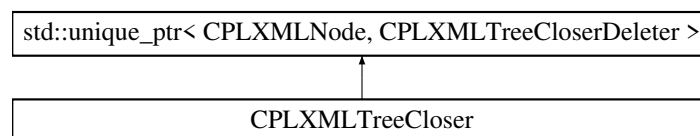
The documentation for this struct was generated from the following file:

- **cpl\_minixml.h**

## 11.35 CPLXMLTreeCloser Class Reference

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

Inheritance diagram for CPLXMLTreeCloser:



### Public Member Functions

- **CPLXMLTreeCloser** ( **CPLXMLNode** \*data)
- **CPLXMLNode** \* **getDocumentElement** ()

### 11.35.1 Detailed Description

Manage a tree of XML nodes so that all nodes are freed when the instance goes out of scope. Only the top level node should be in a **CPLXMLTreeCloser** (p. ??).

## 11.35.2 Constructor & Destructor Documentation

### 11.35.2.1 CPLXMLTreeCloser()

```
CPLXMLTreeCloser::CPLXMLTreeCloser (
    CPLXMLNode * data ) [inline], [explicit]
```

Constructor

## 11.35.3 Member Function Documentation

### 11.35.3.1 getElement()

```
CPLXMLNode * CPLXMLTreeCloser::getElement ( )
```

Returns a pointer to the document (root) element

Returns

the node pointer

References CXT\_Element, CPLXMLNode::eType, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

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

- **cpl\_minixml.h**
- cpl\_minixml.cpp

## 11.36 CPLZip Struct Reference

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

- cpl\_minizip\_zip.cpp

## 11.37 ctb Struct Reference

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

- cpl\_csv.cpp

## 11.38 curfile\_info Struct Reference

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

- cpl\_minizip\_zip.cpp

## 11.39 CurlProcessData Struct Reference

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

- cpl\_http.cpp

## 11.40 DefaultCSVFileNameTLS Struct Reference

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

- cpl\_csv.cpp

## 11.41 errHandler Struct Reference

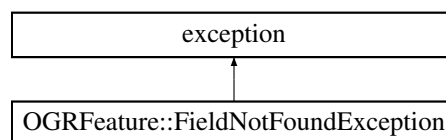
The documentation for this struct was generated from the following file:

- cpl\_error.cpp

## 11.42 OGRFeature::FieldNotFoundException Class Reference

```
#include <ogr_feature.h>
```

Inheritance diagram for OGRFeature::FieldNotFoundException:



### 11.42.1 Detailed Description

Exception raised by **operator[]**(const char\*) (p. ??) when a field is not found.

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

- ogr\_feature.h

## 11.43 OGRFeature::FieldValue Class Reference

```
#include <ogr_feature.h>
```

### Classes

- struct **Private**

### Public Member Functions

- **FieldValue** & **operator=** (const **FieldValue** &oOther)
- **FieldValue** & **operator=** (int nVal)
- **FieldValue** & **operator=** ( **GIntBig** nVal)
- **FieldValue** & **operator=** (double dfVal)
- **FieldValue** & **operator=** (const char \*pszVal)
- **FieldValue** & **operator=** (const std::string &osVal)
- **FieldValue** & **operator=** (const std::vector< int > &oArray)
- **FieldValue** & **operator=** (const std::vector< **GIntBig** > &oArray)
- **FieldValue** & **operator=** (const std::vector< double > &oArray)
- **FieldValue** & **operator=** (const std::vector< std::string > &oArray)
- **FieldValue** & **operator=** ( **CSLConstList** papszValues)
- void **SetNull** ()
- void **clear** ()
- void **Unset** ()
- void **SetDateTime** (int nYear, int nMonth, int nDay, int nHour=0, int nMinute=0, float fSecond=0.f, int nTZ↵  
Flag=0)
- int **GetIndex** () const
- const **OGRFieldDefn** \* **GetDefn** () const
- const char \* **GetName** () const
- **OGRFieldType** **GetType** () const
- **OGRFieldSubType** **GetSubType** () const
- bool **empty** () const
- bool **IsUnset** () const
- bool **IsNull** () const
- const **OGRField** \* **GetRawValue** () const
- int **GetInteger** () const
- **GIntBig** **GetInteger64** () const
- double **GetDouble** () const
- const char \* **GetString** () const
- bool **GetDateTime** (int \*pnYear, int \*pnMonth, int \*pnDay, int \*pnHour, int \*pnMinute, float \*pfSecond, int  
\*pnTZFlag) const
- **operator int** () const
- **operator GIntBig** () const
- **operator double** () const
- **operator const char \*** () const
- **operator const std::vector< int > &** () const
- **operator const std::vector< GIntBig > &** () const
- **operator const std::vector< double > &** () const
- **operator const std::vector< std::string > &** () const
- **operator CSLConstList** () const
- int **GetAsInteger** () const
- **GIntBig** **GetAsInteger64** () const
- double **GetAsDouble** () const
- const char \* **GetAsString** () const
- const std::vector< int > & **GetAsIntegerList** () const
- const std::vector< **GIntBig** > & **GetAsInteger64List** () const
- const std::vector< double > & **GetAsDoubleList** () const
- const std::vector< std::string > & **GetAsStringList** () const



## Friends

- class **OGRFeature**

### 11.43.1 Detailed Description

Field value.

### 11.43.2 Member Function Documentation

#### 11.43.2.1 clear()

```
void OGRFeature::FieldValue::clear ( )
```

Unset the field.

#### 11.43.2.2 empty()

```
bool OGRFeature::FieldValue::empty ( ) const [inline]
```

Return whether the field value is unset/empty.

#### 11.43.2.3 GetAsDouble()

```
double OGRFeature::FieldValue::GetAsDouble ( ) const
```

Return the field value as double, with potential conversion

References OGRFeature::GetFieldAsDouble().

#### 11.43.2.4 GetAsDoubleList()

```
const std::vector< double > & OGRFeature::FieldValue::GetAsDoubleList ( ) const
```

Return the field value as double list, with potential conversion

References OGRFeature::GetFieldAsDoubleList().

#### 11.43.2.5 GetAsInteger()

```
int OGRFeature::FieldValue::GetAsInteger ( ) const
```

Return the field value as integer, with potential conversion

References OGRFeature::GetFieldAsInteger().

#### 11.43.2.6 GetAsInteger64()

```
GIntBig OGRFeature::FieldValue::GetAsInteger64 ( ) const
```

Return the field value as 64-bit integer, with potential conversion

References OGRFeature::GetFieldAsInteger64().

#### 11.43.2.7 GetAsInteger64List()

```
const std::vector< GIntBig > & OGRFeature::FieldValue::GetAsInteger64List ( ) const
```

Return the field value as 64-bit integer list, with potential conversion

References OGRFeature::GetFieldAsInteger64List().

#### 11.43.2.8 GetAsIntegerList()

```
const std::vector< int > & OGRFeature::FieldValue::GetAsIntegerList ( ) const
```

Return the field value as integer list, with potential conversion

References OGRFeature::GetFieldAsIntegerList().

#### 11.43.2.9 GetAsString()

```
const char * OGRFeature::FieldValue::GetAsString ( ) const
```

Return the field value as string, with potential conversion

References OGRFeature::GetFieldAsString().

#### 11.43.2.10 GetAsStringList()

```
const std::vector< std::string > & OGRFeature::FieldValue::GetAsStringList ( ) const
```

Return the field value as string list, with potential conversion

References OGRFeature::GetFieldAsStringList().

#### 11.43.2.11 GetDateTime()

```
bool OGRFeature::FieldValue::GetDateTime (
    int * pnYear,
    int * pnMonth,
    int * pnDay,
    int * pnHour,
    int * pnMinute,
    float * pfSecond,
    int * pnTZFlag ) const
```

Return the date/time/datetime value.

Referenced by operator=().

#### 11.43.2.12 GetDefn()

```
const OGRFieldDefn * OGRFeature::FieldValue::GetDefn ( ) const
```

Return field definition.

#### 11.43.2.13 GetDouble()

```
double OGRFeature::FieldValue::GetDouble ( ) const [inline]
```

Return the double value. Only use that method if and only if **GetType()** (p. ??) == OFTReal.

#### 11.43.2.14 GetIndex()

```
int OGRFeature::FieldValue::GetIndex ( ) const
```

Return field index.

#### 11.43.2.15 GetInteger()

```
int OGRFeature::FieldValue::GetInteger ( ) const [inline]
```

Return the integer value. Only use that method if and only if **GetType()** (p. ??) == OFTInteger.

#### 11.43.2.16 GetInteger64()

```
GIntBig OGRFeature::FieldValue::GetInteger64 ( ) const [inline]
```

Return the 64-bit integer value. Only use that method if and only if **GetType()** (p. ??) == OFTInteger64.

#### 11.43.2.17 GetName()

```
const char* OGRFeature::FieldValue::GetName ( ) const [inline]
```

Return field name.

#### 11.43.2.18 GetRawValue()

```
const OGRField * OGRFeature::FieldValue::GetRawValue ( ) const
```

Return the raw field value

#### 11.43.2.19 GetString()

```
const char* OGRFeature::FieldValue::GetString ( ) const [inline]
```

Return the string value. Only use that method if and only if **GetType()** (p. ??) == OFTString.

#### 11.43.2.20 GetSubType()

```
OGRFieldSubType OGRFeature::FieldValue::GetSubType ( ) const [inline]
```

Return field subtype.

#### 11.43.2.21 GetType()

```
OGRFieldType OGRFeature::FieldValue::GetType ( ) const [inline]
```

Return field type.

#### 11.43.2.22 IsNull()

```
bool OGRFeature::FieldValue::IsNull ( ) const
```

Return whether the field value is null.

#### 11.43.2.23 IsUnset()

```
bool OGRFeature::FieldValue::IsUnset ( ) const
```

Return whether the field value is unset/empty.

**11.43.2.24 operator const char \*()**

```
OGRFeature::FieldValue::operator const char * ( ) const [inline]
```

Return the field value as string, with potential conversion

**11.43.2.25 operator const std::vector< double > &()**

```
OGRFeature::FieldValue::operator const std::vector< double > & ( ) const [inline]
```

Return the field value as double list, with potential conversion

**11.43.2.26 operator const std::vector< GIntBig > &()**

```
OGRFeature::FieldValue::operator const std::vector< GIntBig > & ( ) const [inline]
```

Return the field value as 64-bit integer list, with potential conversion

**11.43.2.27 operator const std::vector< int > &()**

```
OGRFeature::FieldValue::operator const std::vector< int > & ( ) const [inline]
```

Return the field value as integer list, with potential conversion

**11.43.2.28 operator const std::vector< std::string > &()**

```
OGRFeature::FieldValue::operator const std::vector< std::string > & ( ) const [inline]
```

Return the field value as string list, with potential conversion

**11.43.2.29 operator CSLConstList()**

```
OGRFeature::FieldValue::operator CSLConstList ( ) const
```

Return the field value as string list, with potential conversion

References OGRFeature::GetFieldAsStringList().

**11.43.2.30 operator double()**

```
OGRFeature::FieldValue::operator double ( ) const [inline]
```

Return the field value as double, with potential conversion

**11.43.2.31 operator GIntBig()**

```
OGRFeature::FieldValue::operator GIntBig ( ) const [inline]
```

Return the field value as 64-bit integer, with potential conversion

**11.43.2.32 operator int()**

```
OGRFeature::FieldValue::operator int ( ) const [inline]
```

Return the field value as integer, with potential conversion

**11.43.2.33 operator=()** [1/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    const FieldValue & oOther )
```

Set a field value from another one.

References GetDateTime(), OFTDate, OFTDateTime, OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

**11.43.2.34 operator=()** [2/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    int nVal )
```

Set an integer value to the field.

**11.43.2.35 operator=()** [3/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    GIntBig nVal )
```

Set an integer value to the field.

**11.43.2.36 operator=()** [4/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    double dfVal )
```

Set a real value to the field.

**11.43.2.37 operator=()** [5/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    const char * pszVal )
```

Set a string value to the field.

**11.43.2.38 operator=()** [6/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    const std::string & osVal )
```

Set a string value to the field.

**11.43.2.39 operator=()** [7/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    const std::vector< int > & oArray )
```

Set an array of integer to the field.

**11.43.2.40 operator=()** [8/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    const std::vector< GIntBig > & oArray )
```

Set an array of big integer to the field.

**11.43.2.41 operator=()** [9/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    const std::vector< double > & oArray )
```

Set an array of double to the field.

**11.43.2.42 operator=()** [10/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    const std::vector< std::string > & oArray )
```

Set an array of strings to the field.

References CPLStringList::AddString(), and CPLStringList::List().

#### 11.43.2.43 operator=() [11/11]

```
OGRFeature::FieldValue & OGRFeature::FieldValue::operator= (
    CSLConstList papszValues )
```

Set an array of strings to the field.

#### 11.43.2.44 SetDateTime()

```
void OGRFeature::FieldValue::SetDateTime (
    int nYear,
    int nMonth,
    int nDay,
    int nHour = 0,
    int nMinute = 0,
    float fSecond = 0.f,
    int nTZFlag = 0 )
```

Set date time value/

#### 11.43.2.45 SetNull()

```
void OGRFeature::FieldValue::SetNull ( )
```

Set a null value to the field.

#### 11.43.2.46 Unset()

```
void OGRFeature::FieldValue::Unset ( ) [inline]
```

Unset the field.

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

- **ogr\_feature.h**
- ogrfeature.cpp

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

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

- cpl\_minizip\_unzip.cpp

### 11.45 FindFileTLS Struct Reference

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

- cpl\_findfile.cpp



## 11.46 GDALScaledProgressInfo Struct Reference

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

- `cpl_progress.cpp`

## 11.47 GOA2Manager Class Reference

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

### Public Types

- enum **AuthMethod**

### Public Member Functions

- **GOA2Manager** ()
- bool **SetAuthFromGCE** ( **CSLConstList** papszOptions)
- bool **SetAuthFromRefreshToken** (const char \*pszRefreshToken, const char \*pszClientId, const char \*pszClientSecret, **CSLConstList** papszOptions)
- bool **SetAuthFromServiceAccount** (const char \*pszPrivateKey, const char \*pszClientEmail, const char \*pszScope, **CSLConstList** papszAdditionalClaims, **CSLConstList** papszOptions)
- **AuthMethod** **GetAuthMethod** () const
- const char \* **GetBearer** () const
- const **CPLString** & **GetPrivateKey** () const
- const **CPLString** & **GetClientEmail** () const

### 11.47.1 Detailed Description

Manager of Google OAuth2 authentication.

This class handles different authentication methods and handles renewal of access token.

Since

GDAL 2.3

### 11.47.2 Member Enumeration Documentation

#### 11.47.2.1 AuthMethod

```
enum GOA2Manager::AuthMethod
```

Authentication method

### 11.47.3 Constructor & Destructor Documentation

#### 11.47.3.1 GOA2Manager()

```
GOA2Manager::GOA2Manager ( )
```

Constructor

### 11.47.4 Member Function Documentation

#### 11.47.4.1 GetAuthMethod()

```
AuthMethod GOA2Manager::GetAuthMethod ( ) const [inline]
```

Returns the authentication method.

#### 11.47.4.2 GetBearer()

```
const char * GOA2Manager::GetBearer ( ) const
```

Return the access token.

This is the value to append to a "Authorization: Bearer " HTTP header.

A network request is issued only if no access token has been yet queried, or if its expiration delay has been reached.

**Returns**

the access token, or NULL in case of error.

References CSLDestroy(), CSLFetchNameValue(), GOA2GetAccessTokenFromCloudEngineVM(), GOA2Get↵  
AccessTokenFromServiceAccount(), and CPLStringList::List().

#### 11.47.4.3 GetClientEmail()

```
const CPLString& GOA2Manager::GetClientEmail ( ) const [inline]
```

Returns client email for SERVICE\_ACCOUNT method

#### 11.47.4.4 GetPrivateKey()

```
const CPLString& GOA2Manager::GetPrivateKey ( ) const [inline]
```

Returns private key for SERVICE\_ACCOUNT method

#### 11.47.4.5 SetAuthFromGCE()

```
bool GOA2Manager::SetAuthFromGCE (
    CSLConstList papszOptions )
```

Specifies that the authentication will be done using the local credentials of the current Google Compute Engine VM

This queries `http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/de`

## Parameters

<i>papszOptions</i>	NULL terminated list of options.
---------------------	----------------------------------

## Returns

true in case of success (no network access is done at this stage)

## 11.47.4.6 SetAuthFromRefreshToken()

```
bool GOA2Manager::SetAuthFromRefreshToken (
    const char * pszRefreshToken,
    const char * pszClientId,
    const char * pszClientSecret,
    CSLConstList papszOptions )
```

Specifies that the authentication will be done using the OAuth2 client id method.

See <http://code.google.com/apis/accounts/docs/OAuth2.html>

## Parameters

<i>pszRefreshToken</i>	refresh token. Must be non NULL.
<i>pszClientId</i>	client id (may be NULL, in which case the GOA2_CLIENT_ID configuration option is used)
<i>pszClientSecret</i>	client secret (may be NULL, in which case the GOA2_CLIENT_SECRET configuration option is used)
<i>papszOptions</i>	NULL terminated list of options, or NULL.

## Returns

true in case of success (no network access is done at this stage)

References CPLError().

## 11.47.4.7 SetAuthFromServiceAccount()

```
bool GOA2Manager::SetAuthFromServiceAccount (
    const char * pszPrivateKey,
    const char * pszClientEmail,
    const char * pszScope,
    CSLConstList papszAdditionalClaims,
    CSLConstList papszOptions )
```

Specifies that the authentication will be done using the OAuth2 service account method.

See <https://developers.google.com/identity/protocols/OAuth2ServiceAccount>

## Parameters

<i>pszPrivateKey</i>	RSA private key. Must be non NULL.
<i>pszClientEmail</i>	client email. Must be non NULL.
<i>pszScope</i>	authorization scope. Must be non NULL.
<i>papszAdditionalClaims</i>	NULL terminate list of additional claims, or NULL.
<i>papszOptions</i>	NULL terminated list of options, or NULL.

## Returns

true in case of success (no network access is done at this stage)

References CPLError(), and EQUAL.

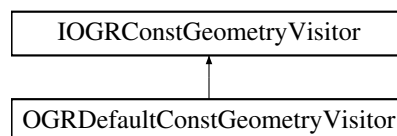
The documentation for this class was generated from the following files:

- **cpl\_http.h**
- **cpl\_google\_oauth2.cpp**

## 11.48 IOGRConstGeometryVisitor Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for IOGRConstGeometryVisitor:



### Public Member Functions

- virtual **~IOGRConstGeometryVisitor** ()=default
- virtual void **visit** (const **OGRPoint** \*)=0
- virtual void **visit** (const **OGRLineString** \*)=0
- virtual void **visit** (const **OGRLinearRing** \*)=0
- virtual void **visit** (const **OGRPolygon** \*)=0
- virtual void **visit** (const **OGRMultiPoint** \*)=0
- virtual void **visit** (const **OGRMultiLineString** \*)=0
- virtual void **visit** (const **OGRMultiPolygon** \*)=0
- virtual void **visit** (const **OGRGeometryCollection** \*)=0
- virtual void **visit** (const **OGRCircularString** \*)=0
- virtual void **visit** (const **OGRCompoundCurve** \*)=0
- virtual void **visit** (const **OGRCurvePolygon** \*)=0
- virtual void **visit** (const **OGRMultiCurve** \*)=0
- virtual void **visit** (const **OGRMultiSurface** \*)=0
- virtual void **visit** (const **OGRTriangle** \*)=0
- virtual void **visit** (const **OGRPolyhedralSurface** \*)=0
- virtual void **visit** (const **OGRTriangulatedSurface** \*)=0

### 11.48.1 Detailed Description

**OGRGeometry** (p. ??) visitor interface.

Since

GDAL 2.3

### 11.48.2 Constructor & Destructor Documentation

#### 11.48.2.1 ~IOGRConstGeometryVisitor()

```
virtual IOGRConstGeometryVisitor::~IOGRConstGeometryVisitor ( ) [virtual], [default]
```

Destructor/

### 11.48.3 Member Function Documentation

#### 11.48.3.1 visit() [1/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRPoint * ) [pure virtual]
```

Visit **OGRPoint** (p. ??).

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

Referenced by **OGRPoint::accept()**, **OGRLineString::accept()**, **OGRLinearRing::accept()**, **OGRCircularString::accept()**, **OGRCompoundCurve::accept()**, **OGRCurvePolygon::accept()**, **OGRPolygon::accept()**, **OGRTriangle::accept()**, **OGRGeometryCollection::accept()**, **OGRMultiSurface::accept()**, **OGRMultiPolygon::accept()**, **OGRPolyhedralSurface::accept()**, **OGRTriangulatedSurface::accept()**, **OGRMultiPoint::accept()**, **OGRMultiCurve::accept()**, and **OGRMultiLineString::accept()**.

#### 11.48.3.2 visit() [2/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRLineString * ) [pure virtual]
```

Visit **OGRLineString** (p. ??).

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

**11.48.3.3 visit()** [3/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRLinearRing * ) [pure virtual]
```

Visit **OGRLinearRing** (p. ??).

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

**11.48.3.4 visit()** [4/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRPolygon * ) [pure virtual]
```

Visit **OGRPolygon** (p. ??).

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

**11.48.3.5 visit()** [5/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRMultiPoint * ) [pure virtual]
```

Visit **OGRMultiPoint** (p. ??).

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

**11.48.3.6 visit()** [6/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRMultiLineString * ) [pure virtual]
```

Visit **OGRMultiLineString** (p. ??).

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

**11.48.3.7 visit()** [7/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRMultiPolygon * ) [pure virtual]
```

Visit **OGRMultiPolygon** (p. ??).

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

**11.48.3.8 visit()** [8/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRGeometryCollection * ) [pure virtual]
```

Visit **OGRGeometryCollection** (p. ??).

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

**11.48.3.9 visit()** [9/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRCircularString * ) [pure virtual]
```

Visit **OGRCircularString** (p. ??).

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

**11.48.3.10 visit()** [10/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRCompoundCurve * ) [pure virtual]
```

Visit **OGRCompoundCurve** (p. ??).

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

**11.48.3.11 visit()** [11/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRCurvePolygon * ) [pure virtual]
```

Visit **OGRCurvePolygon** (p. ??).

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

**11.48.3.12 visit()** [12/16]

```
virtual void IOGRConstGeometryVisitor::visit (
    const OGRMultiCurve * ) [pure virtual]
```

Visit **OGRMultiCurve** (p. ??).

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

**11.48.3.13 visit()** [13/16]

```
virtual void IOGRConstGeometryVisitor::visit (  
    const OGRMultiSurface * ) [pure virtual]
```

Visit **OGRMultiSurface** (p. ??).

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

**11.48.3.14 visit()** [14/16]

```
virtual void IOGRConstGeometryVisitor::visit (  
    const OGRTriangle * ) [pure virtual]
```

Visit **OGRTriangle** (p. ??).

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

**11.48.3.15 visit()** [15/16]

```
virtual void IOGRConstGeometryVisitor::visit (  
    const OGRPolyhedralSurface * ) [pure virtual]
```

Visit **OGRPolyhedralSurface** (p. ??).

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

**11.48.3.16 visit()** [16/16]

```
virtual void IOGRConstGeometryVisitor::visit (  
    const OGRTriangulatedSurface * ) [pure virtual]
```

Visit **OGRTriangulatedSurface** (p. ??).

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

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

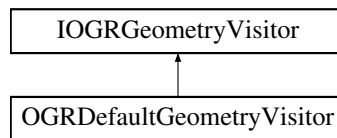
- **ogr\_geometry.h**



## 11.49 IOGRGeometryVisitor Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for IOGRGeometryVisitor:



### Public Member Functions

- virtual `~IOGRGeometryVisitor()`=default
- virtual void `visit ( OGRPoint *)`=0
- virtual void `visit ( OGRLineString *)`=0
- virtual void `visit ( OGRLinearRing *)`=0
- virtual void `visit ( OGRPolygon *)`=0
- virtual void `visit ( OGRMultiPoint *)`=0
- virtual void `visit ( OGRMultiLineString *)`=0
- virtual void `visit ( OGRMultiPolygon *)`=0
- virtual void `visit ( OGRGeometryCollection *)`=0
- virtual void `visit ( OGRCircularString *)`=0
- virtual void `visit ( OGRCompoundCurve *)`=0
- virtual void `visit ( OGRCurvePolygon *)`=0
- virtual void `visit ( OGRMultiCurve *)`=0
- virtual void `visit ( OGRMultiSurface *)`=0
- virtual void `visit ( OGRTriangle *)`=0
- virtual void `visit ( OGRPolyhedralSurface *)`=0
- virtual void `visit ( OGRTriangulatedSurface *)`=0

### 11.49.1 Detailed Description

**OGRGeometry** (p. ??) visitor interface.

Since

GDAL 2.3

### 11.49.2 Constructor & Destructor Documentation

#### 11.49.2.1 `~IOGRGeometryVisitor()`

```
virtual IOGRGeometryVisitor::~~IOGRGeometryVisitor ( ) [virtual], [default]
```

Destructor/

### 11.49.3 Member Function Documentation

#### 11.49.3.1 visit() [1/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRPoint * ) [pure virtual]
```

Visit **OGRPoint** (p. ??).

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

Referenced by **OGRPoint::accept()**, **OGRLineString::accept()**, **OGRLinearRing::accept()**, **OGRCircularString::accept()**, **OGRCompoundCurve::accept()**, **OGRCurvePolygon::accept()**, **OGRPolygon::accept()**, **OGRTriangle::accept()**, **OGRGeometryCollection::accept()**, **OGRMultiSurface::accept()**, **OGRMultiPolygon::accept()**, **OGRPolyhedralSurface::accept()**, **OGRTriangulatedSurface::accept()**, **OGRMultiPoint::accept()**, **OGRMultiCurve::accept()**, and **OGRMultiLineString::accept()**.

#### 11.49.3.2 visit() [2/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRLineString * ) [pure virtual]
```

Visit **OGRLineString** (p. ??).

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

#### 11.49.3.3 visit() [3/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRLinearRing * ) [pure virtual]
```

Visit **OGRLinearRing** (p. ??).

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

#### 11.49.3.4 visit() [4/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRPolygon * ) [pure virtual]
```

Visit **OGRPolygon** (p. ??).

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

**11.49.3.5 visit()** [5/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRMultiPoint * ) [pure virtual]
```

Visit **OGRMultiPoint** (p. ??).

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

**11.49.3.6 visit()** [6/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRMultiLineString * ) [pure virtual]
```

Visit **OGRMultiLineString** (p. ??).

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

**11.49.3.7 visit()** [7/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRMultiPolygon * ) [pure virtual]
```

Visit **OGRMultiPolygon** (p. ??).

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

**11.49.3.8 visit()** [8/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRGeometryCollection * ) [pure virtual]
```

Visit **OGRGeometryCollection** (p. ??).

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

**11.49.3.9 visit()** [9/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRCircularString * ) [pure virtual]
```

Visit **OGRCircularString** (p. ??).

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

**11.49.3.10 visit()** [10/16]

```
virtual void IOGRGeometryVisitor::visit (  
    OGRCompoundCurve * ) [pure virtual]
```

Visit **OGRCompoundCurve** (p. ??).

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

**11.49.3.11 visit()** [11/16]

```
virtual void IOGRGeometryVisitor::visit (  
    OGRCurvePolygon * ) [pure virtual]
```

Visit **OGRCurvePolygon** (p. ??).

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

**11.49.3.12 visit()** [12/16]

```
virtual void IOGRGeometryVisitor::visit (  
    OGRMultiCurve * ) [pure virtual]
```

Visit **OGRMultiCurve** (p. ??).

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

**11.49.3.13 visit()** [13/16]

```
virtual void IOGRGeometryVisitor::visit (  
    OGRMultiSurface * ) [pure virtual]
```

Visit **OGRMultiSurface** (p. ??).

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

**11.49.3.14 visit()** [14/16]

```
virtual void IOGRGeometryVisitor::visit (  
    OGRTriangle * ) [pure virtual]
```

Visit **OGRTriangle** (p. ??).

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

**11.49.3.15 visit()** [15/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRPolyhedralSurface * ) [pure virtual]
```

Visit **OGRPolyhedralSurface** (p. ??).

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

**11.49.3.16 visit()** [16/16]

```
virtual void IOGRGeometryVisitor::visit (
    OGRTriangulatedSurface * ) [pure virtual]
```

Visit **OGRTriangulatedSurface** (p. ??).

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

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

- **ogr\_geometry.h**

**11.50 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

**11.50.1 Detailed Description**

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

Since

GDAL 2.0

**11.50.2 Member Function Documentation****11.50.2.1 CommitTransaction()**

```
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

<i>poDSInOut</i>	datasource handle that may be modified
<i>bOutHasReopenedDS</i>	output boolean to indicate if datasource has been closed

## Returns

OGRERR\_NONE in case of success

## 11.50.2.2 RollbackTransaction()

```
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

<i>poDSInOut</i>	datasource handle that may be modified
<i>bOutHasReopenedDS</i>	output boolean to indicate if datasource has been closed

## Returns

OGRERR\_NONE in case of success

## 11.50.2.3 StartTransaction()

```
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

<i>poDSInOut</i>	datasource handle that may be modified
<i>bOutHasReopenedDS</i>	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:

- ocremulatedtransaction.h
- ocremulatedtransaction.cpp

## 11.51 LinearUnitsStruct Struct Reference

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

- ogr\_srs\_proj4.cpp

## 11.52 linkedlist\_data\_s Struct Reference

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

- cpl\_minizip\_zip.cpp

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

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

- cpl\_minizip\_zip.cpp

## 11.54 OGR\_SRSNode Class Reference

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

## Public Member Functions

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

### 11.54.1 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 its data structure; however, this class is user accessible for detailed access to components of an SRS definition.

### 11.54.2 Constructor & Destructor Documentation

#### 11.54.2.1 OGR\_SRSNode()

```
OGR_SRSNode::OGR_SRSNode (
    const char * pszValueIn = nullptr ) [explicit]
```

Constructor.



## Parameters

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

Referenced by Clone().

### 11.54.3 Member Function Documentation

#### 11.54.3.1 AddChild()

```
void OGR_SRSNode::AddChild (
    OGR_SRSNode * poNew )
```

Add passed node as a child of target node.

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

## Parameters

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

References InsertChild().

Referenced by Clone(), OGRSpatialReference::CloneGeogCS(), OGRSpatialReference::importFromWkt(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and OGRSpatialReference::SetVertCS().

#### 11.54.3.2 applyRemapper()

```
OGRERR OGR_SRSNode::applyRemapper (
    const char * pszNode,
    const char *const * papszSrcValues,
    const char *const * papszDstValues,
    int nStepSize = 1,
    int bChildOfHit = FALSE )
```

Remap node values matching list.

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

## Parameters

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

## Returns

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

< Success

References applyRemapper(), EQUAL, GetChild(), GetChildCount(), OGRERR\_NONE, and SetValue().

Referenced by applyRemapper(), OGRSpatialReference::morphFromESRI(), and OGRSpatialReference::morph↵ToESRI().

## 11.54.3.3 ClearChildren()

```
void OGR_SRSNode::ClearChildren ( )
```

Clear children nodes

References CPLFree.

Referenced by OGRSpatialReference::SetVertCS().

## 11.54.3.4 Clone()

```
OGR_SRSNode * OGR_SRSNode::Clone ( ) const
```

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

## Returns

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

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

Referenced by OGRSpatialReference::Clone(), OGRSpatialReference::CloneGeogCS(), OGRSpatialReference↵::CopyGeogCSFrom(), OGRSpatialReference::OGRSpatialReference(), OGRSpatialReference::operator=(), OG↵RSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), and OGRSpatialReference::Strip↵Vertical().

#### 11.54.3.5 DestroyChild()

```
void OGR_SRSNode::DestroyChild (
    int iChild )
```

Remove a child node, and it's subtree.

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

## Parameters

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

Referenced by `OGRSpatialReference::CopyGeogCSFrom()`, `OGRSpatialReference::importFromProj4()`, `OGRSpatialReference::morphFromESRI()`, `OGRSpatialReference::morphToESRI()`, `OGRSpatialReference::SetAuthority()`, `OGRSpatialReference::SetAxes()`, `OGRSpatialReference::SetGeogCS()`, `OGRSpatialReference::SetTargetLinearUnits()`, `OGRSpatialReference::SetTOWGS84()`, and `StripNodes()`.

11.54.3.6 `exportToPrettyWkt()`

```
OGRERR OGR_SRSNode::exportToPrettyWkt (
    char ** ppszResult,
    int nDepth = 1 ) const
```

Convert this tree of nodes into pretty WKT format.

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

## Parameters

<i>ppszResult</i>	the resulting string is returned in this pointer.
<i>nDepth</i>	depth of the node

## Returns

currently `OGRERR_NONE` is always returned, but the future it is possible error conditions will develop.

< Success

References `CPLCalloc()`, `CPLMalloc()`, `CSLDestroy()`, `exportToPrettyWkt()`, `GetChildCount()`, and `OGRERR_NONE`.

Referenced by `exportToPrettyWkt()`, and `OGRSpatialReference::exportToPrettyWkt()`.

11.54.3.7 `exportToWkt()`

```
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 **CPLFree()** (p. ??) when no longer needed. It is the responsibility of the caller.

## Parameters

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

## Returns

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

< Success

References CPLCalloc(), CPLMalloc(), CSLDestroy(), exportToWkt(), and OGRERR\_NONE.

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

## 11.54.3.8 FindChild()

```
int OGR_SRSNode::FindChild (
    const char * pszValueIn ) const
```

Find the index of the child matching the given string.

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

## Parameters

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

## Returns

the child index, or -1 on failure.

References EQUAL.

Referenced by OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::Fixup(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::ImportFromProj4(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and StripNodes().

## 11.54.3.9 FixupOrdering()

```
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.

< Success

< Success

< Success

References CPLCalloc(), CPLDebug(), CPLFree, CSLFindString(), EQUAL, FixupOrdering(), GetChild(), GetChildCount(), GetValue(), and OGRERR\_NONE.

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

**11.54.3.10 GetChild()** [1/2]

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

Fetch requested child.

**Parameters**

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

**Returns**

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

Referenced by applyRemapper(), OGRSpatialReference::EPSGTreatsAsLatLong(), OGRSpatialReference::EPSGTreatsAsNorthingEasting(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::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::IsSame(), MakeValueSafe(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTargetLinearUnits(), StripNodes(), and OGRSpatialReference::StripVertical().

**11.54.3.11 GetChild()** [2/2]

```
const OGR_SRSNode * OGR_SRSNode::GetChild (
    int iChild ) const
```

Fetch requested child.

## Parameters

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

## Returns

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

## 11.54.3.12 GetChildCount()

```
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()`, `exportToPrettyWkt()`, `OGRSpatialReference::FindProjParm()`, `FixupOrdering()`, `OGRSpatialReference::GetAngularUnits()`, `OGRSpatialReference::GetAttributeValue()`, `OGRSpatialReference::GetAuthorityCode()`, `OGRSpatialReference::GetAuthorityName()`, `OGRSpatialReference::GetAxis()`, `OGRSpatialReference::GetExtension()`, `OGRSpatialReference::GetInvFlattening()`, `OGRSpatialReference::GetPrimeMeridian()`, `OGRSpatialReference::GetSemiMajor()`, `OGRSpatialReference::GetTargetLinearUnits()`, `OGRSpatialReference::GetTOWGS84()`, `OGRSpatialReference::IsSame()`, `MakeValueSafe()`, `OGRSpatialReference::morphToESRI()`, `OGRSpatialReference::SetAngularUnits()`, `OGRSpatialReference::SetExtension()`, `OGRSpatialReference::SetLinearUnitsAndUpdateParameters()`, `OGRSpatialReference::SetNode()`, `OGRSpatialReference::SetProjParm()`, `OGRSpatialReference::SetTargetLinearUnits()`, `OGRSpatialReference::SetTOWGS84()`, and `StripNodes()`.

## 11.54.3.13 GetNode() [1/2]

```
OGR_SRSNode * OGR_SRSNode::GetNode (
    const char * pszName )
```

Find named node in tree.

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

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

## Parameters

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

**Returns**

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

References EQUAL, and GetNode().

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

**11.54.3.14 GetNode()** [2/2]

```
const OGR_SRSNode * OGR_SRSNode::GetNode (
    const char * pszName ) const
```

Find named node in tree.

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

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

**Parameters**

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

**Returns**

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

References GetNode().

**11.54.3.15 GetValue()**

```
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::EPSGTreatsAs↵  
 NorthingEasting(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::FindProjParm(), Fixup↵  
 Ordering(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAttrValue(), OGRSpatial↵  
 Reference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::GetAxis(),



OGRSpatialReference::GetExtension(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTargetLinearUnits(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::IsCompound(), OGRSpatialReference::IsGeocentric(), OGRSpatialReference::IsGeographic(), OGRSpatialReference::IsProjected(), OGRSpatialReference::IsSame(), OGRSpatialReference::IsVertical(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetVertCS(), and OGRSpatialReference::StripCTParms().

#### 11.54.3.16 importFromWkt() [1/2]

```
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 its children is consumed from the input string, and the input string pointer is then updated to point to the remaining (unused) input.

##### Parameters

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

##### Returns

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

**Deprecated** GDAL 2.3. Use **importFromWkt(const char\*\*)** (p. ??) instead.

#### 11.54.3.17 importFromWkt() [2/2]

```
OGRERR OGR_SRSNode::importFromWkt (
    const 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 its children is consumed from the input string, and the input string pointer is then updated to point to the remaining (unused) input.

##### Parameters

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

**Returns**

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

**Since**

GDAL 2.3

**11.54.3.18 InsertChild()**

```
void OGR_SRSNode::InsertChild (
    OGR_SRSNode * poNew,
    int iChild )
```

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

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

**Parameters**

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

References CPLRealloc().

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

**11.54.3.19 IsLeafNode()**

```
int OGR_SRSNode::IsLeafNode ( ) const [inline]
```

Return whether this is a leaf node.

**Returns**

TRUE or FALSE

## 11.54.3.20 MakeValueSafe()

```
void OGR_SRSNode::MakeValueSafe ( )
```

Message 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().

## 11.54.3.21 SetValue()

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

Set the node value.

## Parameters

<i>pszNewValue</i>	the new value to assign to this node. The passed string is duplicated and remains the responsibility of the caller.
--------------------	---

References CPLFree, and CPLStrdup().

Referenced by applyRemapper(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), and OGRSpatialReference::SetTargetLinearUnits().

## 11.54.3.22 StripNodes()

```
void OGR_SRSNode::StripNodes (
    const char * pszName )
```

Strip child nodes matching name.

Removes any descendant nodes of this node that match the given name. Of course children of removed nodes are also discarded.

## Parameters

<i>pszName</i>	the name for nodes that should be removed.
----------------	--

References DestroyChild(), FindChild(), GetChild(), GetChildCount(), and StripNodes().

Referenced by OGRSpatialReference::exportToPrettyWkt(), OGRSpatialReference::importFromEPSG(), OGRSpatialReference::StripCTParms(), and StripNodes().

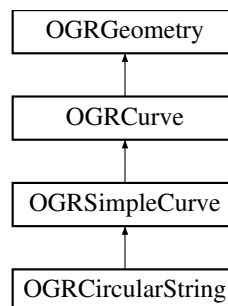
The documentation for this class was generated from the following files:

- **ogr\_spatialref.h**
- **ogr\_srsnode.cpp**

## 11.55 OGRCircularString Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRCircularString:



### Public Member Functions

- **OGRCircularString ()**  
*Create an empty circular string.*
- **OGRCircularString (const OGRCircularString &other)**  
*Copy constructor.*
- **OGRCircularString & operator= (const OGRCircularString &other)**  
*Assignment operator.*
- virtual **OGRERR importFromWkb** (const unsigned char \*, int, **OGRWkbVariant**, int &nBytesConsumedOut) override  
*Assign geometry from well known binary data.*
- virtual **OGRERR exportToWkb** ( **OGRWkbByteOrder**, unsigned char \*, **OGRWkbVariant= wkbVariantOldOgc**) const override  
*Convert a geometry into well known binary format.*
- **OGRERR importFromWkt** (const char \*\*) override
- virtual **OGRERR exportToWkt** (char \*\*ppszDstText, **OGRWkbVariant= wkbVariantOldOgc**) const override  
*Convert a geometry into well known text format.*
- virtual **OGRBoolean IsValid** () const override  
*Test if the geometry is valid.*
- virtual void **getEnvelope** (OGREnvelope \*psEnvelope) const override  
*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (OGREnvelope3D \*psEnvelope) const override  
*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- virtual double **get\_Length** () const override  
*Returns the length of the curve.*
- virtual **OGRLineString \* CurveToLine** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*ppszOptions=NULLPTR) const override  
*Return a linestring from a curve geometry.*

- virtual void **Value** (double, **OGRPoint** \*) const override  
*Fetch point at given distance along curve.*
- virtual double **get\_Area** () const override  
*Get the area of the (closed) curve.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Fetch geometry type.*
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual void **segmentize** (double dfMaxLength) override  
*Modify the geometry such it has no segment longer then the given distance.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override  
*Returns if this geometry is or has curve geometry.*
- virtual **OGRGeometry** \* **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*papszOptions=nullptr) const override  
*Return, possibly approximate, non-curve version of this geometry.*
- **OGRSimpleCurve** \* **toUpperClass** ()
- const **OGRSimpleCurve** \* **toUpperClass** () const
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override
- virtual **OGRErr** **importFromWkt** (const char \*\*papszInput)=0  
*Assign geometry from well known text data.*
- **OGRErr** **importFromWkt** (char \*\*papszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

## Additional Inherited Members

### 11.55.1 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

### 11.55.2 Constructor & Destructor Documentation

#### 11.55.2.1 OGRCircularString()

```
OGRCircularString::OGRCircularString (
    const OGRCircularString & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

### 11.55.3 Member Function Documentation

#### 11.55.3.1 `accept()` [1/2]

```
virtual void OGRCircularString::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRGeometryVisitor::visit().

#### 11.55.3.2 `accept()` [2/2]

```
virtual void OGRCircularString::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRConstGeometryVisitor::visit().

#### 11.55.3.3 `CurveToLine()`

```
OGRLineString * OGRCircularString::CurveToLine (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings or NULL. See <b>OGRGeometryFactory::curveToLineString()</b> (p. ??) for valid options.

**Returns**

a line string approximating the curve

**Since**

GDAL 2.0

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

References OGRSimpleCurve::addSubLineString(), OGRGeometry::assignSpatialReference(), OGRGeometryFactory::curveToLineString(), OGRGeometry::getCoordinateDimension(), and OGRGeometry::getSpatialReference().

Referenced by get\_Area(), and getLinearGeometry().

**11.55.3.4 exportToWkb()**

```
OGRERR OGRCircularString::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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**

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
<i>eWkbVariant</i>	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.

< Failure

< Success

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

References OGRSimpleCurve::exportToWkb(), OGRERR\_FAILURE, wkbVariantIso, and wkbVariantOldOgc.

### 11.55.3.5 exportToWkt()

```
OGRErr OGRCircularString::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

#### Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

#### Returns

Currently OGRERR\_NONE is always returned.

< Failure

< Success

< Not enough memory

< Not enough memory

< Success

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

References OGRSimpleCurve::exportToWkt(), OGRERR\_FAILURE, and wkbVariantIso.

### 11.55.3.6 get\_Area()

```
double OGRCircularString::get_Area ( ) const [override], [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\(\)](#), [OGRCurve::get\\_AreaOfCurveSegments\(\)](#), [OGRCurve::get\\_IsClosed\(\)](#), [OGRSimpleCurve::get\\_LinearArea\(\)](#), [OGRCurve::IsConvex\(\)](#), [OGRSimpleCurve::IsEmpty\(\)](#), and [M\\_PI](#).

### 11.55.3.7 get\_Length()

```
double OGRCircularString::get_Length ( ) const [override], [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. ??).

References [OGRGeometryFactory::GetCurveParameters\(\)](#).

### 11.55.3.8 getEnvelope() [1/2]

```
void OGRCircularString::getEnvelope (
    OGREnvelope * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

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

References [OGRSimpleCurve::getEnvelope\(\)](#).

### 11.55.3.9 `getEnvelope()` [2/2]

```
void OGRCircularString::getEnvelope (
    OGREnvelope3D * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope (3D) for this geometry in the passed `psEnvelope` structure.

This method is the same as the C function `OGR_G_GetEnvelope3D()` (p. ??).

#### Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

#### Since

OGR 1.9.0

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

References `OGRSimpleCurve::getEnvelope()`.

### 11.55.3.10 `getGeometryName()`

```
const char * OGRCircularString::getGeometryName ( ) const [override], [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. ??).

### 11.55.3.11 `getGeometryType()`

```
OGRwkbGeometryType OGRCircularString::getGeometryType ( ) const [override], [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 `wkbCircularString`, `wkbCircularStringM`, `wkbCircularStringZ`, and `wkbCircularStringZM`.

## 11.55.3.12 getLinearGeometry()

```
OGRGeometry * OGRCircularString::getLinearGeometry (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings. See <b>OGRGeometryFactory::curveToLineString()</b> (p. ??) for valid options.

## Returns

a new geometry.

## Since

GDAL 2.0

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

References CurveToLine().

## 11.55.3.13 hasCurveGeometry()

```
OGRBoolean OGRCircularString::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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

<i>bLookForNonLinear</i>	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. ??).

## 11.55.3.14 importFromWkb()

```
OGRERR OGRCircularString::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	if wkbVariantPostGIS1, special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

## Since

GDAL 2.3

< Success  
 < Corrupt data

< Success

< Corrupt data

< Not enough data to deserialize

< Failure  
 < Success

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

References OGRSimpleCurve::empty(), OGRSimpleCurve::importFromWkb(), OGRERR\_CORRUPT\_DATA, and OGRERR\_NONE.

#### 11.55.3.15 importFromWkt() [1/3]

```
OGRErr OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

#### 11.55.3.16 importFromWkt() [2/3]

```
OGRErr OGRGeometry::importFromWkt
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

#### Parameters

<i>ppsInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
-----------------	---

**Returns**

OGRErr\_NONE if all goes well, otherwise any of OGRErr\_NOT\_ENOUGH\_DATA, OGRErr\_UNSUPP↵ORTED\_GEOMETRY\_TYPE, or OGRErr\_CORRUPT\_DATA may be returned.

**11.55.3.17 importFromWkt()** [3/3]

```
OGRErr OGRCircularString::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated < Success

< Corrupt data

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

References OGRSimpleCurve::empty(), OGRSimpleCurve::importFromWkt(), OGRErr\_CORRUPT\_DATA, and OGRErr\_NONE.

**11.55.3.18 IsValid()**

```
OGRBoolean OGRCircularString::IsValid ( ) const [override], [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().

**11.55.3.19 operator=()**

```
OGRCircularString & OGRCircularString::operator= (
    const OGRCircularString & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

Since

GDAL 2.1

References OGRSimpleCurve::operator=().

## 11.55.3.20 segmentize()

```
void OGRCircularString::segmentize (
    double dfMaxLength ) [override], [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

<i>dfMaxLength</i>	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 consecutive vertices is lower than dfMaxLength.

## Parameters

<i>dfMaxLength</i>	maximum length between 2 consecutive vertices.
--------------------	--

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

References OGRGeometryFactory::GetCurveParameters(), and OGRSimpleCurve::reversePoints().

## 11.55.3.21 toUpperClass() [1/2]

```
OGRSimpleCurve* OGRCircularString::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

## 11.55.3.22 toUpperClass() [2/2]

```
const OGRSimpleCurve* OGRCircularString::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

## 11.55.3.23 Value()

```
void OGRCircularString::Value (
    double dfDistance,
    OGRPoint * poPoint ) const [override], [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

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

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

References **OGRSimpleCurve::EndPoint()**, **OGRGeometry::getCoordinateDimension()**, **OGRGeometryFactory::GetCurveParameters()**, **OGRPoint::setX()**, **OGRPoint::setY()**, **OGRPoint::setZ()**, and **OGRSimpleCurve::StartPoint()**.

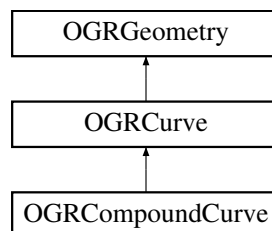
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrcircularstring.cpp**

## 11.56 OGRCompoundCurve Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for **OGRCompoundCurve**:



### Public Types

- typedef **OGRCurve** **ChildType**

### Public Member Functions

- **OGRCompoundCurve** ()  
*Create an empty compound curve.*
- **OGRCompoundCurve** (const **OGRCompoundCurve** &other)  
*Copy constructor.*
- **OGRCompoundCurve** & **operator=** (const **OGRCompoundCurve** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual int **WkbSize** () const override  
*Returns size of related binary representation.*



- virtual **OGRERR** **importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumedOut) override  
*Assign geometry from well known binary data.*
- virtual **OGRERR** **exportToWkb** ( **OGRwkbByteOrder**, unsigned char \*, **OGRwkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known binary format.*
- **OGRERR** **importFromWkt** (const char \*\*) override
- virtual **OGRERR** **exportToWkt** (char \*\*ppszDstText, **OGRwkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known text format.*
- virtual **OGRGeometry** \* **clone** () const override  
*Make a copy of this object.*
- virtual void **empty** () override  
*Clear geometry information. This restores the geometry to its initial state after construction, and before assignment of actual geometry.*
- virtual void **getEnvelope** (OGREnvelope \*psEnvelope) const override  
*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (OGREnvelope3D \*psEnvelope) const override  
*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- virtual **OGRBoolean** **IsEmpty** () const override  
*Returns TRUE (non-zero) if the object has no points.*
- virtual double **get\_Length** () const override  
*Returns the length of the curve.*
- virtual void **StartPoint** ( **OGRPoint** \*) const override  
*Return the curve start point.*
- virtual void **EndPoint** ( **OGRPoint** \*) const override  
*Return the curve end point.*
- virtual void **Value** (double, **OGRPoint** \*) const override  
*Fetch point at given distance along curve.*
- virtual **OGRLineString** \* **CurveToLine** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*ppszOptions=NULLPTR) const override  
*Return a linestring from a curve geometry.*
- virtual int **getNumPoints** () const override  
*Return the number of points of a curve geometry.*
- virtual double **get\_AreaOfCurveSegments** () const override
- virtual double **get\_Area** () const override  
*Get the area of the (closed) curve.*
- virtual **OGRBoolean** **Equals** (const **OGRGeometry** \*) const override  
*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) override  
*Set the coordinate dimension.*
- virtual void **set3D** ( **OGRBoolean** bls3D) override  
*Add or remove the Z coordinate dimension.*
- virtual void **setMeasured** ( **OGRBoolean** blsMeasured) override  
*Add or remove the M coordinate dimension.*
- virtual void **assignSpatialReference** ( **OGRSpatialReference** \*poSR) override

- Assign spatial reference to this object.*
- **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.*
- **OGRCurve** \* **stealCurve** (int)  
*"Steal" reference to curve.*
- virtual **OGRPointIterator** \* **getPointIterator** () const override  
*Returns a point iterator over the curve.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Fetch geometry type.*
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRErr transform** ( **OGRCoordinateTransformation** \*poCT) override  
*Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** () override  
*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual void **segmentize** (double dfMaxLength) override  
*Modify the geometry such it has no segment longer then the given distance.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override  
*Returns if this geometry is or has curve geometry.*
- virtual **OGRGeometry** \* **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*papszOptions=nullptr) const override  
*Return, possibly approximate, non-curve version of this geometry.*
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override
- virtual void **swapXY** () override  
*Swap x and y coordinates.*
- virtual **OGRErr importFromWkt** (const char \*\*ppszInput)=0  
*Assign geometry from well known text data.*
- **OGRErr importFromWkt** (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

## Additional Inherited Members

### 11.56.1 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 Concrete representation of a compound curve, made of curves: **OGRLineString** (p. ??) and **OGR**↔  
**RCircularString** (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

## 11.56.2 Member Typedef Documentation

### 11.56.2.1 ChildType

```
typedef OGRCurve OGRCompoundCurve::ChildType
```

Type of child elements.

## 11.56.3 Constructor & Destructor Documentation

### 11.56.3.1 OGRCompoundCurve()

```
OGRCompoundCurve::OGRCompoundCurve (
    const OGRCompoundCurve & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

## 11.56.4 Member Function Documentation

### 11.56.4.1 accept() [1/2]

```
virtual void OGRCompoundCurve::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRGeometryVisitor::visit().

#### 11.56.4.2 accept() [2/2]

```
virtual void OGRCompoundCurve::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRConstGeometryVisitor::visit().

#### 11.56.4.3 addCurve()

```
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

<i>poCurve</i>	geometry to add to the container.
<i>dfToleranceEps</i>	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)

< Success

References addCurveDirectly(), OGRGeometry::clone(), OGRERR\_NONE, and OGRGeometry::toCurve().

Referenced by clone().

#### 11.56.4.4 addCurveDirectly()

```
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

<i>poCurve</i>	geometry to add to the container.
<i>dfToleranceEps</i>	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()`.

11.56.4.5 `assignSpatialReference()`

```
void OGRCompoundCurve::assignSpatialReference (
    OGRSpatialReference * poSR ) [override], [virtual]
```

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.

Starting with GDAL 2.3, this will also assign the spatial reference to potential sub-geometries of the geometry (**OGRGeometryCollection** (p. ??), **OGRCurvePolygon**/**OGRPolygon**, **OGRCompoundCurve** (p. ??), **OGR↵PolyhedralSurface** (p. ??) and their derived classes).

This is similar to the SFCOM `IGeometry::put_SpatialReference()` method.

This method is the same as the C function **OGR\_G\_AssignSpatialReference()** (p. ??).

## Parameters

<i>poSR</i>	new spatial reference system to apply.
-------------	--

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

Referenced by `OGRCurve::CastToCompoundCurve()`, and `clone()`.

11.56.4.6 `begin()` [1/2]

```
ChildType** OGRCompoundCurve::begin ( ) [inline]
```

Return begin of curve iterator.

Since

GDAL 2.3

#### 11.56.4.7 begin() [2/2]

```
const ChildType* const* OGRCompoundCurve::begin ( ) const [inline]
```

Return begin of curve iterator.

Since

GDAL 2.3

#### 11.56.4.8 clone()

```
OGRGeometry * OGRCompoundCurve::clone ( ) const [override], [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(), assignSpatialReference(), OGRGeometry::getSpatialReference(), and OGRCompoundCurve().

#### 11.56.4.9 CurveToLine()

```
OGRLineString * OGRCompoundCurve::CurveToLine (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings or NULL. See <b>OGRGeometryFactory::curveToLineString()</b> (p. ??) for valid options.

## Returns

a line string approximating the curve

## Since

GDAL 2.0

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

Referenced by `get_Area()`, and `getLinearGeometry()`.

11.56.4.10 `empty()`

```
void OGRCompoundCurve::empty ( ) [override], [virtual]
```

Clear geometry information. This restores the geometry to its 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. ??).

11.56.4.11 `end()` [1/2]

```
ChildType** OGRCompoundCurve::end ( ) [inline]
```

Return end of curve iterator.

11.56.4.12 `end()` [2/2]

```
const ChildType* const* OGRCompoundCurve::end ( ) const [inline]
```

Return end of curve iterator.

11.56.4.13 `EndPoint()`

```
void OGRCompoundCurve::EndPoint (
    OGRPoint * poPoint ) const [override], [virtual]
```

Return the curve end point.

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



## Parameters

<i>poPoint</i>	the point to be assigned the end location.
----------------	--

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

References CPLAssert.

Referenced by Value().

## 11.56.4.14 Equals()

```
OGRBoolean OGRCompoundCurve::Equals (
    const OGRGeometry * ) const [override], [virtual]
```

Returns TRUE if two geometries are equivalent.

This operation implements the SQL/MM ST\_OrderingEquals() operation.

The comparison is done in a structural way, that is to say that the geometry types must be identical, as well as the number and ordering of sub-geometries and vertices. Or equivalently, two geometries are considered equal by this method if their WKT/WKB representation is equal. Note: this must be distinguished for equality in a spatial way (which is the purpose of the ST\_Equals() operation).

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(), and OGRGeometry::toCompoundCurve().

## 11.56.4.15 exportToWkb()

```
OGRERR OGRCompoundCurve::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
<i>eWkbVariant</i>	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.

## 11.56.4.16 exportToWkt()

```
OGRErr OGRCompoundCurve::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

## Returns

Currently OGRERR\_NONE is always returned.

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

11.56.4.17 `flattenTo2D()`

```
void OGRCompoundCurve::flattenTo2D ( ) [override], [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. ??).

11.56.4.18 `get_Area()`

```
double OGRCompoundCurve::get_Area ( ) const [override], [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()`, `OGRCurve::IsConvex()`, `IsEmpty()`, `OGRSimpleCurve::setNumPoints()`, and `OGRSimpleCurve::setPoint()`.

11.56.4.19 `get_AreaOfCurveSegments()`

```
double OGRCompoundCurve::get_AreaOfCurveSegments ( ) const [override], [virtual]
```

Return area of curve segments

**Returns**

area.

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

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

Referenced by `get_Area()`.

11.56.4.20 `get_Length()`

```
double OGRCompoundCurve::get_Length ( ) const [override], [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. ??).

11.56.4.21 `getCurve()` [1/2]

```
OGRCurve * OGRCompoundCurve::getCurve (
    int iRing )
```

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**

<i>iRing</i>	curve index from 0 to <b>getNumCurves()</b> (p. ??) - 1.
--------------	--

**Returns**

pointer to curve. May be NULL.

Referenced by `OGRGeometry::dumpReadable()`, `get_AreaOfCurveSegments()`, and `OGRCompoundCurvePointIterator::getNextPoint()`.

11.56.4.22 `getCurve()` [2/2]

```
const OGRCurve * OGRCompoundCurve::getCurve (
    int iCurve ) 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

<i>iCurve</i>	curve index from 0 to <b>getNumCurves()</b> (p. ??) - 1.
---------------	--

## Returns

pointer to curve. May be NULL.

11.56.4.23 **getEnvelope()** [1/2]

```
void OGRCompoundCurve::getEnvelope (
    OGREnvelope * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope for this geometry in the passed *psEnvelope* structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

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

11.56.4.24 **getEnvelope()** [2/2]

```
void OGRCompoundCurve::getEnvelope (
    OGREnvelope3D * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope (3D) for this geometry in the passed *psEnvelope* structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

## Since

OGR 1.9.0

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

#### 11.56.4.25 getGeometryName()

```
const char * OGRCompoundCurve::getGeometryName ( ) const [override], [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. ??).

#### 11.56.4.26 getGeometryType()

```
OGRwkbGeometryType OGRCompoundCurve::getGeometryType ( ) const [override], [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 **wkbCompoundCurve**, **wkbCompoundCurveM**, **wkbCompoundCurveZ**, and **wkbCompoundCurveZM**.

Referenced by **Equals()**.

#### 11.56.4.27 getLinearGeometry()

```
OGRGeometry * OGRCompoundCurve::getLinearGeometry (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings. See <b>OGRGeometryFactory::curveToLineString()</b> (p. ??) for valid options.

## Returns

a new geometry.

## Since

GDAL 2.0

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

References `CurveToLine()`.

11.56.4.28 `getNumCurves()`

```
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_AreaOfCurveSegments()`, and `OGRCompoundCurvePointIterator::getNextPoint()`.

11.56.4.29 `getNumPoints()`

```
int OGRCompoundCurve::getNumPoints ( ) const [override], [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. ??).

Referenced by `get_Area()`.

11.56.4.30 `getPointIterator()`

```
OGRPointIterator * OGRCompoundCurve::getPointIterator ( ) const  [override], [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()`.

11.56.4.31 `hasCurveGeometry()`

```
OGRBoolean OGRCompoundCurve::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const  [override], [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**

<i>bLookForNonLinear</i>	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. ??).

#### 11.56.4.32 importFromWkb()

```
OGRERR OGRCompoundCurve::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

##### Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of <i>pabyData</i> in bytes, or -1 if not known.
<i>eWkbVariant</i>	if <i>wkbVariantPostGIS1</i> , special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

##### 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.

Since

GDAL 2.3

< Success

< Success

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

References OGRERR\_NONE, and wkbNDR.

11.56.4.33 `importFromWkt()` [1/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

11.56.4.34 `importFromWkt()` [2/3]

```
OGRERR OGRGeometry::importFromWkt
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppsInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
-----------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

11.56.4.35 `importFromWkt()` [3/3]

```
OGRERR OGRCompoundCurve::importFromWkt (
    const char ** ppsInput ) [override], [virtual]
```

deprecated

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

## 11.56.4.36 isEmpty()

```
OGRBoolean OGRCompoundCurve::IsEmpty ( ) const [override], [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().

## 11.56.4.37 operator=()

```
OGRCompoundCurve & OGRCompoundCurve::operator= (
    const OGRCompoundCurve & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

## Since

GDAL 2.1

References OGRGeometry::operator=().

## 11.56.4.38 segmentize()

```
void OGRCompoundCurve::segmentize (
    double dfMaxLength ) [override], [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

<i>dfMaxLength</i>	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 consecutive vertices is lower than *dfMaxLength*.

## Parameters

<i>dfMaxLength</i>	maximum length between 2 consecutive vertices.
--------------------	--

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

11.56.4.39 **set3D()**

```
void OGRCompoundCurve::set3D (
    OGRBoolean bIs3D ) [override], [virtual]
```

Add or remove the Z coordinate dimension.

This method adds or removes the explicit Z coordinate dimension. Removing the Z coordinate dimension of a geometry will remove any existing Z values. Adding the Z dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

## Parameters

<i>bIs3D</i>	Should the geometry have a Z dimension, either TRUE or FALSE.
--------------	---

## Since

GDAL 2.1

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

11.56.4.40 **setCoordinateDimension()**

```
void OGRCompoundCurve::setCoordinateDimension (
    int nNewDimension ) [override], [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, a compound curve, a polygon, etc. will affect the children geometries. This will also remove the M dimension if present before this call.

**Deprecated** use **set3D()** (p. ??) or **setMeasured()** (p. ??).

## Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

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

## 11.56.4.41 setMeasured()

```
void OGRCompoundCurve::setMeasured (
    OGRBoolean bIsMeasured ) [override], [virtual]
```

Add or remove the M coordinate dimension.

This method adds or removes the explicit M coordinate dimension. Removing the M coordinate dimension of a geometry will remove any existing M values. Adding the M dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

## Parameters

<i>bIsMeasured</i>	Should the geometry have a M dimension, either TRUE or FALSE.
--------------------	---

## Since

GDAL 2.1

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

## 11.56.4.42 StartPoint()

```
void OGRCompoundCurve::StartPoint (
    OGRPoint * poPoint ) const [override], [virtual]
```

Return the curve start point.

This method relates to the SF COM ICurve::get\_StartPoint() method.

## Parameters

<i>poPoint</i>	the point to be assigned the start location.
----------------	--

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

References CPLAssert.

Referenced by Value().

## 11.56.4.43 stealCurve()

```
OGRCurve * OGRCompoundCurve::stealCurve (
    int iCurve )
```

"Steal" reference to curve.

## Parameters

<i>iCurve</i>	curve index from 0 to <b>getNumCurves()</b> (p. ??) - 1.
---------------	--

## Returns

pointer to curve. May be NULL.

Referenced by OGRGeometryFactory::curveFromLineString().

## 11.56.4.44 swapXY()

```
void OGRCompoundCurve::swapXY ( ) [override], [virtual]
```

Swap x and y coordinates.

## Since

OGR 1.8.0

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

## 11.56.4.45 transform()

```
OGRERR OGRCompoundCurve::transform (
    OGRCoordinateTransformation * poCT ) [override], [virtual]
```

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGR↔SpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

## Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

## Returns

OGRERR\_NONE on success or an error code.

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

## 11.56.4.46 Value()

```
void OGRCompoundCurve::Value (
    double dfDistance,
    OGRPoint * poPoint ) const [override], [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

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

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

References EndPoint(), and StartPoint().

## 11.56.4.47 WkbSize()

```
int OGRCompoundCurve::WkbSize ( ) const [override], [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.

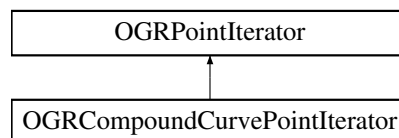
Implements **OGRGeometry** (p. ??).

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

- **ogr\_geometry.h**
- **ogrcompoundcurve.cpp**

## 11.57 OGRCompoundCurvePointIterator Class Reference

Inheritance diagram for OGRCompoundCurvePointIterator:

**Public Member Functions**

- virtual **OGRBoolean** **getNextPoint** ( **OGRPoint** \*p) override  
*Returns the next point followed by the iterator.*

**Additional Inherited Members**

### 11.57.1 Member Function Documentation

#### 11.57.1.1 getNextPoint()

```
OGRBoolean OGRCompoundCurvePointIterator::getNextPoint (
    OGRPoint * p ) [override], [virtual]
```

Returns the next point followed by the iterator.

**Parameters**

<i>p</i>	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()`.

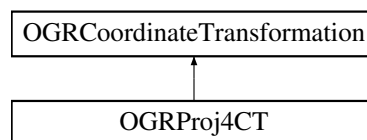
The documentation for this class was generated from the following file:

- ogrcompoundcurve.cpp

## 11.58 OGRCoordinateTransformation Class Reference

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

Inheritance diagram for OGRCoordinateTransformation:

**Public Member Functions**

- virtual **OGRSpatialReference \*** **GetSourceCS** ()=0
- virtual **OGRSpatialReference \*** **GetTargetCS** ()=0
- virtual bool **GetEmitErrors** ()
- virtual void **SetEmitErrors** (bool)
- virtual int **Transform** (int nCount, double \*x, double \*y, double \*z=nullptr)=0
- virtual int **TransformEx** (int nCount, double \*x, double \*y, double \*z=nullptr, int \*pabSuccess=nullptr)=0

**Static Public Member Functions**

- static void **DestroyCT** ( **OGRCoordinateTransformation \***poCT)  
**OGRCoordinateTransformation** (p. ??) destructor.
- static **OGRCoordinateTransformationH** **ToHandle** ( **OGRCoordinateTransformation \***poCT)
- static **OGRCoordinateTransformation \*** **FromHandle** ( **OGRCoordinateTransformationH** hCT)

### 11.58.1 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.

### 11.58.2 Member Function Documentation

#### 11.58.2.1 DestroyCT()

```
void OGRCoordinateTransformation::DestroyCT (
    OGRCoordinateTransformation * poCT ) [static]
```

**OGRCoordinateTransformation** (p. ??) destructor.

This function is the same as `OGRCoordinateTransformation::~~OGRCoordinateTransformation()` and **OCT↔DestroyCoordinateTransformation()** (p. ??)

This static method will destroy a **OGRCoordinateTransformation** (p. ??). It is equivalent to calling delete on the object, but it ensures that the deallocation is properly executed within the OGR libraries heap on platforms where this can matter (win32).

#### Parameters

<i>poCT</i>	the object to delete
-------------	----------------------

#### Since

GDAL 1.7.0

#### 11.58.2.2 FromHandle()

```
static OGRCoordinateTransformation* OGRCoordinateTransformation::FromHandle (
    OGRCoordinateTransformationH hCT ) [inline], [static]
```

Convert a OGRCoordinateTransformationH to a OGRCoordinateTransformation\*.

#### Since

GDAL 2.3

Referenced by `OCTDestroyCoordinateTransformation()`, and `OGR_G_Transform()`.

### 11.58.2.3 GetEmitErrors()

```
virtual bool OGRCoordinateTransformation::GetEmitErrors ( ) [inline], [virtual]
```

Whether the transformer will emit CPLError

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

### 11.58.2.4 GetSourceCS()

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

Fetch internal source coordinate system.

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

Referenced by OGRGeometryFactory::transformWithOptions().

### 11.58.2.5 GetTargetCS()

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

Fetch internal target coordinate system.

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

Referenced by OGRPoint::transform(), OGRGeometryCollection::transform(), and OGRGeometryFactory↵  
::transformWithOptions().

### 11.58.2.6 SetEmitErrors()

```
virtual void OGRCoordinateTransformation::SetEmitErrors (
    bool ) [inline], [virtual]
```

Set if the transformer must emit CPLError

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

### 11.58.2.7 ToHandle()

```
static  OGRCoordinateTransformationH OGRCoordinateTransformation::ToHandle (
    OGRCoordinateTransformation * poCT ) [inline], [static]
```

Convert a OGRCoordinateTransformation\* to a OGRCoordinateTransformationH.

Since

GDAL 2.3

### 11.58.2.8 Transform()

```
virtual int OGRCoordinateTransformation::Transform (
    int nCount,
    double * x,
    double * y,
    double * z = nullptr ) [pure virtual]
```

Transform points from source to destination space.

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

The method **TransformEx()** (p. ??) allows extended success information to be captured indicating which points failed to transform.

#### Parameters

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.

#### Returns

TRUE on success, or FALSE if some or all points fail to transform.

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

Referenced by OGRPoint::transform().

### 11.58.2.9 TransformEx()

```
virtual int OGRCoordinateTransformation::TransformEx (
    int nCount,
    double * x,
```

```
double * y,
double * z = nullptr,
int * pabSuccess = nullptr ) [pure virtual]
```

Transform points from source to destination space.

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

#### Parameters

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.
<i>pabSuccess</i>	array of per-point flags set to TRUE if that point transforms, or FALSE if it does not.

#### Returns

TRUE if some or all points transform successfully, or FALSE if none transform.

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

Referenced by OGRSimpleCurve::transform().

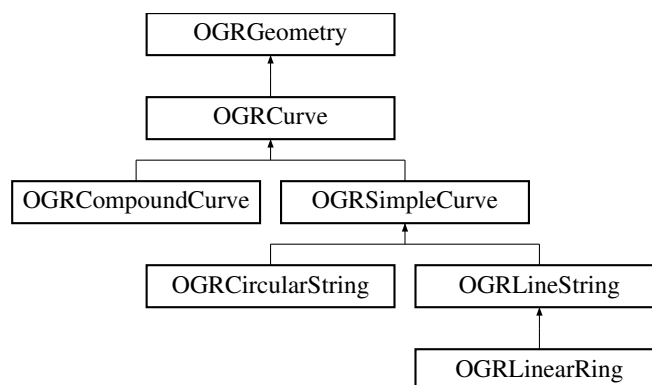
The documentation for this class was generated from the following files:

- **ogr\_spatialref.h**
- **ogrct.cpp**

## 11.59 OGRCurve Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRCurve:



#### Public Types

- typedef **OGRPoint** **ChildType**

## Public Member Functions

- ConstIterator **begin** () const
- ConstIterator **end** () const
- 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 \*papszOptions=NULLPTR) const =0  
*Return a linestring from a curve geometry.*
- virtual int **getDimension** () const override  
*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.*
- **OGRSimpleCurve** \* **toSimpleCurve** ()
- const **OGRSimpleCurve** \* **toSimpleCurve** () const

## 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 int **IntersectsPoint** (const **OGRPoint** \*p) const  
*Returns if a point intersects a (closed) curve.*
- virtual double **get\_AreaOfCurveSegments** () const =0  
*Get the area of the purely curve portions of a (closed) curve.*

### 11.59.1 Detailed Description

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

### 11.59.2 Member Typedef Documentation

#### 11.59.2.1 ChildType

```
typedef OGRPoint OGRCurve::ChildType
```

Type of child elements.

### 11.59.3 Member Function Documentation

#### 11.59.3.1 begin()

```
OGRCurve::ConstIterator OGRCurve::begin ( ) const
```

Return begin of a point iterator.

Using this iterator for standard range-based loops is safe, but due to implementation limitations, you shouldn't try to access (dereference) more than one iterator step at a time, since you will get a reference to the same **OGRPoint** (p. ??)& object.

Since

GDAL 2.3

#### 11.59.3.2 CastToCompoundCurve()

```
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

<i>poCurve</i>	the input geometry - ownership is passed to the method.
----------------	---

**Returns**

new geometry

**Since**

GDAL 2.0

< Success

References `OGRCompoundCurve::addCurveDirectly()`, `OGRCompoundCurve::assignSpatialReference()`, `CastToLineString()`, `OGRGeometry::getGeometryType()`, `OGRGeometry::getSpatialReference()`, `OGRGeometry::IsEmpty()`, `OGRERR_NONE`, and `wkbLineString`.

**11.59.3.3 CastToLinearRing()**

```
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**

<i>poCurve</i>	the input geometry - ownership is passed to the method.
----------------	---

**Returns**

new geometry.

**Since**

GDAL 2.0

Referenced by `OGRCurvePolygon::CastToPolygon()`, `OGRCurvePolygon::CurvePolyToPoly()`, and `OGRGeometryFactory::forceToPolygon()`.

**11.59.3.4 CastToLineString()**

```
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

<i>poCurve</i>	the input geometry - ownership is passed to the method.
----------------	---

## Returns

new geometry.

## Since

GDAL 2.0

Referenced by `CastToCompoundCurve()`, `OGRMultiCurve::CastToMultiLineString()`, `OGRGeometryFactory::forceTo()`, and `OGRGeometryFactory::forceToLineString()`.

## 11.59.3.5 ContainsPoint()

```
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

<i>p</i>	the point to test
----------	-------------------

## Returns

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

## Since

GDAL 2.0

## 11.59.3.6 CurveToLine()

```
OGRLineString * OGRCurve::CurveToLine (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) 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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings or NULL. See <b>OGRGeometryFactory::curveToLineString()</b> (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 **OGRGeometryFactory::forceToLineString()**, **OGRGeometryFactory::forceToMultiLineString()**, and **OGRGeometryFactory::forceToPolygon()**.

## 11.59.3.7 end()

```
OGRCurve::ConstIterator OGRCurve::end ( ) const
```

Return end of a point iterator.

## 11.59.3.8 EndPoint()

```
void OGRCurve::EndPoint (
    OGRPoint * poPoint ) const [pure virtual]
```

Return the curve end point.

This method relates to the SF COM ICurve::get\_EndPoint() method.

## Parameters

<i>poPoint</i>	the point to be assigned the end location.
----------------	--

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

Referenced by **get\_IsClosed()**.

## 11.59.3.9 get\_Area()

```
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()**, and **OGRGeometryCollection::get\_Area()**.

#### 11.59.3.10 get\_AreaOfCurveSegments()

```
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. ??).

Referenced by **OGRCircularString::get\_Area()**, and **OGRCompoundCurve::get\_AreaOfCurveSegments()**.

#### 11.59.3.11 get\_IsClosed()

```
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.

For equality tests, the M dimension is ignored.

This method relates to the **SFCOM ICurve::get\_IsClosed()** method.

#### Returns

TRUE if closed, else FALSE.

References **EndPoint()**, **OGRPoint::getX()**, **OGRPoint::getY()**, **OGRPoint::getZ()**, **OGRGeometry::Is3D()**, and **StartPoint()**.

Referenced by **OGRLineString::CastToLinearRing()**, **OGRGeometryFactory::forceToPolygon()**, **OGRCircularString::get\_Area()**, **OGRCompoundCurve::get\_Area()**, and **OGRTriangle::OGRTriangle()**.

11.59.3.12 `get_Length()`

```
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 **OGRCCompoundCurve** (p. ??), **OGRCircularString** (p. ??), and **OGRSimpleCurve** (p. ??).

Referenced by `OGRGeometryCollection::get_Length()`.

11.59.3.13 `getDimension()`

```
int OGRCurve::getDimension ( ) const [override], [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 **OGRCurve::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 **OGRCurve** (p. ??).

11.59.3.14 `getNumPoints()`

```
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 **OGRCCompoundCurve** (p. ??), and **OGRSimpleCurve** (p. ??).

Referenced by `OGRGeometry::dumpReadable()`, `OGRGeometryFactory::forceToPolygon()`, `OGR_G_GetPointCount()`, and `OGRTriangle::OGRTriangle()`.

11.59.3.15 `getPointIterator()`

```
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()`.

11.59.3.16 `IntersectsPoint()`

```
int OGRCurve::IntersectsPoint (
    const OGRPoint * p ) const [protected], [virtual]
```

Returns if a point intersects a (closed) curve.

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

**Parameters**

<i>p</i>	the point to test
----------	-------------------

**Returns**

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

**Since**

GDAL 2.3

## 11.59.3.17 IsConvex()

```
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()`.

## 11.59.3.18 StartPoint()

```
void OGRCurve::StartPoint (
    OGRPoint * poPoint ) const [pure virtual]
```

Return the curve start point.

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

## Parameters

<i>poPoint</i>	the point to be assigned the start location.
----------------	--

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

Referenced by `get_IsClosed()`.

## 11.59.3.19 toSimpleCurve() [1/2]

```
OGRSimpleCurve* OGRCurve::toSimpleCurve ( ) [inline]
```

Down-cast to `OGRSimpleCurve*`. Implies prior checking that `wkbFlatten(getGeometryType())` (p. ??) == `wkbLineString` or `wkbCircularString`.

### 11.59.3.20 toSimpleCurve() [2/2]

```
const OGRSimpleCurve* OGRCurve::toSimpleCurve ( ) const [inline]
```

Down-cast to OGRSimpleCurve\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkb↔ LineString or wkbCircularString.

### 11.59.3.21 Value()

```
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

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

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

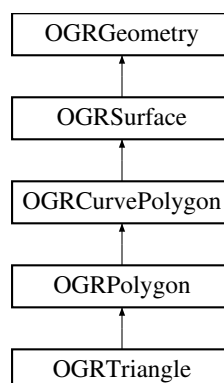
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrcurve.cpp**

## 11.60 OGRCurvePolygon Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRCurvePolygon:





## Public Types

- typedef **OGRCurve ChildType**

## Public Member Functions

- **OGRCurvePolygon** ()  
*Create an empty curve polygon.*
- **OGRCurvePolygon** (const **OGRCurvePolygon** &)  
*Copy constructor.*
- **OGRCurvePolygon** & **operator=** (const **OGRCurvePolygon** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Fetch geometry type.*
- virtual **OGRGeometry** \* **clone** () const override  
*Make a copy of this object.*
- virtual void **empty** () override  
*Clear geometry information. This restores the geometry to its initial state after construction, and before assignment of actual geometry.*
- virtual **OGRErr** **transform** (**OGRCoordinateTransformation** \*poCT) override  
*Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** () override  
*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual **OGRBoolean** **IsEmpty** () const override  
*Returns TRUE (non-zero) if the object has no points.*
- virtual void **segmentize** (double dfMaxLength) override  
*Modify the geometry such it has no segment longer then the given distance.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override  
*Returns if this geometry is or has curve geometry.*
- virtual **OGRGeometry** \* **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*papszOptions=NULLptr) const override  
*Return, possibly approximate, non-curve version of this geometry.*
- virtual double **get\_Area** () const override  
*Get the area of the surface object.*
- virtual int **WkbSize** () const override  
*Returns size of related binary representation.*
- virtual **OGRErr** **importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumedOut) override  
*Assign geometry from well known binary data.*
- virtual **OGRErr** **exportToWkb** (**OGRwkbByteOrder**, unsigned char \*, **OGRwkbVariant**= **wkbVariant↔OldOgc**) const override  
*Convert a geometry into well known binary format.*
- **OGRErr** **importFromWkt** (const char \*\*) override
- virtual **OGRErr** **exportToWkt** (char \*\*papszDstText, **OGRwkbVariant** eWkbVariant= **wkbVariantOldOgc**) const override

- Convert a geometry into well known text format.*

  - virtual int **getDimension** () const override

*Get the dimension of this object.*
- virtual void **getEnvelope** (OGREnvelope \*psEnvelope) const override

*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (OGREnvelope3D \*psEnvelope) const override

*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=nullptr) const

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

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

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

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

*Set the coordinate dimension.*
- virtual void **set3D** ( **OGRBoolean** bls3D) override

*Add or remove the Z coordinate dimension.*
- virtual void **setMeasured** ( **OGRBoolean** blsMeasured) override

*Add or remove the M coordinate dimension.*
- virtual void **assignSpatialReference** ( **OGRSpatialReference** \*poSR) override

*Assign spatial reference to this object.*
- virtual **OGRErr** **addRing** ( **OGRCurve** \*)

*Add a ring to a polygon.*
- virtual **OGRErr** **addRingDirectly** ( **OGRCurve** \*)

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

*Fetch reference to external polygon ring.*
- const **OGRCurve** \* **getExteriorRingCurve** () const

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

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

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

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

*"Steal" reference to external ring.*
- **OGRErr** **removeRing** (int iIndex, bool bDelete=true)

*Remove a geometry from the container.*
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override
- virtual void **swapXY** () override

*Swap x and y coordinates.*
- virtual **OGRErr** **importFromWkt** (const char \*\*ppszInput)=0

*Assign geometry from well known text data.*
- **OGRErr** **importFromWkt** (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

## Static Protected Member Functions

- static **OGRPolygon** \* **CastToPolygon** ( **OGRCurvePolygon** \*poCP)  
*Convert to polygon.*

## Additional Inherited Members

### 11.60.1 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

### 11.60.2 Member Typedef Documentation

#### 11.60.2.1 ChildType

```
typedef OGRCurve OGRCurvePolygon::ChildType
```

Type of child elements.

### 11.60.3 Constructor & Destructor Documentation

#### 11.60.3.1 OGRCurvePolygon()

```
OGRCurvePolygon::OGRCurvePolygon (  
    const OGRCurvePolygon & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

## 11.60.4 Member Function Documentation

### 11.60.4.1 `accept()` [1/2]

```
virtual void OGRCurvePolygon::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

Reimplemented in **OGRTriangle** (p. ??), and **OGRPolygon** (p. ??).

References IOGRGeometryVisitor::visit().

### 11.60.4.2 `accept()` [2/2]

```
virtual void OGRCurvePolygon::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

Reimplemented in **OGRTriangle** (p. ??), and **OGRPolygon** (p. ??).

References IOGRConstGeometryVisitor::visit().

### 11.60.4.3 `addRing()`

```
OGRERR OGRCurvePolygon::addRing (
    OGRCurve * poNewRing ) [virtual]
```

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

<i>poNewRing</i>	ring to be added to the polygon.
------------------	----------------------------------

**Returns**

OGRERR\_NONE in case of success

< Success

References `addRingDirectly()`, `OGRGeometry::clone()`, `OGRERR_NONE`, and `OGRGeometry::toCurve()`.

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

**11.60.4.4 addRingDirectly()**

```
OGRErr OGRCurvePolygon::addRingDirectly (
    OGRCurve * poNewRing ) [virtual]
```

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**

<i>poNewRing</i>	ring to be added to the polygon.
------------------	----------------------------------

**Returns**

OGRERR\_NONE in case of success

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

Referenced by `addRing()`, `OGRGeometryFactory::forceToPolygon()`, and `OGRPolygon::getCurveGeometry()`.

**11.60.4.5 assignSpatialReference()**

```
void OGRCurvePolygon::assignSpatialReference (
    OGRSpatialReference * poSR ) [override], [virtual]
```

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.

Starting with GDAL 2.3, this will also assign the spatial reference to potential sub-geometries of the geometry (**OGRGeometryCollection** (p. ??), `OGRCurvePolygon/OGRPolygon`, **OGRCompoundCurve** (p. ??), **OGRPolyhedralSurface** (p. ??) and their derived classes).

This is similar to the SFCOM `IGeometry::put_SpatialReference()` method.

This method is the same as the C function **OGR\_G\_AssignSpatialReference()** (p. ??).

## Parameters

<i>poSR</i>	new spatial reference system to apply.
-------------	--

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

Referenced by CastToPolygon(), clone(), CurvePolyToPoly(), OGRGeometryFactory::forceToPolygon(), and OGRPolygon::getCurveGeometry().

#### 11.60.4.6 begin() [1/2]

```
ChildType** OGRCurvePolygon::begin ( ) [inline]
```

Return begin of curve iterator.

Since

GDAL 2.3

#### 11.60.4.7 begin() [2/2]

```
const ChildType* const* OGRCurvePolygon::begin ( ) const [inline]
```

Return begin of curve iterator.

Since

GDAL 2.3

#### 11.60.4.8 CastToPolygon()

```
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

<i>poCP</i>	the input geometry - ownership is passed to the method.
-------------	---

## Returns

new geometry.

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

Referenced by `OGRGeometryFactory::forceTo()`.

## 11.60.4.9 clone()

```
OGRGeometry * OGRCurvePolygon::clone ( ) const [override], [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.

< Success

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

References `addRing()`, `assignSpatialReference()`, `OGRGeometryFactory::createGeometry()`, `getGeometryType()`, `OGRGeometry::getSpatialReference()`, `OGRERR_NONE`, and `OGRGeometry::toCurvePolygon()`.

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

## 11.60.4.10 Contains()

```
OGRBoolean OGRCurvePolygon::Contains (
    const OGRGeometry * ) const [override], [virtual]
```

Test for containment.

Tests if actual geometry object contains the passed geometry.

This method is the same as the C function **OGR\_G\_Contains()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a `CPLE_NotSupported` error.

## Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

## Returns

TRUE if poOtherGeom contains this geometry, otherwise FALSE.

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

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

Referenced by OGRPoint::Within().

## 11.60.4.11 CurvePolyToPoly()

```
OGRPolygon * OGRCurvePolygon::CurvePolyToPoly (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) 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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	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 assignSpatialReference(), OGRCurve::CastToLinearRing(), CPLError(), OGRLineString::CurveToLine(), and OGRGeometry::getSpatialReference().

Referenced by OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), and getLinearGeometry().



## 11.60.4.12 empty()

```
void OGRCurvePolygon::empty ( ) [override], [virtual]
```

Clear geometry information. This restores the geometry to its 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 OGRTriangle::importFromWkb().

## 11.60.4.13 end() [1/2]

```
ChildType** OGRCurvePolygon::end ( ) [inline]
```

Return end of curve iterator.

## 11.60.4.14 end() [2/2]

```
const ChildType* const* OGRCurvePolygon::end ( ) const [inline]
```

Return end of curve iterator.

## 11.60.4.15 Equals()

```
OGRBoolean OGRCurvePolygon::Equals (
    const OGRGeometry * ) const [override], [virtual]
```

Returns TRUE if two geometries are equivalent.

This operation implements the SQL/MM ST\_OrderingEquals() operation.

The comparison is done in a structural way, that is to say that the geometry types must be identical, as well as the number and ordering of sub-geometries and vertices. Or equivalently, two geometries are considered equal by this method if their WKT/WKB representation is equal. Note: this must be distinguished for equality in a spatial way (which is the purpose of the ST\_Equals() operation).

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(), IsEmpty(), and OGRGeometry::toCurvePolygon().

## 11.60.4.16 exportToWkb()

```

OGRERR OGRCurvePolygon::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
<i>eWkbVariant</i>	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.

## 11.60.4.17 exportToWkt()

```

OGRERR OGRCurvePolygon::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]

```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> </ul>
	<ul style="list-style-type: none"> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

**Returns**

Currently OGRERR\_NONE is always returned.

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

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

**11.60.4.18 flattenTo2D()**

```
void OGRCurvePolygon::flattenTo2D ( ) [override], [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. ??).

**11.60.4.19 get\_Area()**

```
double OGRCurvePolygon::get_Area ( ) const [override], [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().

**11.60.4.20 getDimension()**

```
int OGRCurvePolygon::getDimension ( ) const [override], [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. ??).

**11.60.4.21 getEnvelope()** [1/2]

```
void OGRCurvePolygon::getEnvelope (
    OGREnvelope * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

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

11.60.4.22 **getEnvelope()** [2/2]

```
void OGRCurvePolygon::getEnvelope (
    OGREnvelope3D * psEnvelope ) const  [override], [virtual]
```

Computes and returns the bounding envelope (3D) for this geometry in the passed *psEnvelope* structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

## Since

OGR 1.9.0

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

11.60.4.23 **getExteriorRingCurve()** [1/2]

```
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 Simple Features for COM (SFCOM) IPolygon::get\_ExteriorRing() method. TODO(rouault): What does that mean?

## Returns

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

Referenced by **OGRGeometry::dumpReadable()**, **get\_Area()**, and **OGRTriangle::OGRTriangle()**.

11.60.4.24 `getExteriorRingCurve()` [2/2]

```
const OGRCurve * OGRCurvePolygon::getExteriorRingCurve ( ) const
```

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.

11.60.4.25 `getGeometryName()`

```
const char * OGRCurvePolygon::getGeometryName ( ) const [override], [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 **OGRTriangle** (p. ??), and **OGRPolygon** (p. ??).

Referenced by `segmentize()`.

11.60.4.26 `getGeometryType()`

```
OGRwkbGeometryType OGRCurvePolygon::getGeometryType ( ) const [override], [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 **OGRTriangle** (p. ??), and **OGRPolygon** (p. ??).

References `wkbCurvePolygon`, `wkbCurvePolygonM`, `wkbCurvePolygonZ`, and `wkbCurvePolygonZM`.

Referenced by `clone()`, and `Equals()`.

11.60.4.27 `getInteriorRingCurve()` [1/2]

```
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

<i>iRing</i>	internal ring index from 0 to <b>getNumInteriorRings()</b> (p. ??) - 1.
--------------	---

## Returns

pointer to interior ring. May be NULL.

Referenced by OGRGeometry::dumpReadable(), and get\_Area().

11.60.4.28 `getInteriorRingCurve()` [2/2]

```
const OGRCurve * OGRCurvePolygon::getInteriorRingCurve (
    int iRing ) const
```

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

<i>iRing</i>	internal ring index from 0 to <b>getNumInteriorRings()</b> (p. ??) - 1.
--------------	---

## Returns

pointer to interior ring. May be NULL.

## 11.60.4.29 getLinearGeometry()

```
OGRGeometry * OGRCurvePolygon::getLinearGeometry (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings. See <b>OGRGeometryFactory::curveToLineString()</b> (p. ??) for valid options.

## Returns

a new geometry.

## Since

GDAL 2.0

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

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

References [CurvePolyToPoly\(\)](#).

## 11.60.4.30 getNumInteriorRings()

```
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::forceToLineString\(\)](#), [OGRGeometryFactory::forceToMultiLineString\(\)](#), [OGRGeometryFactory::forceToPolygon\(\)](#), [getArea\(\)](#), and [OGRTriangle::OGRTriangle\(\)](#).

## 11.60.4.31 hasCurveGeometry()

```
OGRBoolean OGRCurvePolygon::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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

<i>bLookForNonLinear</i>	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. ??).

## 11.60.4.32 importFromWkb()

```
OGRERR OGRCurvePolygon::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).



## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	if wkbVariantPostGIS1, special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

## Since

GDAL 2.3

< Success

< Success

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

Reimplemented in **OGRTriangle** (p. ??), and **OGRPolygon** (p. ??).

References OGRERR\_NONE.

## 11.60.4.33 importFromWkt() [1/3]

**OGRERR** OGRGeometry::importFromWkt

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppsInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
-----------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

11.60.4.34 `importFromWkt()` [2/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

11.60.4.35 `importFromWkt()` [3/3]

```
OGRERR OGRCurvePolygon::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated

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

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

11.60.4.36 `Intersects()`

```
OGRBoolean OGRCurvePolygon::Intersects (
    const OGRGeometry * poOtherGeom ) const [override], [virtual]
```

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigorous fashion otherwise TRUE is returned if the envelopes (bounding boxes) of the two geometries overlap.

The *poOtherGeom* argument may be safely NULL, but in this case the method will always return TRUE. That is, a NULL geometry is treated as being everywhere.

This method is the same as the C function **OGR\_G\_Intersects()** (p. ??).

#### Parameters

<i>poOtherGeom</i>	the other geometry to test against.
--------------------	-------------------------------------

#### Returns

TRUE if the geometries intersect, otherwise FALSE.

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

References `OGRGeometry::getGeometryType()`, `OGRGeometry::Intersects()`, `IsEmpty()`, `OGRGeometry::toPoint()`, `wkbFlatten`, and `wkbPoint`.

Referenced by `OGRPoint::Intersects()`.

## 11.60.4.37 isEmpty()

```
OGRBoolean OGRCurvePolygon::IsEmpty ( ) const [override], [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().

## 11.60.4.38 operator=()

```
OGRCurvePolygon & OGRCurvePolygon::operator= (
    const OGRCurvePolygon & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

## Since

GDAL 2.1

References OGRGeometry::operator=().

Referenced by OGRPolygon::operator=().

## 11.60.4.39 removeRing()

```
OGRERR OGRCurvePolygon::removeRing (
    int iIndex,
    bool bDelete = true )
```

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.

## Parameters

<i>iIndex</i>	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
<i>bDelete</i>	if true the geometry will be deallocated, otherwise it will not. The default is true as the container is considered to own the geometries in it.

## Returns

OGRERR\_NONE if successful, or OGRERR\_FAILURE if the index is out of range.

## 11.60.4.40 segmentize()

```
void OGRCurvePolygon::segmentize (
    double dfMaxLength ) [override], [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

<i>dfMaxLength</i>	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 consecutive vertices is lower than dfMaxLength.

## Parameters

<i>dfMaxLength</i>	maximum length between 2 consecutive vertices.
--------------------	--

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

References CPL\_Error(), EQUAL, and getGeometryName().

## 11.60.4.41 set3D()

```
void OGRCurvePolygon::set3D (
    OGRBoolean bIs3D ) [override], [virtual]
```

Add or remove the Z coordinate dimension.

This method adds or removes the explicit Z coordinate dimension. Removing the Z coordinate dimension of a geometry will remove any existing Z values. Adding the Z dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

## Parameters

<i>bIs3D</i>	Should the geometry have a Z dimension, either TRUE or FALSE.
--------------	---

## Since

GDAL 2.1

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

## 11.60.4.42 setCoordinateDimension()

```
void OGRCurvePolygon::setCoordinateDimension (
    int nNewDimension ) [override], [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, a compound curve, a polygon, etc. will affect the children geometries. This will also remove the M dimension if present before this call.

**Deprecated** use **set3D()** (p. ??) or **setMeasured()** (p. ??).

## Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

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

Referenced by CastToPolygon().

## 11.60.4.43 setMeasured()

```
void OGRCurvePolygon::setMeasured (
    OGRBoolean bIsMeasured ) [override], [virtual]
```

Add or remove the M coordinate dimension.

This method adds or removes the explicit M coordinate dimension. Removing the M coordinate dimension of a geometry will remove any existing M values. Adding the M dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

## Parameters

<i>bIsMeasured</i>	Should the geometry have a M dimension, either TRUE or FALSE.
--------------------	---

Since

GDAL 2.1

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

#### 11.60.4.44 stealExteriorRingCurve()

```
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::forceToLineString(), and OGRPolygon::stealExteriorRing().

#### 11.60.4.45 swapXY()

```
void OGRCurvePolygon::swapXY ( ) [override], [virtual]
```

Swap x and y coordinates.

Since

OGR 1.8.0

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

#### 11.60.4.46 transform()

```
OGRERR OGRCurvePolygon::transform (
    OGRCoordinateTransformation * poCT ) [override], [virtual]
```

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGR↔SpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

## Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

## Returns

OGRERR\_NONE on success or an error code.

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

## 11.60.4.47 WkbSize()

```
int OGRCurvePolygon::WkbSize ( ) const [override], [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. ??).

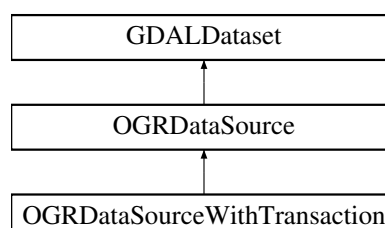
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrcurvepolygon.cpp**

## 11.61 OGRDataSource Class Reference

```
#include <ogr_sfrmts.h>
```

Inheritance diagram for OGRDataSource:



### 11.61.1 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 its 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\*.

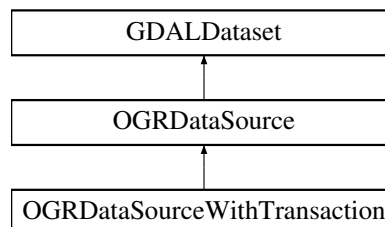
#### Deprecated

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

- **ogrsf\_frmts.h**
- ogrdatasource.cpp

## 11.62 OGRDataSourceWithTransaction Class Reference

Inheritance diagram for OGRDataSourceWithTransaction:



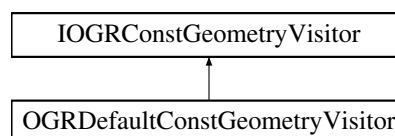
The documentation for this class was generated from the following file:

- ocremulatedtransaction.cpp

## 11.63 OGRDefaultConstGeometryVisitor Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRDefaultConstGeometryVisitor:





## Public Member Functions

- void **visit** (const **OGRPoint** \*) override
- void **visit** (const **OGRLineString** \*) override
- void **visit** (const **OGRLinearRing** \*) override
- void **visit** (const **OGRPolygon** \*) override
- void **visit** (const **OGRMultiPoint** \*) override
- void **visit** (const **OGRMultiLineString** \*) override
- void **visit** (const **OGRMultiPolygon** \*) override
- void **visit** (const **OGRGeometryCollection** \*) override
- void **visit** (const **OGRCircularString** \*) override
- void **visit** (const **OGRCompoundCurve** \*) override
- void **visit** (const **OGRCurvePolygon** \*) override
- void **visit** (const **OGRMultiCurve** \*) override
- void **visit** (const **OGRMultiSurface** \*) override
- void **visit** (const **OGRTriangle** \*) override
- void **visit** (const **OGRPolyhedralSurface** \*) override
- void **visit** (const **OGRTriangulatedSurface** \*) override

### 11.63.1 Detailed Description

**OGRGeometry** (p. ??) visitor default implementation.

This default implementation will recurse down to calling **visit(const OGRPoint\*)** (p. ??) on each point.

Since

GDAL 2.3

### 11.63.2 Member Function Documentation

#### 11.63.2.1 visit() [1/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRPoint * ) [inline], [override], [virtual]
```

Visit **OGRPoint** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

Referenced by visit().

### 11.63.2.2 visit() [2/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRLineString * ) [override], [virtual]
```

Visit **OGRLineString** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

### 11.63.2.3 visit() [3/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRLinearRing * ) [override], [virtual]
```

Visit **OGRLinearRing** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRLinearRing::toUpperClass()**, and **visit()**.

### 11.63.2.4 visit() [4/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRPolygon * ) [override], [virtual]
```

Visit **OGRPolygon** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRPolygon::toUpperClass()**, and **visit()**.

### 11.63.2.5 visit() [5/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRMultiPoint * ) [override], [virtual]
```

Visit **OGRMultiPoint** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRMultiPoint::toUpperClass()**, and **visit()**.

#### 11.63.2.6 visit() [6/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRMultiLineString * ) [override], [virtual]
```

Visit **OGRMultiLineString** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRMultiLineString::toUpperClass()**, and **visit()**.

#### 11.63.2.7 visit() [7/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRMultiPolygon * ) [override], [virtual]
```

Visit **OGRMultiPolygon** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRMultiPolygon::toUpperClass()**, and **visit()**.

#### 11.63.2.8 visit() [8/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRGeometryCollection * ) [override], [virtual]
```

Visit **OGRGeometryCollection** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

#### 11.63.2.9 visit() [9/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRCircularString * ) [override], [virtual]
```

Visit **OGRCircularString** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

**11.63.2.10 visit()** [10/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRCompoundCurve * ) [override], [virtual]
```

Visit **OGRCompoundCurve** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

**11.63.2.11 visit()** [11/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRCurvePolygon * ) [override], [virtual]
```

Visit **OGRCurvePolygon** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

**11.63.2.12 visit()** [12/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRMultiCurve * ) [override], [virtual]
```

Visit **OGRMultiCurve** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRMultiCurve::toUpperClass()**, and **visit()**.

**11.63.2.13 visit()** [13/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRMultiSurface * ) [override], [virtual]
```

Visit **OGRMultiSurface** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRMultiSurface::toUpperClass()**, and **visit()**.

**11.63.2.14 visit()** [14/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRTriangle * ) [override], [virtual]
```

Visit **OGRTriangle** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRTriangle::toUpperClass()**, and **visit()**.

**11.63.2.15 visit()** [15/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRPolyhedralSurface * ) [override], [virtual]
```

Visit **OGRPolyhedralSurface** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

**11.63.2.16 visit()** [16/16]

```
void OGRDefaultConstGeometryVisitor::visit (
    const OGRTriangulatedSurface * ) [override], [virtual]
```

Visit **OGRTriangulatedSurface** (p. ??).

Implements **IOGRConstGeometryVisitor** (p. ??).

References **OGRTriangulatedSurface::toUpperClass()**, and **visit()**.

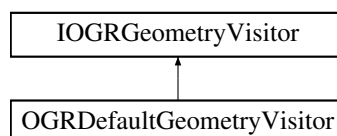
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrgeometry.cpp**

**11.64 OGRDefaultGeometryVisitor Class Reference**

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRDefaultGeometryVisitor:



## Public Member Functions

- void **visit** ( **OGRPoint** \*) override
- void **visit** ( **OGRLineString** \*) override
- void **visit** ( **OGRLinearRing** \*) override
- void **visit** ( **OGRPolygon** \*) override
- void **visit** ( **OGRMultiPoint** \*) override
- void **visit** ( **OGRMultiLineString** \*) override
- void **visit** ( **OGRMultiPolygon** \*) override
- void **visit** ( **OGRGeometryCollection** \*) override
- void **visit** ( **OGRCircularString** \*) override
- void **visit** ( **OGRCompoundCurve** \*) override
- void **visit** ( **OGRCurvePolygon** \*) override
- void **visit** ( **OGRMultiCurve** \*) override
- void **visit** ( **OGRMultiSurface** \*) override
- void **visit** ( **OGRTriangle** \*) override
- void **visit** ( **OGRPolyhedralSurface** \*) override
- void **visit** ( **OGRTriangulatedSurface** \*) override

### 11.64.1 Detailed Description

**OGRGeometry** (p. ??) visitor default implementation.

This default implementation will recurse down to calling **visit(OGRPoint\*)** (p. ??) on each point.

Since

GDAL 2.3

### 11.64.2 Member Function Documentation

#### 11.64.2.1 visit() [1/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRPoint * ) [inline], [override], [virtual]
```

Visit **OGRPoint** (p. ??).

Implements **OGRGeometryVisitor** (p. ??).

Referenced by visit().

### 11.64.2.2 visit() [2/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRLineString * ) [override], [virtual]
```

Visit **OGRLineString** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

### 11.64.2.3 visit() [3/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRLinearRing * ) [override], [virtual]
```

Visit **OGRLinearRing** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRLinearRing::toUpperClass()**, and **visit()**.

### 11.64.2.4 visit() [4/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRPolygon * ) [override], [virtual]
```

Visit **OGRPolygon** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRPolygon::toUpperClass()**, and **visit()**.

### 11.64.2.5 visit() [5/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRMultiPoint * ) [override], [virtual]
```

Visit **OGRMultiPoint** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRMultiPoint::toUpperClass()**, and **visit()**.

#### 11.64.2.6 visit() [6/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRMultiLineString * ) [override], [virtual]
```

Visit **OGRMultiLineString** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRMultiLineString::toUpperClass()**, and **visit()**.

#### 11.64.2.7 visit() [7/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRMultiPolygon * ) [override], [virtual]
```

Visit **OGRMultiPolygon** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRMultiPolygon::toUpperClass()**, and **visit()**.

#### 11.64.2.8 visit() [8/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRGeometryCollection * ) [override], [virtual]
```

Visit **OGRGeometryCollection** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

#### 11.64.2.9 visit() [9/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRCircularString * ) [override], [virtual]
```

Visit **OGRCircularString** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).



**11.64.2.10 visit()** [10/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRCompoundCurve * ) [override], [virtual]
```

Visit **OGRCompoundCurve** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

**11.64.2.11 visit()** [11/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRCurvePolygon * ) [override], [virtual]
```

Visit **OGRCurvePolygon** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

**11.64.2.12 visit()** [12/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRMultiCurve * ) [override], [virtual]
```

Visit **OGRMultiCurve** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRMultiCurve::toUpperClass()**, and **visit()**.

**11.64.2.13 visit()** [13/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRMultiSurface * ) [override], [virtual]
```

Visit **OGRMultiSurface** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRMultiSurface::toUpperClass()**, and **visit()**.

#### 11.64.2.14 visit() [14/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRTriangle * ) [override], [virtual]
```

Visit **OGRTriangle** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRTriangle::toUpperClass()**, and **visit()**.

#### 11.64.2.15 visit() [15/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRPolyhedralSurface * ) [override], [virtual]
```

Visit **OGRPolyhedralSurface** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

#### 11.64.2.16 visit() [16/16]

```
void OGRDefaultGeometryVisitor::visit (
    OGRTriangulatedSurface * ) [override], [virtual]
```

Visit **OGRTriangulatedSurface** (p. ??).

Implements **IOGRGeometryVisitor** (p. ??).

References **OGRTriangulatedSurface::toUpperClass()**, and **visit()**.

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

- **ogr\_geometry.h**
- **ogrgeometry.cpp**

## 11.65 OGRFeature Class Reference

```
#include <ogr_feature.h>
```

### Classes

- class **ConstFieldIterator**
- class **FieldNotFoundException**
- class **FieldValue**

## Public Member Functions

- **OGRFeature** ( **OGRFeatureDefn** \*)  
*Constructor.*
- **ConstFieldIterator** **begin** () const
- **ConstFieldIterator** **end** () const
- const **FieldValue** **operator[]** (int iField) const  
*Return a field value.*
- **FieldValue** **operator[]** (int iField)  
*Return a field value.*
- const **FieldValue** **operator[]** (const char \*pszFieldName) const  
*Return a field value.*
- **FieldValue** **operator[]** (const char \*pszFieldName)  
*Return a field value.*
- **OGRFeatureDefn** \* **GetDefnRef** ()  
*Fetch feature definition.*
- const **OGRFeatureDefn** \* **GetDefnRef** () const  
*Fetch feature definition.*
- **OGRErr** **SetGeometryDirectly** ( **OGRGeometry** \*)  
*Set feature geometry.*
- **OGRErr** **SetGeometry** (const **OGRGeometry** \*)  
*Set feature geometry.*
- **OGRGeometry** \* **GetGeometryRef** ()  
*Fetch pointer to feature geometry.*
- const **OGRGeometry** \* **GetGeometryRef** () const  
*Fetch pointer to feature geometry.*
- **OGRGeometry** \* **StealGeometry** () **CPL\_WARN\_UNUSED\_RESULT**  
*Take away ownership of geometry.*
- int **GetGeomFieldCount** () const  
*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.*
- const **OGRGeomFieldDefn** \* **GetGeomFieldDefnRef** (int iField) const  
*Fetch definition for this geometry field.*
- int **GetGeomFieldIndex** (const char \*pszName) const  
*Fetch the geometry field index given geometry field name.*
- **OGRGeometry** \* **GetGeomFieldRef** (int iField)  
*Fetch pointer to feature geometry.*
- const **OGRGeometry** \* **GetGeomFieldRef** (int iField) const  
*Fetch pointer to feature geometry.*
- **OGRGeometry** \* **StealGeometry** (int iField)  
*Take away ownership of geometry.*
- **OGRGeometry** \* **GetGeomFieldRef** (const char \*pszFName)  
*Fetch pointer to feature geometry.*
- const **OGRGeometry** \* **GetGeomFieldRef** (const char \*pszFName) const  
*Fetch pointer to feature geometry.*
- **OGRErr** **SetGeomFieldDirectly** (int iField, **OGRGeometry** \*)  
*Set feature geometry of a specified geometry field.*
- **OGRErr** **SetGeomField** (int iField, const **OGRGeometry** \*)  
*Set feature geometry of a specified geometry field.*

- **OGRFeature \* Clone () const CPL\_WARN\_UNUSED\_RESULT**  
*Duplicate feature.*
- virtual **OGRBoolean Equal (const OGRFeature \*poFeature) const**  
*Test if two features are the same.*
- int **GetFieldCount () const**  
*Fetch number of fields on this feature. This will always be the same as the field count for the **OGRFeatureDefn** (p. ??).*
- const **OGRFieldDefn \* GetFieldDefnRef (int iField) const**  
*Fetch definition for this field.*
- **OGRFieldDefn \* GetFieldDefnRef (int iField)**  
*Fetch definition for this field.*
- int **GetFieldIndex (const char \*pszName) const**  
*Fetch the field index given field name.*
- int **IsFieldSet (int iField) const**  
*Test if a field has ever been assigned a value or not.*
- void **UnsetField (int iField)**  
*Clear a field, marking it as unset.*
- bool **IsFieldNull (int iField) const**  
*Test if a field is null.*
- void **SetFieldNull (int iField)**  
*Clear a field, marking it as null.*
- bool **IsFieldSetAndNotNull (int iField) const**  
*Test if a field is set and not null.*
- **OGRField \* GetRawFieldRef (int i)**  
*Fetch a pointer to the internal field value given the index.*
- const **OGRField \* GetRawFieldRef (int i) const**  
*Fetch a pointer to the internal field value given the index.*
- int **GetFieldAsInteger (int i) const**  
*Fetch field value as integer.*
- **GIntBig GetFieldAsInteger64 (int i) const**  
*Fetch field value as integer 64 bit.*
- double **GetFieldAsDouble (int i) const**  
*Fetch field value as a double.*
- const char \* **GetFieldAsString (int i) const**  
*Fetch field value as a string.*
- const int \* **GetFieldAsIntegerList (int i, int \*pnCount) const**  
*Fetch field value as a list of integers.*
- const **GIntBig \* GetFieldAsInteger64List (int i, int \*pnCount) const**  
*Fetch field value as a list of 64 bit integers.*
- const double \* **GetFieldAsDoubleList (int i, int \*pnCount) const**  
*Fetch field value as a list of doubles.*
- char \*\* **GetFieldAsStringList (int i) const**  
*Fetch field value as a list of strings.*
- **GByte \* GetFieldAsBinary (int i, int \*pnCount) const**  
*Fetch field value as binary data.*
- int **GetFieldAsDateTime (int i, int \*pnYear, int \*pnMonth, int \*pnDay, int \*pnHour, int \*pnMinute, int \*pnSecond, int \*pnTZFlag) const**  
*Fetch field value as date and time.*
- int **GetFieldAsDateTime (int i, int \*pnYear, int \*pnMonth, int \*pnDay, int \*pnHour, int \*pnMinute, float \*pfSecond, int \*pnTZFlag) const**  
*Fetch field value as date and time.*

- char \* **GetFieldAsSerializedJSON** (int i) const  
*Fetch field value as a serialized JSON object.*
- int **GetFieldAsInteger** (const char \*pszFName) const  
*Fetch field value as integer.*
- **GIntBig** **GetFieldAsInteger64** (const char \*pszFName) const  
*Fetch field value as integer 64 bit.*
- double **GetFieldAsDouble** (const char \*pszFName) const  
*Fetch field value as a double.*
- const char \* **GetFieldAsString** (const char \*pszFName) const  
*Fetch field value as a string.*
- const int \* **GetFieldAsIntegerList** (const char \*pszFName, int \*pnCount) const  
*Fetch field value as a list of integers.*
- const **GIntBig** \* **GetFieldAsInteger64List** (const char \*pszFName, int \*pnCount) const  
*Fetch field value as a list of 64 bit integers.*
- const double \* **GetFieldAsDoubleList** (const char \*pszFName, int \*pnCount) const  
*Fetch field value as a list of doubles.*
- char \*\* **GetFieldAsStringList** (const char \*pszFName) const  
*Fetch field value as a list of strings.*
- 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, const 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, const double \*padfValues)  
*Set field to list of doubles value.*
- void **SetField** (int i, const char \*const \*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.*
- void **SetField** (const char \*pszFName, 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.*
- void **SetField** (const char \*pszFName, **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.*
- void **SetField** (const char \*pszFName, double dfValue)  
*Set field to double value.*
- void **SetField** (const char \*pszFName, const char \*pszValue)

- Set field to string value.*

  - void **SetField** (const char \*pszFName, int nCount, const int \*panValues)

*Set field to list of integers value.*

  - void **SetField** (const char \*pszFName, int nCount, const **GIntBig** \*panValues)

*Set field to list of 64 bit integers value.*

  - void **SetField** (const char \*pszFName, int nCount, const double \*padfValues)

*Set field to list of doubles value.*

  - void **SetField** (const char \*pszFName, const char \*const \*papszValues)

*Set field to list of strings value.*

  - void **SetField** (const char \*pszFName, **OGRField** \*puValue)

*Set field.*

  - void **SetField** (const char \*pszFName, int nYear, int nMonth, int nDay, int nHour=0, int nMinute=0, float fSecond=0.f, int nTZFlag=0)

*Set field to date.*

  - **GIntBig** **GetFID** () const

*Get feature identifier.*

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

*Set the feature identifier.*

  - void **DumpReadable** (FILE \*, char \*\*papszOptions=NULLPTR) const

*Dump this feature in a human readable form.*

  - **OGRERR** **SetFrom** (const **OGRFeature** \*, int=TRUE)

*Set one feature from another.*

  - **OGRERR** **SetFrom** (const **OGRFeature** \*, const int \*, int=TRUE)

*Set one feature from another.*

  - **OGRERR** **SetFieldsFrom** (const **OGRFeature** \*, const int \*, int=TRUE)

*Set fields from another feature.*

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

*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** () const

*Fetch style string for this feature.*

  - virtual void **SetStyleString** (const char \*)

*Set feature style string.*

  - virtual void **SetStyleStringDirectly** (char \*)

*Set feature style string.*

  - virtual **OGRStyleTable** \* **GetStyleTable** () const
  - virtual void **SetStyleTable** ( **OGRStyleTable** \*poStyleTable)
  - virtual void **SetStyleTableDirectly** ( **OGRStyleTable** \*poStyleTable)
  - const char \* **GetNativeData** () const

*Returns the native data for the feature.*

  - const char \* **GetNativeMediaType** () const

*Returns the native media type for the feature.*

  - void **SetNativeData** (const char \*pszNativeData)

*Sets the native data for the feature.*

  - void **SetNativeMediaType** (const char \*pszNativeMediaType)

*Sets the native media type for the feature.*

## Static Public Member Functions

- static **OGRFeature** \* **CreateFeature** ( **OGRFeatureDefn** \*)  
*Feature factory.*
- static void **DestroyFeature** ( **OGRFeature** \*)  
*Destroy feature.*
- static **OGRFeatureH** **ToHandle** ( **OGRFeature** \*poFeature)
- static **OGRFeature** \* **FromHandle** ( **OGRFeatureH** hFeature)

## Protected Member Functions

- bool **CopySelfTo** ( **OGRFeature** \*poNew) const  
*Copies the innards of this **OGRFeature** (p. ??) into the supplied object.*

### 11.65.1 Detailed Description

A simple feature, including geometry and attributes.

### 11.65.2 Constructor & Destructor Documentation

#### 11.65.2.1 OGRFeature()

```
OGRFeature::OGRFeature (
    OGRFeatureDefn * poDefnIn ) [explicit]
```

Constructor.

Note that the **OGRFeature** (p. ??) will increment the reference count of its defining **OGRFeatureDefn** (p. ??). Destruction of the **OGRFeatureDefn** (p. ??) before destruction of all **OGRFeatures** that depend on it is likely to result in a crash.

This method is the same as the C function **OGR\_F\_Create**() (p. ??).

#### Parameters

<i>poDefnIn</i>	feature class (layer) definition to which the feature will adhere.
-----------------	--

References **OGR\_RawField\_SetUnset**(), **VSI\_CALLOC\_VERBOSE**, and **VSI\_MALLOC\_VERBOSE**.

Referenced by **CreateFeature**().

### 11.65.3 Member Function Documentation

### 11.65.3.1 begin()

```
OGRFeature::ConstFieldIterator OGRFeature::begin ( ) const
```

Return begin of field value iterator.

Using this iterator for standard range-based loops is safe, but due to implementation limitations, you shouldn't try to access (dereference) more than one iterator step at a time, since you will get a reference to the same object (**FieldValue** (p. ??)) at each iteration step.

```
for( auto&& oField: poFeature )
{
    std::cout << oField.GetIndex() << ", " << oField.GetName() << ": " << oField.GetAsString() << s
}
}
```

Since

GDAL 2.3

### 11.65.3.2 Clone()

```
OGRFeature * OGRFeature::Clone ( ) const
```

Duplicate feature.

The newly created feature is owned by the caller, and will have its 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. Or, starting with GDAL 2.1, NULL in case of out of memory situation.

References **CopySelfTo()**, and **CreateFeature()**.

### 11.65.3.3 CopySelfTo()

```
bool OGRFeature::CopySelfTo (
    OGRFeature * poNew ) const [protected]
```

Copies the innards of this **OGRFeature** (p. ??) into the supplied object.

This is mainly intended to allow derived classes to implement their own Clone functions.



## Parameters

<i>poNew</i>	The object into which to copy the data of this object.
--------------	--

## Returns

True if successful, false if the copy failed.

References OGRGeometry::clone(), GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetGeomFieldCount(), SetFID(), and VSI\_STRDUP\_VERBOSE.

Referenced by Clone().

## 11.65.3.4 CreateFeature()

```
OGRFeature * OGRFeature::CreateFeature (
    OGRFeatureDefn * poDefn ) [static]
```

Feature factory.

This is essentially a feature factory, useful for applications creating features but wanting to ensure they are created out of the OGR/GDAL heap.

This method is the same as the C function **OGR\_F\_Create()** (p. ??).

## Parameters

<i>poDefn</i>	Feature definition defining schema.
---------------	-------------------------------------

## Returns

new feature object with null fields and no geometry, or, starting with GDAL 2.1, NULL in case of out of memory situation. May be deleted with **DestroyFeature()** (p. ??).

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

Referenced by Clone(), and OGR\_F\_Create().

## 11.65.3.5 DestroyFeature()

```
void OGRFeature::DestroyFeature (
    OGRFeature * poFeature ) [static]
```

Destroy feature.

The feature is deleted, but within the context of the GDAL/OGR heap. This is necessary when higher level applications use GDAL/OGR from a DLL and they want to delete a feature created within the DLL. If the delete is done in the calling application the memory will be freed onto the application heap which is inappropriate.

This method is the same as the C function **OGR\_F\_Destroy()** (p. ??).

## Parameters

<i>poFeature</i>	the feature to delete.
------------------	------------------------

## 11.65.3.6 DumpReadable()

```
void OGRFeature::DumpReadable (
    FILE * fpOut,
    char ** papszOptions = nullptr ) const
```

Dump this feature in a human readable form.

This dumps the attributes, and geometry; however, it doesn't definition information (other than field types and names), nor does it report the geometry spatial reference system.

A few options can be defined to change the default dump :

- DISPLAY\_FIELDS=NO : to hide the dump of the attributes
- DISPLAY\_STYLE=NO : to hide the dump of the style string
- DISPLAY\_GEOMETRY=NO : to hide the dump of the geometry
- DISPLAY\_GEOMETRY=SUMMARY : to get only a summary of the geometry

This method is the same as the C function **OGR\_F\_DumpReadable()** (p. ??).

## Parameters

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

References CPL\_FRMT\_GIB, CPLsnprintf(), CPLSPrintf(), CPLTestBool(), CSLFetchNameValue(), OGR↔Geometry::dumpReadable(), EQUAL, GetFID(), GetFieldAsString(), GetFieldCount(), OGRFeatureDefn::Get↔FieldDefn(), OGRFieldDefn::GetFieldSubTypeName(), OGRFieldDefn::GetFieldType(), GetGeomField↔Count(), OGRFeatureDefn::GetGeomFieldDefn(), OGRFeatureDefn::GetName(), OGRFieldDefn::GetNameRef(), OGRGeomFieldDefn::GetNameRef(), GetStyleString(), OGRFieldDefn::GetSubType(), OGRFieldDefn::GetType(), IsFieldNull(), IsFieldSet(), and OFSTNone.

Referenced by OGR\_F\_DumpReadable().

## 11.65.3.7 end()

```
OGRFeature::ConstFieldIterator OGRFeature::end ( ) const
```

Return end of field value iterator.

References GetFieldCount().

## 11.65.3.8 Equal()

```
OGRBoolean OGRFeature::Equal (
    const OGRFeature * poFeature ) const [virtual]
```

Test if two features are the same.

Two features are considered equal if they share the same (pointer equality) same **OGRFeatureDefn** (p. ??), have the same field values, and the same geometry (as tested by **OGRGeometry::Equal()**) as well as the same feature id.

This method is the same as the C function **OGR\_F\_Equal()** (p. ??).

## Parameters

<i>poFeature</i>	the other feature to test this one against.
------------------	---

## Returns

TRUE if they are equal, otherwise FALSE.

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

## 11.65.3.9 FillUnsetWithDefault()

```
void OGRFeature::FillUnsetWithDefault (
    int bNotNullableOnly,
    char ** ppszOptions )
```

Fill unset fields with default values that might be defined.

This method is the same as the C function **OGR\_F\_FillUnsetWithDefault()** (p. ??).

## Parameters

<i>bNotNullableOnly</i>	if we should fill only unset fields with a not-null constraint.
<i>ppszOptions</i>	unused currently. Must be set to NULL.

## Since

GDAL 2.0

References **CPLES\_SQL**, **CPLFree**, **CPLUnescapeString()**, **OGRFieldDefn::GetDefault()**, **OGRFeatureDefn::GetFieldCount()**, **OGRFeatureDefn::GetFieldDefn()**, **OGRFieldDefn::GetType()**, **IsFieldSet()**, **OGRFieldDefn::IsNullable()**, **OFTDate**, **OFTDateTime**, **OFTString**, **OFTTime**, **SetField()**, and **STARTS\_WITH\_CI**.

### 11.65.3.10 FromHandle()

```
static OGRFeature* OGRFeature::FromHandle (
    OGRFeatureH hFeature ) [inline], [static]
```

Convert a OGRFeatureH to a OGRFeature\*.

Since

GDAL 2.3

Referenced by OGR\_F\_Clone(), OGR\_F\_Destroy(), OGR\_F\_DumpReadable(), OGR\_F\_Equal(), OGR\_F\_Fill↵  
UnsetWithDefault(), OGR\_F\_GetDefnRef(), OGR\_F\_GetFID(), OGR\_F\_GetFieldAsBinary(), OGR\_F\_GetField↵  
AsDateTimeEx(), OGR\_F\_GetFieldAsDouble(), OGR\_F\_GetFieldAsDoubleList(), OGR\_F\_GetFieldAsInteger(),  
OGR\_F\_GetFieldAsInteger64(), OGR\_F\_GetFieldAsInteger64List(), OGR\_F\_GetFieldAsIntegerList(), OGR↵  
\_F\_GetFieldAsString(), OGR\_F\_GetFieldAsStringList(), OGR\_F\_GetFieldCount(), OGR\_F\_GetFieldDefnRef(),  
OGR\_F\_GetFieldIndex(), OGR\_F\_GetGeometryRef(), OGR\_F\_GetGeomFieldCount(), OGR\_F\_GetGeomField↵  
DefnRef(), OGR\_F\_GetGeomFieldIndex(), OGR\_F\_GetGeomFieldRef(), OGR\_F\_GetNativeData(), OGR\_F↵  
GetNativeMediaType(), OGR\_F\_GetRawFieldRef(), OGR\_F\_GetStyleString(), OGR\_F\_GetStyleTable(), OGR↵  
\_F\_IsFieldNull(), OGR\_F\_IsFieldSet(), OGR\_F\_IsFieldSetAndNotNull(), OGR\_F\_SetFID(), OGR\_F\_SetField↵  
Binary(), OGR\_F\_SetFieldDateTime(), OGR\_F\_SetFieldDateTimeEx(), OGR\_F\_SetFieldDouble(), OGR\_F↵  
SetFieldDoubleList(), OGR\_F\_SetFieldInteger(), OGR\_F\_SetFieldInteger64(), OGR\_F\_SetFieldInteger64List(),  
OGR\_F\_SetFieldIntegerList(), OGR\_F\_SetFieldNull(), OGR\_F\_SetFieldRaw(), OGR\_F\_SetFieldString(), OG↵  
R\_F\_SetFieldStringList(), OGR\_F\_SetFrom(), OGR\_F\_SetFromWithMap(), OGR\_F\_SetGeometry(), OGR\_F↵  
\_SetGeometryDirectly(), OGR\_F\_SetGeomField(), OGR\_F\_SetGeomFieldDirectly(), OGR\_F\_SetNativeData(),  
OGR\_F\_SetNativeMediaType(), OGR\_F\_SetStyleString(), OGR\_F\_SetStyleStringDirectly(), OGR\_F\_SetStyle↵  
Table(), OGR\_F\_SetStyleTableDirectly(), OGR\_F\_StealGeometry(), OGR\_F\_UnsetField(), OGR\_F\_Validate(),  
OGR\_L\_CreateFeature(), and OGR\_L\_SetFeature().

### 11.65.3.11 GetDefnRef() [1/2]

```
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(), and SetFrom().

## 11.65.3.12 GetDefnRef() [2/2]

```
const OGRFeatureDefn * OGRFeature::GetDefnRef ( ) const [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.

## Since

GDAL 2.3

## 11.65.3.13 GetFID()

```
GIntBig OGRFeature::GetFID ( ) const [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 CopySelfTo(), DumpReadable(), Equal(), GetFieldAsDouble(), GetFieldAsString(), IsFieldSet(), and OGR\_F\_GetFID().

## 11.65.3.14 GetFieldAsBinary()

```
GByte * OGRFeature::GetFieldAsBinary (
    int iField,
    int * pnBytes ) const
```

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

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnBytes</i>	location to put the number of bytes returned.

**Returns**

the field value. This data is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSetAndNotNull(), OFTBinary, and OFTString.

Referenced by Equal().

**11.65.3.15 GetFieldAsDateTime()** [1/2]

```
int OGRFeature::GetFieldAsDateTime (
    int iField,
    int * pnYear,
    int * pnMonth,
    int * pnDay,
    int * pnHour,
    int * pnMinute,
    int * pnSecond,
    int * pnTZFlag ) const
```

Fetch field value as date and time.

Currently this method only works for OFTDate, OFTTime and OFTDateTime fields.

This method is the same as the C function **OGR\_F\_GetFieldAsDateTime()** (p. ??).

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnYear</i>	(including century)
<i>pnMonth</i>	(1-12)
<i>pnDay</i>	(1-31)
<i>pnHour</i>	(0-23)
<i>pnMinute</i>	(0-59)
<i>pnSecond</i>	(0-59)
<i>pnTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

**Returns**

TRUE on success or FALSE on failure.

Referenced by Equal().

**11.65.3.16 GetFieldAsDateTime()** [2/2]

```
int OGRFeature::GetFieldAsDateTime (
    int iField,
```

```

    int * pnYear,
    int * pnMonth,
    int * pnDay,
    int * pnHour,
    int * pnMinute,
    float * pfSecond,
    int * pnTZFlag ) const

```

Fetch field value as date and time.

Currently this method only works for OFTDate, OFTTime and OFTDateTime fields.

This method is the same as the C function **OGR\_F\_GetFieldAsDateTime()** (p. ??).

#### Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnYear</i>	(including century)
<i>pnMonth</i>	(1-12)
<i>pnDay</i>	(1-31)
<i>pnHour</i>	(0-23)
<i>pnMinute</i>	(0-59)
<i>pfSecond</i>	(0-59 with millisecond accuracy)
<i>pnTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

#### Returns

TRUE on success or FALSE on failure.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSetAndNotNull(), OFTDate, OFTDateTime, and OFTTime.

#### 11.65.3.17 GetFieldAsDouble() [1/2]

```

double OGRFeature::GetFieldAsDouble (
    int iField ) const

```

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

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

**Returns**

the field value.

References CPLAtof(), GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetGeomFieldCount(), OGRFieldDefn::GetType(), IsFieldSetAndNotNull(), OFTInteger, OFTInteger64, OFTReal, OFTString, OGR\_G\_Area(), and OGRGeometry::ToHandle().

Referenced by Equal(), OGRFeature::FieldValue::GetAsDouble(), OGR\_F\_GetFieldAsDouble(), and SetFieldsFrom().

**11.65.3.18 GetFieldAsDouble()** [2/2]

```
OGRFeature::GetFieldAsDouble (
    const char * pszFName ) const [inline]
```

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.

**Parameters**

<i>pszFName</i>	the name of the field to fetch.
-----------------	---------------------------------

**Returns**

the field value.

**11.65.3.19 GetFieldAsDoubleList()** [1/2]

```
const double * OGRFeature::GetFieldAsDoubleList (
    int iField,
    int * pnCount ) const
```

Fetch field value as a list of doubles.

Currently this method only works for OFTRealList fields.

This method is the same as the C function **OGR\_F\_GetFieldAsDoubleList()** (p. ??).

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnCount</i>	an integer to put the list count (number of doubles) into.



**Returns**

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSetAndNotNull(), and OFTReal↵List.

Referenced by Equal(), OGRFeature::FieldValue::GetAsDoubleList(), GetFieldAsSerializedJSon(), and SetFields↵From().

**11.65.3.20 GetFieldAsDoubleList()** [2/2]

```
OGRFeature::GetFieldAsDoubleList (
    const char * pszFName,
    int * pnCount ) const [inline]
```

Fetch field value as a list of doubles.

Currently this method only works for OFTRealList fields.

**Parameters**

<i>pszFName</i>	the name of the field to fetch.
<i>pnCount</i>	an integer to put the list count (number of doubles) into.

**Returns**

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

**11.65.3.21 GetFieldAsInteger()** [1/2]

```
int OGRFeature::GetFieldAsInteger (
    int iField ) const
```

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**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

**Returns**

the field value.

References `CPL_Error()`, and `OGR_FeatureDefn::GetFieldCount()`.

Referenced by `Equal()`, `OGR_Feature::FieldValue::GetAsInteger()`, `OGR_F_GetFieldAsInteger()`, and `SetFieldsFrom()`.

**11.65.3.22 GetFieldAsInteger()** [2/2]

```
OGR_Feature::GetFieldAsInteger (
    const char * pszFieldName ) const [inline]
```

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.

**Parameters**

<i>pszFieldName</i>	the name of the field to fetch.
---------------------	---------------------------------

**Returns**

the field value.

**11.65.3.23 GetFieldAsInteger64()** [1/2]

```
GIntBig OGR_Feature::GetFieldAsInteger64 (
    int iField ) const
```

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**

<i>iField</i>	the field to fetch, from 0 to <code>GetFieldCount()</code> (p. ??)-1.
---------------	---

**Returns**

the field value.

Since

GDAL 2.0

References CPLAtoGIntBigEx(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetGeomFieldCount(), OGRFieldDefn::GetType(), IsFieldSetAndNotNull(), OFTInteger, OFTInteger64, OFTReal, OFTString, OGR\_G\_Area(), and OGRGeometry::ToHandle().

Referenced by Equal(), OGRFeature::FieldValue::GetAsInteger64(), OGR\_F\_GetFieldAsInteger64(), and SetFieldsFrom().

### 11.65.3.24 GetFieldAsInteger64() [2/2]

```
OGRFeature::GetFieldAsInteger64 (
    const char * pszFName ) const [inline]
```

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.

Parameters

<i>pszFName</i>	the name of the field to fetch.
-----------------	---------------------------------

Returns

the field value.

### 11.65.3.25 GetFieldAsInteger64List() [1/2]

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

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

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnCount</i>	an integer to put the list count (number of integers) into.

**Returns**

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

**Since**

GDAL 2.0

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

Referenced by Equal(), OGRFeature::FieldValue::GetAsInteger64List(), GetFieldAsSerializedJSoN(), and SetFieldsFrom().

**11.65.3.26 GetFieldAsInteger64List()** [2/2]

```
OGRFeature::GetFieldAsInteger64List (
    const char * pszFName,
    int * pnCount ) const [inline]
```

Fetch field value as a list of 64 bit integers.

Currently this method only works for OFTInteger64List fields.

**Parameters**

<i>pszFName</i>	the name of the field to fetch.
<i>pnCount</i>	an integer to put the list count (number of integers) into.

**Returns**

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

**Since**

GDAL 2.0

**11.65.3.27 GetFieldAsIntegerList()** [1/2]

```
const int * OGRFeature::GetFieldAsIntegerList (
    int iField,
    int * pnCount ) const
```

Fetch field value as a list of integers.

Currently this method only works for OFTIntegerList fields.

This method is the same as the C function **OGR\_F\_GetFieldAsIntegerList()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnCount</i>	an integer to put the list count (number of integers) into.

## Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSetAndNotNull(), and OFTIntegerList.

Referenced by Equal(), OGRFeature::FieldValue::GetAsIntegerList(), GetFieldAsSerializedJson(), and SetFieldsFrom().

## 11.65.3.28 GetFieldAsIntegerList() [2/2]

```
OGRFeature::GetFieldAsIntegerList (
    const char * pszFName,
    int * pnCount ) const [inline]
```

Fetch field value as a list of integers.

Currently this method only works for OFTIntegerList fields.

## Parameters

<i>pszFName</i>	the name of the field to fetch.
<i>pnCount</i>	an integer to put the list count (number of integers) into.

## Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL. 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.

## 11.65.3.29 GetFieldAsSerializedJson()

```
char * OGRFeature::GetFieldAsSerializedJson (
    int iField ) const
```

Fetch field value as a serialized Json object.

Currently this method only works for OFTStringList, OFTIntegerList, OFTInteger64List and OFTRealList

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

**Returns**

a string that must be de-allocate with **CPLFree()** (p. ??)

**Since**

GDAL 2.2

References [CPLStrdup\(\)](#), [GetFieldAsDoubleList\(\)](#), [GetFieldAsInteger64List\(\)](#), [GetFieldAsIntegerList\(\)](#), [GetFieldAsStringList\(\)](#), [OGRFeatureDefn::GetFieldCount\(\)](#), [OGRFeatureDefn::GetFieldDefn\(\)](#), [OGRFieldDefn::GetType\(\)](#), [IsFieldSetAndNotNull\(\)](#), [OFTInteger64List](#), [OFTIntegerList](#), [OFTRealList](#), and [OFTStringList](#).

**11.65.3.30 GetFieldAsString()** [1/2]

```
const char * OGRFeature::GetFieldAsString (
    int iField ) const
```

Fetch field value as a string.

OFTReal and OFTInteger fields will be translated to string using `sprintf()`, but not necessarily using the established formatting rules. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR\_F\_GetFieldAsString()** (p. ??).

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

**Returns**

the field value. This string is internal, and should not be modified, or freed. Its lifetime may be very brief.

< Success

References [CPL\\_FRMT\\_GIB](#), [CPLFree](#), [CPLsprintf\(\)](#), [GetFID\(\)](#), [OGRFeatureDefn::GetFieldCount\(\)](#), [OGRFeatureDefn::GetFieldDefn\(\)](#), [OGRGeometry::getGeometryName\(\)](#), [GetGeomFieldCount\(\)](#), [OGRFieldDefn::GetPrecision\(\)](#), [GetStyleString\(\)](#), [OGRFieldDefn::GetType\(\)](#), [OGRFieldDefn::GetWidth\(\)](#), [IsFieldSetAndNotNull\(\)](#), [OFTDate](#), [OFTDateTime](#), [OFTInteger](#), [OFTInteger64](#), [OFTReal](#), [OFTString](#), [OFTTime](#), [OGR\\_G\\_Area\(\)](#), [OGR\\_G\\_ET\\_MS\(\)](#), [OGRERR\\_NONE](#), [OGRGeometry::ToHandle\(\)](#), and [VSI\\_STRDUP\\_VERBOSE](#).

Referenced by [DumpReadable\(\)](#), [Equal\(\)](#), [OGRFeature::FieldValue::GetAsString\(\)](#), [GetStyleString\(\)](#), [OGR\\_F\\_GetFieldAsString\(\)](#), and [SetFieldsFrom\(\)](#).

**11.65.3.31 GetFieldAsString()** [2/2]

```
OGRFeature::GetFieldAsString (
    const char * pszFName ) const [inline]
```

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.

**Parameters**

<i>pszFName</i>	the name of the field to fetch.
-----------------	---------------------------------

**Returns**

the field value. This string is internal, and should not be modified, or freed. Its lifetime may be very brief.

**11.65.3.32 GetFieldAsStringList()** [1/2]

```
char ** OGRFeature::GetFieldAsStringList (
    int iField ) const
```

Fetch field value as a list of strings.

Currently this method only works for OFTStringList fields.

The returned list is terminated by a NULL pointer. The number of elements can also be calculated using **CSL↔Count()** (p. ??).

This method is the same as the C function **OGR\_F\_GetFieldAsStringList()** (p. ??).

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

**Returns**

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

References `OGRFeatureDefn::GetFieldDefn()`, `OGRFieldDefn::GetType()`, `IsFieldSetAndNotNull()`, and `OFT↔StringList`.

Referenced by `Equal()`, `OGRFeature::FieldValue::GetAsStringList()`, `GetFieldAsSerializedJSoN()`, `OGR_F_Get↔FieldAsStringList()`, and `OGRFeature::FieldValue::operator CSLConstList()`.

**11.65.3.33 GetFieldAsStringList()** [2/2]

```
OGRFeature::GetFieldAsStringList (
    const char * pszFieldName ) const [inline]
```

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. ??).

**Parameters**

<i>pszFieldName</i>	the name of the field to fetch.
---------------------	---------------------------------

**Returns**

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

**11.65.3.34 GetFieldCount()**

```
int OGRFeature::GetFieldCount ( ) const [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(), end(), OGR\_F\_GetFieldCount(), OGR\_F\_IsFieldNull(), OGR\_F\_IsFieldSet(), OGR\_F\_IsFieldSetAndNotNull(), and SetFieldsFrom().

**11.65.3.35 GetFieldDefnRef()** [1/2]

```
const OGRFieldDefn * OGRFeature::GetFieldDefnRef (
    int iField ) const [inline]
```

Fetch definition for this field.

This method is the same as the C function **OGR\_F\_GetFieldDefnRef()** (p. ??).



## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

## Returns

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

## Since

GDAL 2.3

References **OGRFeatureDefn::GetFieldDefn()**.

Referenced by **OGR\_F\_GetFieldDefnRef()**, and **SetFieldsFrom()**.

## 11.65.3.36 GetFieldDefnRef() [2/2]

```
OGRFieldDefn * OGRFeature::GetFieldDefnRef (
    int iField ) [inline]
```

Fetch definition for this field.

This method is the same as the C function **OGR\_F\_GetFieldDefnRef()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

## Returns

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

References **OGRFeatureDefn::GetFieldDefn()**.

## 11.65.3.37 GetFieldIndex()

```
int OGRFeature::GetFieldIndex (
    const char * pszName ) const [inline]
```

Fetch the field index given field name.

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

This method is the same as the C function **OGR\_F\_GetFieldIndex()** (p. ??).

## Parameters

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

## Returns

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

References `OGRFeatureDefn::GetFieldIndex()`.

Referenced by `GetStyleString()`, `OGR_F_GetFieldIndex()`, and `operator[]()`.

11.65.3.38 `GetGeometryRef()` [1/2]

```
OGRGeometry * OGRFeature::GetGeometryRef ( )
```

Fetch pointer to feature geometry.

This method is essentially the same as the C function `OGR_F_GetGeometryRef()` (p. ??). (the only difference is that the C function honours `OGRGetNonLinearGeometriesEnabledFlag()` (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 `OGR_F_GetGeometryRef()`.

11.65.3.39 `GetGeometryRef()` [2/2]

```
const OGRGeometry * OGRFeature::GetGeometryRef ( ) const
```

Fetch pointer to feature geometry.

This method is essentially the same as the C function `OGR_F_GetGeometryRef()` (p. ??). (the only difference is that the C function honours `OGRGetNonLinearGeometriesEnabledFlag()` (p. ??))

## Returns

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

## Since

GDAL 2.3

References `GetGeomFieldCount()`, and `GetGeomFieldRef()`.

## 11.65.3.40 GetGeomFieldCount()

```
int OGRFeature::GetGeomFieldCount ( ) const [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()**, **GetFieldAsInteger64()**, **GetFieldAsString()**, **GetGeometryRef()**, **GetGeomFieldRef()**, **IsFieldSet()**, **OGR\_F\_GetGeomFieldCount()**, **SetFrom()**, **SetGeometry()**, **SetGeometryDirectly()**, **SetGeomField()**, **SetGeomFieldDirectly()**, and **StealGeometry()**.

## 11.65.3.41 GetGeomFieldDefnRef() [1/2]

```
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

<i>iGeomField</i>	the field to fetch, from 0 to <b>GetGeomFieldCount()</b> (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()**.

### 11.65.3.42 GetGeomFieldDefnRef() [2/2]

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

Fetch definition for this geometry field.

This method is the same as the C function **OGR\_F\_GetGeomFieldDefnRef()** (p. ??).

#### Parameters

<i>iGeomField</i>	the field to fetch, from 0 to <b>GetGeomFieldCount()</b> (p. ??)-1.
-------------------	---

#### Returns

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

#### Since

GDAL 2.3

References **OGRFeatureDefn::GetGeomFieldDefn()**.

### 11.65.3.43 GetGeomFieldIndex()

```
int OGRFeature::GetGeomFieldIndex (
    const char * pszName ) const [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

<i>pszName</i>	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()**, **OGR\_F\_GetGeomFieldIndex()**, and **SetFrom()**.

## 11.65.3.44 GetGeomFieldRef() [1/4]

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

Fetch pointer to feature geometry.

This method is the same as the C function **OGR\_F\_GetGeomFieldRef()** (p. ??).

## Parameters

<i>iField</i>	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(), OGR\_F\_GetGeomFieldRef(), SetFrom(), and Validate().

## 11.65.3.45 GetGeomFieldRef() [2/4]

```
const OGRGeometry * OGRFeature::GetGeomFieldRef (
    int iField ) const
```

Fetch pointer to feature geometry.

This method is the same as the C function **OGR\_F\_GetGeomFieldRef()** (p. ??).

## Parameters

<i>iField</i>	geometry field to get.
---------------	------------------------

## Returns

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

## Since

GDAL 2.3

References GetGeomFieldCount().

#### 11.65.3.46 GetGeomFieldRef() [ 3 / 4 ]

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

Fetch pointer to feature geometry.

##### Parameters

<i>pszFName</i>	name of geometry field to get.
-----------------	--------------------------------

##### Returns

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

##### Since

GDAL 1.11

References GetGeomFieldIndex().

#### 11.65.3.47 GetGeomFieldRef() [ 4 / 4 ]

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

Fetch pointer to feature geometry.

##### Parameters

<i>pszFName</i>	name of geometry field to get.
-----------------	--------------------------------

##### Returns

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

##### Since

GDAL 2.3

References GetGeomFieldIndex().

### 11.65.3.48 GetNativeData()

```
const char * OGRFeature::GetNativeData ( ) const [inline]
```

Returns the native data for the feature.

The native data is the representation in a "natural" form that comes from the driver that created this feature, or that is aimed at an output driver. The native data may be in different format, which is indicated by **GetNativeMediaType()** (p. ??).

Note that most drivers do not support storing the native data in the feature object, and if they do, generally the `NATIVE_DATA` open option must be passed at dataset opening.

The "native data" does not imply it is something more performant or powerful than what can be obtained with the rest of the API, but it may be useful in round-tripping scenarios where some characteristics of the underlying format are not captured otherwise by the OGR abstraction.

This function is the same as the C function **OGR\_F\_GetNativeData()** (p. ??).

#### Returns

a string with the native data, or NULL if there is none.

#### Since

GDAL 2.1

#### See also

[https://trac.osgeo.org/gdal/wiki/rfc60\\_improved\\_roundtripping\\_in\\_ogr](https://trac.osgeo.org/gdal/wiki/rfc60_improved_roundtripping_in_ogr)

Referenced by **OGR\_F\_GetNativeData()**, and **SetFrom()**.

### 11.65.3.49 GetNativeMediaType()

```
const char * OGRFeature::GetNativeMediaType ( ) const [inline]
```

Returns the native media type for the feature.

The native media type is the identifier for the format of the native data. It follows the IANA RFC 2045 (see [https://en.wikipedia.org/wiki/Media\\_type](https://en.wikipedia.org/wiki/Media_type)), e.g. "application/vnd.geo+json" for JSON.

This function is the same as the C function **OGR\_F\_GetNativeMediaType()** (p. ??).

#### Returns

a string with the native media type, or NULL if there is none.

#### Since

GDAL 2.1

#### See also

[https://trac.osgeo.org/gdal/wiki/rfc60\\_improved\\_roundtripping\\_in\\_ogr](https://trac.osgeo.org/gdal/wiki/rfc60_improved_roundtripping_in_ogr)

Referenced by **OGR\_F\_GetNativeMediaType()**, and **SetFrom()**.

**11.65.3.50 GetRawFieldRef()** [1/2]

```
OGRField * OGRFeature::GetRawFieldRef (
    int iField ) [inline]
```

Fetch a pointer to the internal field value given the index.

This method is the same as the C function **OGR\_F\_GetRawFieldRef()** (p. ??).



## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

## Returns

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

Referenced by `OGR_F_GetRawFieldRef()`, and `SetFieldsFrom()`.

## 11.65.3.51 GetRawFieldRef() [2/2]

```
const OGRField * OGRFeature::GetRawFieldRef (
    int iField ) const [inline]
```

Fetch a pointer to the internal field value given the index.

This method is the same as the C function **OGR\_F\_GetRawFieldRef()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

## Returns

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

## Since

GDAL 2.3

## 11.65.3.52 GetStyleString()

```
const char * OGRFeature::GetStyleString ( ) const [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 `DumpReadable()`, `GetFieldAsString()`, `OGRStyleMgr::InitFromFeature()`, `IsFieldSet()`, `OGR_F_↵GetStyleString()`, and `SetFrom()`.

**11.65.3.53 GetStyleTable()**

```
virtual OGRStyleTable* OGRFeature::GetStyleTable ( ) const [inline], [virtual]
```

Return style table.

**Returns**

style table.

Referenced by `OGR_F_GetStyleTable()`.

**11.65.3.54 IsFieldNull()**

```
bool OGRFeature::IsFieldNull (
    int iField ) const
```

Test if a field is null.

This method is the same as the C function **OGR\_F\_IsFieldNull()** (p. ??).

**Parameters**

<i>iField</i>	the field to test.
---------------	--------------------

**Returns**

TRUE if the field is null, otherwise false.

**Since**

GDAL 2.2

References `OGRFeatureDefn::GetFieldCount()`.

Referenced by `DumpReadable()`, `Equal()`, `SetFieldNull()`, `SetFieldsFrom()`, and `UnsetField()`.

**11.65.3.55 IsFieldSet()**

```
int OGRFeature::IsFieldSet (
    int iField ) const
```

Test if a field has ever been assigned a value or not.

This method is the same as the C function **OGR\_F\_IsFieldSet()** (p. ??).

## Parameters

<i>iField</i>	the field to test.
---------------	--------------------

## Returns

TRUE if the field has been set, otherwise false.

References `GetFID()`, `OGRFeatureDefn::GetFieldCount()`, `GetGeomFieldCount()`, `GetStyleString()`, `OGR_G↵Area()`, `OGR_RawField_IsUnset()`, `OGRNullFID`, and `OGRGeometry::ToHandle()`.

Referenced by `DumpReadable()`, `Equal()`, `FillUnsetWithDefault()`, `SetFieldNull()`, `SetFieldsFrom()`, and `UnsetField()`.

11.65.3.56 `IsFieldSetAndNotNull()`

```
bool OGRFeature::IsFieldSetAndNotNull (
    int iField ) const
```

Test if a field is set and not null.

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

## Parameters

<i>iField</i>	the field to test.
---------------	--------------------

## Returns

TRUE if the field is set and not null, otherwise false.

## Since

GDAL 2.2

References `OGRFeatureDefn::GetFieldCount()`.

Referenced by `Equal()`, `GetFieldAsBinary()`, `GetFieldAsDateTime()`, `GetFieldAsDouble()`, `GetFieldAsDoubleList()`, `GetFieldAsInteger64()`, `GetFieldAsInteger64List()`, `GetFieldAsIntegerList()`, `GetFieldAsSerializedJSon()`, `GetField↵AsString()`, `GetFieldAsStringList()`, and `SetField()`.

11.65.3.57 `operator[]()` [1/4]

```
const OGRFeature::FieldValue OGRFeature::operator[] (
    int iField ) const
```

Return a field value.

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1. This is not checked by the method !
---------------	---

## Returns

the field value.

## Since

GDAL 2.3

11.65.3.58 **operator[]()** [2/4]

```
OGRFeature::FieldValue OGRFeature::operator[] (
    int iField )
```

Return a field value.

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1. This is not checked by the method !
---------------	---

## Returns

the field value.

## Since

GDAL 2.3

11.65.3.59 **operator[]()** [3/4]

```
const OGRFeature::FieldValue OGRFeature::operator[] (
    const char * pszFieldName ) const
```

Return a field value.

## Parameters

<i>pszFieldName</i>	field name
---------------------	------------

**Returns**

the field value, or throw a **FieldNotFoundException** (p. ??) if not found.

**Since**

GDAL 2.3

References GetFieldIndex().

**11.65.3.60 operator[]()** [ 4 / 4 ]

```
OGRFeature::FieldValue OGRFeature::operator[] (
    const char * pszFieldName )
```

Return a field value.

**Parameters**

<i>pszFieldName</i>	field name
---------------------	------------

**Returns**

the field value, or throw a **FieldNotFoundException** (p. ??) if not found.

**Since**

GDAL 2.3

References GetFieldIndex().

**11.65.3.61 SetFID()**

```
OGRERR OGRFeature::SetFID (
    GIntBig nFIDIn ) [virtual]
```

Set the feature identifier.

For specific types of features this operation may fail on illegal features ids. Generally it always succeeds. Feature ids should be greater than or equal to zero, with the exception of OGRNullFID (-1) indicating that the feature id is unknown.

This method is the same as the C function **OGR\_F\_SetFID()** (p. ??).

## Parameters

<i>nFID</i>	the new feature identifier value to assign.
-------------	---

## Returns

On success OGRERR\_NONE, or on failure some other value.

< Success

References OGRERR\_NONE.

Referenced by CopySelfTo(), OGR\_F\_SetFID(), and SetFrom().

## 11.65.3.62 SetField() [1/21]

```
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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nValue</i>	the value to assign.

References CPLFree, OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSetAndNotNull(), OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, OGR\_RawField\_SetUnset(), and VSI\_STRDUP\_VERBOSE.

Referenced by FillUnsetWithDefault(), OGR\_F\_SetFieldBinary(), OGR\_F\_SetFieldDouble(), OGR\_F\_SetFieldDoubleList(), OGR\_F\_SetFieldInteger(), OGR\_F\_SetFieldInteger64(), OGR\_F\_SetFieldInteger64List(), OGR\_F\_SetFieldRaw(), OGR\_F\_SetFieldString(), OGR\_F\_SetFieldStringList(), SetField(), and SetFieldsFrom().

**11.65.3.63 SetField()** [2/21]

```
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. ??).

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nValue</i>	the value to assign.

**Since**

GDAL 2.0

References **CPLError()**, **OGRFeatureDefn::GetFieldDefn()**, **OGRFieldDefn::GetType()**, and **OFTInteger**.

**11.65.3.64 SetField()** [3/21]

```
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. ??).

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>dfValue</i>	the value to assign.

References [CPLFree](#), [CPLsnprintf\(\)](#), [OGRFeatureDefn::GetFieldDefn\(\)](#), [OGRFieldDefn::GetType\(\)](#), [IsFieldSetAndNotNull\(\)](#), [OFTInteger](#), [OFTInteger64](#), [OFTInteger64List](#), [OFTIntegerList](#), [OFTReal](#), [OFTRealList](#), [OFTString](#), [OFTStringList](#), [OGR\\_RawField\\_SetUnset\(\)](#), [SetField\(\)](#), and [VSI\\_STRDUP\\_VERBOSE](#).

11.65.3.65 **SetField()** [4/21]

```
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 **CPLAtol()** (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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pszValue</i>	the value to assign.

References [CPLError\(\)](#), [CPLFree](#), [CPLGetConfigOption\(\)](#), [CPLTestBool\(\)](#), [OGRFeatureDefn::GetFieldDefn\(\)](#), [OGRFieldDefn::GetType\(\)](#), [IsFieldSetAndNotNull\(\)](#), [OFTInteger](#), [OFTString](#), [OGR\\_RawField\\_SetUnset\(\)](#), and [VSI\\_STRDUP\\_VERBOSE](#).

11.65.3.66 **SetField()** [5/21]

```
void OGRFeature::SetField (
    int iField,
    int nCount,
    const 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. ??).



**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

References CPLFree, CPLPrintf(), CSLDestroy(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetSubType(), OGRFieldDefn::GetType(), OFSTBoolean, OFSTInt16, OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTStringList, SetField(), VSI\_MALLOC\_VERBOSE, and VSI\_STRDUP\_VERBOSE.

**11.65.3.67 SetField()** [6/21]

```
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().

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

**Since**

GDAL 2.0

References CPLError(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), and OFTIntegerList.

**11.65.3.68 SetField()** [7/21]

```
void OGRFeature::SetField (
    int iField,
    int nCount,
    const 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. ??).

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>padfValues</i>	the values to assign.

References CPLSPrintf(), CSLDestroy(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFT↔Integer, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTStringList, SetField(), VS↔I\_MALLOC\_VERBOSE, and VSI\_STRDUP\_VERBOSE.

**11.65.3.69 SetField()** [8/21]

```
void OGRFeature::SetField (
    int iField,
    const char *const * 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. ??).

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>papszValues</i>	the values to assign.

References CPLError(), CSLCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTIntegerList, OFTStringList, SetField(), and VSI\_MALLOC\_VERBOSE.

### 11.65.3.70 SetField() [9/21]

```
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. ??).

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>puValue</i>	the value to assign.

### 11.65.3.71 SetField() [10/21]

```
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. ??).

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nBytes</i>	bytes of data being set.
<i>pabyData</i>	the raw data being applied.

References [CPLFree](#), [OGRFeatureDefn::GetFieldDefn\(\)](#), [OGRFieldDefn::GetType\(\)](#), [OFTBinary](#), [OFTString](#), [OFTStringList](#), [SetField\(\)](#), and [VSI\\_MALLOC\\_VERBOSE](#).

11.65.3.72 **SetField()** [11/21]

```
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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nYear</i>	(including century)
<i>nMonth</i>	(1-12)
<i>nDay</i>	(1-31)
<i>nHour</i>	(0-23)
<i>nMinute</i>	(0-59)
<i>fSecond</i>	(0-59, with millisecond accuracy)
<i>nTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

References [CPLError\(\)](#), [OGRFeatureDefn::GetFieldDefn\(\)](#), [OGRFieldDefn::GetType\(\)](#), [OFTDate](#), [OFTDateTime](#), and [OFTTime](#).

**11.65.3.73 SetField()** [12/21]

```
OGRFeature::SetField (
    const char * pszFName,
    int nValue ) [inline]
```

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.

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>pszFName</i>	the name of the field to set.
<i>nValue</i>	the value to assign.

**11.65.3.74 SetField()** [13/21]

```
OGRFeature::SetField (
    const char * pszFName,
    GIntBig nValue ) [inline]
```

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.

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>pszFName</i>	the name of the field to set.
<i>nValue</i>	the value to assign.

**11.65.3.75 SetField()** [14/21]

```
OGRFeature::SetField (
    const char * pszFName,
    double dfValue ) [inline]
```

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.

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>pszFName</i>	the name of the field to set.
<i>dfValue</i>	the value to assign.

#### 11.65.3.76 SetField() [15/21]

```
OGRFeature::SetField (
    const char * pszFName,
    const char * pszValue ) [inline]
```

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.

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>pszFName</i>	the name of the field to set.
<i>pszValue</i>	the value to assign.

#### 11.65.3.77 SetField() [16/21]

```
OGRFeature::SetField (
    const char * pszFName,
    int nCount,
    const int * panValues ) [inline]
```

Set field to list of integers value.

This method currently on has an effect of OFTIntegerList, OFTInteger64List and OFTRealList fields.

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>pszFName</i>	the name of the field to set.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

#### 11.65.3.78 SetField() [17/21]

```
OGRFeature::SetField (
    const char * pszFName,
    int nCount,
    const GIntBig * panValues ) [inline]
```

Set field to list of 64 bit integers value.

This method currently on has an effect of OFTIntegerList, OFTInteger64List and OFTRealList fields.

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>pszFName</i>	the name of the field to set.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

#### 11.65.3.79 SetField() [18/21]

```
OGRFeature::SetField (
    const char * pszFName,
    int nCount,
    const double * padfValues ) [inline]
```

Set field to list of doubles value.

This method currently on has an effect of OFTIntegerList, OFTInteger64List, OFTRealList fields.

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>pszFName</i>	the name of the field to set.
<i>nCount</i>	the number of values in the list being assigned.
<i>padfValues</i>	the values to assign.

#### 11.65.3.80 SetField() [19/21]

```
OGRFeature::SetField (
    const char * pszFName,
    const char *const * papszValues ) [inline]
```

Set field to list of strings value.

This method currently on has an effect of OFTStringList fields.

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>pszFName</i>	the name of the field to set.
<i>papszValues</i>	the values to assign.

#### 11.65.3.81 SetField() [20/21]

```
OGRFeature::SetField (
    const char * pszFName,
    OGRField * puValue ) [inline]
```

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.



**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>pszFName</i>	the name of the field to set.
<i>puValue</i>	the value to assign.

**11.65.3.82 SetField()** [21/21]

```
OGRFeature::SetField (
    const char * pszFName,
    int nYear,
    int nMonth,
    int nDay,
    int nHour = 0,
    int nMinute = 0,
    float fSecond = 0.f,
    int nTZFlag = 0 ) [inline]
```

Set field to date.

This method currently only has an effect for OFTDate, OFTTime and OFTDateTime fields.

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>pszFName</i>	the name of the field to set.
<i>nYear</i>	(including century)
<i>nMonth</i>	(1-12)
<i>nDay</i>	(1-31)
<i>nHour</i>	(0-23)
<i>nMinute</i>	(0-59)
<i>fSecond</i>	(0-59, with millisecond accuracy)
<i>nTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

**11.65.3.83 SetFieldNull()**

```
void OGRFeature::SetFieldNull (
```

```
int iField )
```

Clear a field, marking it as null.

This method is the same as the C function **OGR\_F\_SetFieldNull()** (p. ??).

#### Parameters

<i>iField</i>	the field to set to null.
---------------	---------------------------

#### Since

GDAL 2.2

References CPLFree, CSLDestroy(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldNull(), IsFieldSet(), OFTBinary, OFTInteger64List, OFTIntegerList, OFTRealList, OFTString, OFTStringList, and OGR\_F\_RawField\_SetNull().

Referenced by OGR\_F\_SetFieldNull(), and SetFieldsFrom().

#### 11.65.3.84 SetFieldsFrom()

```
OGRERR OGRFeature::SetFieldsFrom (
    const OGRFeature * poSrcFeature,
    const int * panMap,
    int bForgiving = TRUE )
```

Set fields from another feature.

Overwrite the fields of this feature from the attributes of another. The FID and the style string are not set. The poSrcFeature does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied according to the provided indices map. Field types do not have to exactly match. **SetField()** (p. ??) method conversion rules will be applied as needed. This is more efficient than **OGR\_F\_SetFrom()** (p. ??) in that this doesn't lookup the fields by their names. Particularly useful when the field names don't match.

#### Parameters

<i>poSrcFeature</i>	the feature from which geometry, and field values will be copied.
<i>panMap</i>	Array of the indices of the feature's fields stored at the corresponding index of the source feature's fields. A value of -1 should be used to ignore the source's field. The array should not be NULL and be as long as the number of fields in the source feature.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

#### Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

< Failure

< Failure

< Failure

< Success

References `GetFieldAsDouble()`, `GetFieldAsDoubleList()`, `GetFieldAsInteger()`, `GetFieldAsInteger64()`, `GetFieldAsInteger64List()`, `GetFieldAsIntegerList()`, `GetFieldAsString()`, `GetFieldCount()`, `GetFieldDefnRef()`, `GetRawFieldRef()`, `OGRFieldDefn::GetType()`, `IsFieldNull()`, `IsFieldSet()`, `OFTDate`, `OFTDateTime`, `OFTInteger`, `OFTInteger64`, `OFTInteger64List`, `OFTIntegerList`, `OFTReal`, `OFTRealList`, `OFTString`, `OFTStringList`, `OFTTime`, `OGRERR_FAILURE`, `OGRERR_NONE`, `SetField()`, `SetFieldNull()`, and `UnsetField()`.

Referenced by `SetFrom()`.

### 11.65.3.85 SetFrom() [1/2]

```
OGRERR OGRFeature::SetFrom (
    const OGRFeature * poSrcFeature,
    int bForgiving = TRUE )
```

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The `poSrcFeature` does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied by corresponding field names. Field types do not have to exactly match. **SetField()** (p. ??) method conversion rules will be applied as needed.

This method is the same as the C function **OGR\_F\_SetFrom()** (p. ??).

#### Parameters

<i>poSrcFeature</i>	the feature from which geometry, and field values will be copied.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

#### Returns

**OGRERR\_NONE** if the operation succeeds, even if some values are not transferred, otherwise an error code.

< Failure

References `OGRFeatureDefn::ComputeMapForSetFrom()`, and `GetDefnRef()`.

### 11.65.3.86 SetFrom() [2/2]

```
OGRERR OGRFeature::SetFrom (
    const OGRFeature * poSrcFeature,
    const int * panMap,
    int bForgiving = TRUE )
```

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The `poSrcFeature` does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied according to the provided indices map. Field types do not have to exactly match. **SetField()** (p. ??) method conversion rules will be applied as needed. This is more efficient than **OGR\_F\_SetFrom()** (p. ??) in that this doesn't lookup the fields by their names. Particularly useful when the field names don't match.

This method is the same as the C function **OGR\_F\_SetFromWithMap()** (p. ??).

#### Parameters

<i>poSrcFeature</i>	the feature from which geometry, and field values will be copied.
<i>panMap</i>	Array of the indices of the feature's fields stored at the corresponding index of the source feature's fields. A value of -1 should be used to ignore the source's field. The array should not be NULL and be as long as the number of fields in the source feature.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

#### Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

< Failure

< Success

< Success

References `GetGeomFieldCount()`, `GetGeomFieldDefnRef()`, `GetGeomFieldIndex()`, `GetGeomFieldRef()`, `OGRGeomFieldDefn::GetNameRef()`, `GetNativeData()`, `GetNativeMediaType()`, `GetStyleString()`, `OGRERR_FAILURE`, `OGRERR_NONE`, `OGRNullFID`, `SetFID()`, `SetFieldsFrom()`, `SetGeomField()`, `SetNativeData()`, `SetNativeMediaType()`, and `SetStyleString()`.

#### 11.65.3.87 SetGeometry()

```
OGRErr OGRFeature::SetGeometry (
    const 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. ??).

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>po↔ GeomIn</i>	new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.
-----------------------	---

## Returns

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

< Failure

References GetGeomFieldCount(), OGRERR\_FAILURE, and SetGeomField().

## 11.65.3.88 SetGeometryDirectly()

```
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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>po↔ GeomIn</i>	new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.
-----------------------	---

## Returns

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

< Failure

References GetGeomFieldCount(), OGRERR\_FAILURE, and SetGeomFieldDirectly().

## 11.65.3.89 SetGeomField()

```
OGRERR OGRFeature::SetGeomField (
    int iField,
    const 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

<i>iField</i>	geometry field to set.
<i>poGeomIn</i>	new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.

## Returns

OGRERR\_NONE if successful, or 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

< Failure  
< Success

References OGRGeometry::clone(), GetGeomFieldCount(), OGRERR\_FAILURE, and OGRERR\_NONE.

Referenced by SetFrom(), and SetGeometry().

## 11.65.3.90 SetGeomFieldDirectly()

```
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

<i>iField</i>	geometry field to set.
<i>poGeomIn</i>	new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.

## Returns

OGRERR\_NONE if successful, or OGRERR\_FAILURE if the index is invalid, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

## Since

GDAL 1.11

< Failure  
< Success

References `GetGeomFieldCount()`, `OGRERR_FAILURE`, and `OGRERR_NONE`.

Referenced by `OGR_F_GetGeometryRef()`, `OGR_F_GetGeomFieldRef()`, and `SetGeometryDirectly()`.

## 11.65.3.91 SetNativeData()

```
void OGRFeature::SetNativeData (
    const char * pszNativeData )
```

Sets the native data for the feature.

The native data is the representation in a "natural" form that comes from the driver that created this feature, or that is aimed at an output driver. The native data may be in different format, which is indicated by **GetNativeMediaType()** (p. ??).

This function is the same as the C function **OGR\_F\_SetNativeData()** (p. ??).

## Parameters

<i>pszNativeData</i>	a string with the native data, or NULL if there is none.
----------------------	--

## Since

GDAL 2.1

## See also

[https://trac.osgeo.org/gdal/wiki/rfc60\\_improved\\_roundtripping\\_in\\_ogr](https://trac.osgeo.org/gdal/wiki/rfc60_improved_roundtripping_in_ogr)

References CPLFree, and VSI\_STRDUP\_VERBOSE.

Referenced by OGR\_F\_SetNativeData(), and SetFrom().

### 11.65.3.92 SetNativeMediaType()

```
void OGRFeature::SetNativeMediaType (
    const char * pszNativeMediaType )
```

Sets the native media type for the feature.

The native media type is the identifier for the format of the native data. It follows the IANA RFC 2045 (see [https://en.wikipedia.org/wiki/Media\\_type](https://en.wikipedia.org/wiki/Media_type)), e.g. "application/vnd.geo+json" for JSon.

This function is the same as the C function **OGR\_F\_SetNativeMediaType()** (p. ??).

#### Parameters

<i>pszNativeMediaType</i>	a string with the native media type, or NULL if there is none.
---------------------------	--

#### Since

GDAL 2.1

#### See also

[https://trac.osgeo.org/gdal/wiki/rfc60\\_improved\\_roundtripping\\_in\\_ogr](https://trac.osgeo.org/gdal/wiki/rfc60_improved_roundtripping_in_ogr)

References CPLFree, and VSI\_STRDUP\_VERBOSE.

Referenced by SetFrom().

### 11.65.3.93 SetStyleString()

```
void OGRFeature::SetStyleString (
    const char * pszString ) [virtual]
```

Set feature style string.

This method operate exactly as **OGRFeature::SetStyleStringDirectly()** (p. ??) except that it does not assume ownership of the passed string, but instead makes a copy of it.

This method is the same as the C function **OGR\_F\_SetStyleString()** (p. ??).

#### Parameters

<i>pszString</i>	the style string to apply to this feature, cannot be NULL.
------------------	--



References CPLFree, and VSI\_STRDUP\_VERBOSE.

Referenced by OGR\_F\_SetStyleString(), OGRStyleMgr::SetFeatureStyleString(), and SetFrom().

### 11.65.3.94 SetStyleStringDirectly()

```
void OGRFeature::SetStyleStringDirectly (
    char * pszString ) [virtual]
```

Set feature style string.

This method operate exactly as **OGRFeature::SetStyleString()** (p. ??) except that it assumes ownership of the passed string.

This method is the same as the C function **OGR\_F\_SetStyleStringDirectly()** (p. ??).

#### Parameters

<i>pszString</i>	the style string to apply to this feature, cannot be NULL.
------------------	--

References CPLFree.

Referenced by OGR\_F\_SetStyleStringDirectly().

### 11.65.3.95 SetStyleTable()

```
void OGRFeature::SetStyleTable (
    OGRStyleTable * poStyleTable ) [virtual]
```

Set style table.

#### Parameters

<i>poStyleTable</i>	new style table (will be cloned)
---------------------	----------------------------------

References OGRStyleTable::Clone().

### 11.65.3.96 SetStyleTableDirectly()

```
void OGRFeature::SetStyleTableDirectly (
    OGRStyleTable * poStyleTable ) [virtual]
```

Set style table.

## Parameters

<i>poStyleTable</i>	new style table (ownership transferred to the object)
---------------------	---

## 11.65.3.97 StealGeometry() [1/2]

```
OGRGeometry * OGRFeature::StealGeometry ( )
```

Take away ownership of geometry.

Fetch the geometry from this feature, and clear the reference to the geometry on the feature. This is a mechanism for the application to take over ownership of the geometry from the feature without copying. Sort of an inverse to **SetGeometryDirectly()** (p. ??).

After this call the **OGRFeature** (p. ??) will have a NULL geometry.

## Returns

the pointer to the geometry.

References GetGeomFieldCount().

Referenced by OGR\_F\_GetGeometryRef(), and OGR\_F\_GetGeomFieldRef().

## 11.65.3.98 StealGeometry() [2/2]

```
OGRGeometry * OGRFeature::StealGeometry (
    int iGeomField )
```

Take away ownership of geometry.

Fetch the geometry from this feature, and clear the reference to the geometry on the feature. This is a mechanism for the application to take over ownership of the geometry from the feature without copying. Sort of an inverse to **SetGeometryDirectly()** (p. ??).

After this call the **OGRFeature** (p. ??) will have a NULL geometry.

## Parameters

<i>iGeomField</i>	index of the geometry field.
-------------------	------------------------------

## Returns

the pointer to the geometry.

References GetGeomFieldCount().

## 11.65.3.99 ToHandle()

```
static OGRFeatureH OGRFeature::ToHandle (
    OGRFeature * poFeature ) [inline], [static]
```

Convert a OGRFeature\* to a OGRFeatureH.

Since

GDAL 2.3

Referenced by OGR\_F\_Clone(), OGR\_F\_Create(), OGR\_L\_GetFeature(), and OGR\_L\_GetNextFeature().

## 11.65.3.100 UnsetField()

```
void OGRFeature::UnsetField (
    int iField )
```

Clear a field, marking it as unset.

This method is the same as the C function **OGR\_F\_UnsetField()** (p. ??).

Parameters

<i>iField</i>	the field to unset.
---------------	---------------------

References CPLFree, CSLDestroy(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldNull(), IsFieldSet(), OFTBinary, OFTInteger64List, OFTIntegerList, OFTRealList, OFTString, OFTStringList, and OGR\_F\_RawField\_SetUnset().

Referenced by OGR\_F\_UnsetField(), and SetFieldsFrom().

## 11.65.3.101 Validate()

```
int OGRFeature::Validate (
    int nValidateFlags,
    int bEmitError ) const
```

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

<i>nValidateFlags</i>	OGR_F_VAL_ALL or combination of OGR_F_VAL_NULL, OGR_F_VAL_GEOM_TYPE, OGR_F_VAL_WIDTH and OGR_F_VAL_ALLOW_NULL_WHEN_DEFAULT, OGR_F_VAL_ALLOW_DIFFERENT_GEOM_DIM with ' ' operator
<i>bEmitError</i>	TRUE if a <b>CPL</b> Error() (p. ??) must be emitted when a check fails

## Returns

TRUE if all enabled validation tests pass.

## Since

GDAL 2.0

References CPLError(), OGRFeatureDefn::GetGeomFieldCount(), OGRFeatureDefn::GetGeomFieldDefn(), GetGeomFieldRef(), OGRGeomFieldDefn::IsNullable(), and OGR\_F\_VAL\_NULL.

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

- **ogr\_feature.h**
- ogrfeature.cpp

## 11.66 OGRFeatureDefn Class Reference

```
#include <ogr_feature.h>
```

### Public Member Functions

- **OGRFeatureDefn** (const char \*pszName=nullptr)  
*Constructor.*
- void **SetName** (const char \*pszName)  
*Change name of this OGRFeatureDefn (p. ??).*
- virtual const char \* **GetName** () const  
*Get name of this OGRFeatureDefn (p. ??).*
- virtual int **GetFieldCount** () const  
*Fetch number of fields on this feature.*
- virtual **OGRFieldDefn** \* **GetFieldDefn** (int i)  
*Fetch field definition.*
- virtual const **OGRFieldDefn** \* **GetFieldDefn** (int i) const  
*Fetch field definition.*
- virtual int **GetFieldIndex** (const char \*) const  
*Find field by name.*
- virtual void **AddFieldDefn** ( **OGRFieldDefn** \*)  
*Add a new field definition.*
- virtual **OGR**Err **DeleteFieldDefn** (int iField)  
*Delete an existing field definition.*
- virtual **OGR**Err **ReorderFieldDefns** (int \*panMap)

- Reorder the field definitions in the array of the feature definition.*
- virtual int **GetGeomFieldCount** () const  
*Fetch number of geometry fields on this feature.*
- virtual **OGRGeomFieldDefn \* GetGeomFieldDefn** (int i)  
*Fetch geometry field definition.*
- virtual const **OGRGeomFieldDefn \* GetGeomFieldDefn** (int i) const  
*Fetch geometry field definition.*
- virtual int **GetGeomFieldIndex** (const char \*) const  
*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** () const  
*Fetch the geometry base type.*
- virtual void **SetGeomType** ( **OGRwkbGeometryType**)  
*Assign the base geometry type for this layer.*
- virtual **OGRFeatureDefn \* Clone** () const  
*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** () const  
*Fetch current reference count.*
- void **Release** ()  
*Drop a reference to this object, and destroy if no longer referenced.*
- virtual int **IsGeometryIgnored** () const  
*Determine whether the geometry can be omitted when fetching features.*
- virtual void **SetGeometryIgnored** (int blgnore)  
*Set whether the geometry can be omitted when fetching features.*
- virtual int **IsStyleIgnored** () const  
*Determine whether the style can be omitted when fetching features.*
- virtual void **SetStyleIgnored** (int blgnore)  
*Set whether the style can be omitted when fetching features.*
- virtual int **IsSame** (const **OGRFeatureDefn \*poOtherFeatureDefn**) const  
*Test if the feature definition is identical to the other one.*
- std::vector< int > **ComputeMapForSetFrom** (const **OGRFeatureDefn \*poSrcFDefn**, bool bForgiving=true)  
const  
*Compute the map from source to target field that can be passed to SetFrom().*

## Static Public Member Functions

- static **OGRFeatureDefn \* CreateFeatureDefn** (const char \*pszName=nullptr)
- static void **DestroyFeatureDefn** ( **OGRFeatureDefn \***)
- static **OGRFeatureDefnH ToHandle** ( **OGRFeatureDefn \*poFeatureDefn**)
- static **OGRFeatureDefn \* FromHandle** ( **OGRFeatureDefnH hFeatureDefn**)

### 11.66.1 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.

### 11.66.2 Constructor & Destructor Documentation

#### 11.66.2.1 OGRFeatureDefn()

```
OGRFeatureDefn::OGRFeatureDefn (
    const char * pszName = nullptr ) [explicit]
```

Constructor.

The **OGRFeatureDefn** (p. ??) maintains a reference count, but this starts at zero. It is mainly intended to represent a count of **OGRFeature** (p. ??)'s based on this definition.

This method is the same as the C function **OGR\_FD\_Create()** (p. ??).

#### Parameters

<i>pszName</i>	the name to be assigned to this layer/class. It does not need to be unique.
----------------	---

References CPLMalloc(), CPLStrdup(), and wkbUnknown.

Referenced by Clone(), and CreateFeatureDefn().

### 11.66.3 Member Function Documentation

#### 11.66.3.1 AddFieldDefn()

```
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 existence based on this **OGRFeatureDefn** (p. ??). The **OGRFieldDefn** (p. ??) passed in is copied, and remains the responsibility of the caller.

This method is the same as the C function **OGR\_FD\_AddFieldDefn()** (p. ??).

#### Parameters

<i>poNewDefn</i>	the definition of the new field.
------------------	----------------------------------

References CPLRealloc(), and GetFieldCount().

Referenced by Clone().

#### 11.66.3.2 AddGeomFieldDefn()

```
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 existence 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

<i>poNewDefn</i>	the definition of the new geometry field.
<i>bCopy</i>	whether poNewDefn should be copied.

Since

GDAL 1.11

References CPLRealloc(), and GetGeomFieldCount().

Referenced by Clone(), OGR\_FD\_AddGeomFieldDefn(), and SetGeomType().

### 11.66.3.3 Clone()

```
OGRFeatureDefn * OGRFeatureDefn::Clone ( ) const [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().

### 11.66.3.4 ComputeMapForSetFrom()

```
std::vector< int > OGRFeatureDefn::ComputeMapForSetFrom (
    const OGRFeatureDefn * poSrcFDefn,
    bool bForgiving = true ) const
```

Compute the map from source to target field that can be passed to SetFrom().

#### Parameters

<i>poSrcFDefn</i>	the feature definition of source features later passed to SetFrom()
<i>bForgiving</i>	true if the operation should continue despite lacking output fields matching some of the source fields.

#### Returns

an array of size poSrcFDefn->**GetFieldCount()** (p. ??) if everything succeeds, or empty in case a source field definition was not found in the target layer and bForgiving == true.

#### Since

GDAL 2.3

References GetFieldCount(), GetFieldDefn(), OGRFieldDefn::GetNameRef(), and CPLString::toupper().

Referenced by OGRFeature::SetFrom().

### 11.66.3.5 CreateFeatureDefn()

```
OGRFeatureDefn * OGRFeatureDefn::CreateFeatureDefn (
    const char * pszName = nullptr ) [static]
```

Create a new feature definition object.



## Parameters

<i>pszName</i>	name
----------------	------

## Returns

new feature definition object.

References OGRFeatureDefn().

## 11.66.3.6 DeleteFieldDefn()

```
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 existence based on this **OGRFeatureDefn** (p. ??).

This method is the same as the C function **OGR\_FD\_DeleteFieldDefn()** (p. ??).

## Parameters

<i>iField</i>	the index of the field definition.
---------------	------------------------------------

## Returns

OGRERR\_NONE in case of success.

## Since

OGR 1.9.0

< Failure  
< Success

References GetFieldCount(), OGRERR\_FAILURE, and OGRERR\_NONE.

Referenced by OGR\_FD\_DeleteFieldDefn().

### 11.66.3.7 DeleteGeomFieldDefn()

```
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 existence based on this **OGRFeatureDefn** (p. ??).

This method is the same as the C function **OGR\_FD\_DeleteGeomFieldDefn()** (p. ??).

#### Parameters

<i>iGeomField</i>	the index of the geometry field definition.
-------------------	---

#### Returns

`OGRERR_NONE` in case of success.

#### Since

GDAL 1.11

< Failure

< Success

References `GetGeomFieldCount()`, `OGRERR_FAILURE`, and `OGRERR_NONE`.

Referenced by `Clone()`, and `SetGeomType()`.

### 11.66.3.8 Dereference()

```
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 `OGR_FD_Dereference()`, and `Release()`.

### 11.66.3.9 DestroyFeatureDefn()

```
void OGRFeatureDefn::DestroyFeatureDefn (
    OGRFeatureDefn * poDefn ) [static]
```

Destroy a feature definition.

## Parameters

<i>poDefn</i>	feature definition.
---------------	---------------------

## 11.66.3.10 FromHandle()

```
static OGRFeatureDefn* OGRFeatureDefn::FromHandle (
    OGRFeatureDefnH hFeatureDefn ) [inline], [static]
```

Convert a OGRFeatureDefnH to a OGRFeatureDefn\*.

## Since

GDAL 2.3

Referenced by OGR\_F\_Create(), OGR\_FD\_AddFieldDefn(), OGR\_FD\_AddGeomFieldDefn(), OGR\_FD\_DeleteFieldDefn(), OGR\_FD\_DeleteGeomFieldDefn(), OGR\_FD\_Dereference(), OGR\_FD\_Destroy(), OGR\_FD\_GetFieldCount(), OGR\_FD\_GetFieldDefn(), OGR\_FD\_GetFieldIndex(), OGR\_FD\_GetGeomFieldCount(), OGR\_FD\_GetGeomFieldDefn(), OGR\_FD\_GetGeomFieldIndex(), OGR\_FD\_GetGeomType(), OGR\_FD\_GetName(), OGR\_FD\_GetReferenceCount(), OGR\_FD\_IsGeometryIgnored(), OGR\_FD\_IsSame(), OGR\_FD\_IsStyleIgnored(), OGR\_FD\_Reference(), OGR\_FD\_Release(), OGR\_FD\_ReorderFieldDefns(), OGR\_FD\_SetGeometryIgnored(), OGR\_FD\_SetGeomType(), and OGR\_FD\_SetStyleIgnored().

## 11.66.3.11 GetFieldCount()

```
int OGRFeatureDefn::GetFieldCount ( ) const [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(), ComputeMapForSetFrom(), OGRFeature::CopySelfTo(), OGRFeature::CreateFeature(), DeleteFieldDefn(), OGRFeature::Equal(), OGRFeature::FillUnsetWithDefault(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsInteger64(), OGRFeature::GetFieldAsSerializedJSON(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldCount(), GetFieldDefn(), GetFieldIndex(), OGRFeature::IsFieldNull(), OGRFeature::IsFieldSet(), OGRFeature::IsFieldSetAndNotNull(), IsSame(), OGR\_FD\_GetFieldCount(), OGRLayer::ReorderField(), ReorderFieldDefns(), and OGRLayer::SetIgnoredFields().

## 11.66.3.12 GetFieldDefn() [1/2]

```
OGRFieldDefn * OGRFeatureDefn::GetFieldDefn (
    int iField ) [virtual]
```

Fetch field definition.

This method is the same as the C function **OGR\_FD\_GetFieldDefn()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, between 0 and <b>GetFieldCount()</b> (p. ??) - 1.
---------------	---

## Returns

a pointer to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

References CPLError(), and GetFieldCount().

Referenced by Clone(), ComputeMapForSetFrom(), OGRFeature::DumpReadable(), OGRFeature::Equal(), OGRFeature::FillUnsetWithDefault(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger64(), OGRFeature::GetFieldAsInteger64List(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsSerializedJson(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRFeature::GetFieldDefnRef(), GetFieldIndex(), IsSame(), OGRFeature::SetField(), OGRFeature::SetFieldNull(), OGRLayer::SetIgnoredFields(), and OGRFeature::UnsetField().

## 11.66.3.13 GetFieldDefn() [2/2]

```
const OGRFieldDefn * OGRFeatureDefn::GetFieldDefn (
    int iField ) const [virtual]
```

Fetch field definition.

This method is the same as the C function **OGR\_FD\_GetFieldDefn()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, between 0 and <b>GetFieldCount()</b> (p. ??) - 1.
---------------	---

## Returns

a pointer to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

## Since

GDAL 2.3

References CPLError(), and GetFieldCount().

## 11.66.3.14 GetFieldIndex()

```
int OGRFeatureDefn::GetFieldIndex (
    const char * pszFieldName ) const [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

<i>pszFieldName</i>	the field name to search for.
---------------------	-------------------------------

## Returns

the field index, or -1 if no match found.

References EQUAL, GetFieldCount(), GetFieldDefn(), and OGRFieldDefn::GetNameRef().

Referenced by OGRLayer::FindFieldIndex(), OGRFeature::GetFieldIndex(), OGR\_FD\_GetFieldIndex(), and OGR↔  
RLayer::SetIgnoredFields().

## 11.66.3.15 GetGeomFieldCount()

```
int OGRFeatureDefn::GetGeomFieldCount ( ) const [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::CopySelfTo(), OGRFeature::CreateFeature(), Delete↔  
GeomFieldDefn(), OGRFeature::GetGeomFieldCount(), GetGeomFieldDefn(), GetGeomFieldIndex(), GetGeom↔  
Type(), IsGeometryIgnored(), IsSame(), OGR\_FD\_GetGeomFieldCount(), SetGeometryIgnored(), SetGeomType(),  
OGRLayer::SetIgnoredFields(), and OGRFeature::Validate().

## 11.66.3.16 GetGeomFieldDefn() [1/2]

```
OGRGeomFieldDefn * OGRFeatureDefn::GetGeomFieldDefn (
    int iGeomField ) [virtual]
```

Fetch geometry field definition.

This method is the same as the C function **OGR\_FD\_GetGeomFieldDefn()** (p. ??).

**Parameters**

<i>iGeomField</i>	the geometry field to fetch, between 0 and <b>GetGeomFieldCount()</b> (p. ??) - 1.
-------------------	--

**Returns**

a pointer to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

**Since**

GDAL 1.11

References CPLError(), and GetGeomFieldCount().

Referenced by Clone(), OGRFeature::DumpReadable(), OGRLayer::GetGeometryColumn(), OGRFeature::GetGeomFieldDefnRef(), GetGeomFieldIndex(), GetGeomType(), OGRLayer::GetSpatialRef(), IsGeometryIgnored(), IsSame(), SetGeometryIgnored(), SetGeomType(), OGRLayer::SetIgnoredFields(), and OGRFeature::Validate().

**11.66.3.17 GetGeomFieldDefn()** [2/2]

```
const OGRGeomFieldDefn * OGRFeatureDefn::GetGeomFieldDefn (
    int iGeomField ) const [virtual]
```

Fetch geometry field definition.

This method is the same as the C function **OGR\_FD\_GetGeomFieldDefn()** (p. ??).

**Parameters**

<i>iGeomField</i>	the geometry field to fetch, between 0 and <b>GetGeomFieldCount()</b> (p. ??) - 1.
-------------------	--

**Returns**

a pointer to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

**Since**

GDAL 2.3

References CPLError(), and GetGeomFieldCount().

## 11.66.3.18 GetGeomFieldIndex()

```
int OGRFeatureDefn::GetGeomFieldIndex (
    const char * pszGeomFieldName ) const [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

<i>pszGeomFieldName</i>	the geometry field name to search for.
-------------------------	--

## Returns

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

References EQUAL, GetGeomFieldCount(), GetGeomFieldDefn(), and OGRGeomFieldDefn::GetNameRef().

Referenced by OGRFeature::GetGeomFieldIndex(), and OGRLayer::SetIgnoredFields().

## 11.66.3.19 GetGeomType()

```
OGRwkbGeometryType OGRFeatureDefn::GetGeomType ( ) const [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(), CPLTestBool(), GetGeomFieldCount(), GetGeomFieldDefn(), OGRGeomFieldDefn::GetType(), wkbNone, and wkbUnknown.

Referenced by OGRLayer::GetGeomType(), and OGR\_FD\_GetGeomType().

### 11.66.3.20 GetName()

```
const char * OGRFeatureDefn::GetName ( ) const [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(), and OGR\_FD\_GetName().

### 11.66.3.21 GetReferenceCount()

```
int OGRFeatureDefn::GetReferenceCount ( ) const [inline]
```

Fetch current reference count.

This method is the same as the C function **OGR\_FD\_GetReferenceCount()** (p. ??).

#### Returns

the current reference count.

Referenced by OGR\_FD\_GetReferenceCount().

### 11.66.3.22 IsGeometryIgnored()

```
int OGRFeatureDefn::IsGeometryIgnored ( ) const [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().

Referenced by OGR\_FD\_IsGeometryIgnored().

### 11.66.3.23 IsSame()

```
int OGRFeatureDefn::IsSame (
    const OGRFeatureDefn * poOtherFeatureDefn ) const [virtual]
```

Test if the feature definition is identical to the other one.



## Parameters

<i>poOtherFeatureDefn</i>	the other feature definition to compare to.
---------------------------	---

## Returns

TRUE if the feature definition is identical to the other one.

References `GetFieldCount()`, `GetFieldDefn()`, `GetGeomFieldCount()`, `GetGeomFieldDefn()`, `GetName()`, `OGRFeatureDefn::IsSame()`, and `OGRGeomFieldDefn::IsSame()`.

11.66.3.24 `IsStyleIgnored()`

```
int OGRFeatureDefn::IsStyleIgnored ( ) const [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

Referenced by `OGR_FD_IsStyleIgnored()`.

11.66.3.25 `Reference()`

```
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 `OGR_FD_Reference()`.

11.66.3.26 `ReorderFieldDefns()`

```
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 existence based on this **OGRFeatureDefn** (p. ??).

This method is the same as the C function **OGR\_FD\_ReorderFieldDefns()** (p. ??).

## Parameters

<i>panMap</i>	an array of <b>GetFieldCount()</b> (p. ??) elements which is a permutation of [0, <b>GetFieldCount()</b> (p. ??)-1]. <i>panMap</i> is such that, for each field definition at position <i>i</i> after reordering, its position before reordering was <i>panMap</i> [ <i>i</i> ].
---------------	--

## Returns

OGRERR\_NONE in case of success.

## Since

OGR 1.9.0

< Success

< Success

< Success

References CPLFree, CPLMalloc(), GetFieldCount(), and OGRERR\_NONE.

Referenced by OGR\_FD\_ReorderFieldDefns().

## 11.66.3.27 SetGeometryIgnored()

```
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

<i>bIgnore</i>	ignore state
----------------	--------------

References GetGeomFieldCount(), GetGeomFieldDefn(), and OGRGeomFieldDefn::SetIgnored().

Referenced by OGR\_FD\_SetGeometryIgnored(), and OGRLayer::SetIgnoredFields().

## 11.66.3.28 SetGeomType()

```
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

<i>eNewType</i>	the new type to assign.
-----------------	-------------------------

References `AddGeomFieldDefn()`, `DeleteGeomFieldDefn()`, `GetGeomFieldCount()`, `GetGeomFieldDefn()`, `OGRGeomFieldDefn::SetType()`, and `wkbNone`.

Referenced by `OGR_FD_SetGeomType()`.

#### 11.66.3.29 SetName()

```
void OGRFeatureDefn::SetName (
    const char * pszName )
```

Change name of this **OGRFeatureDefn** (p. ??).

#### Parameters

<i>pszName</i>	feature definition name
----------------	-------------------------

#### Since

GDAL 2.3

References `CPLFree`, and `CPLStrdup()`.

#### 11.66.3.30 SetStyleIgnored()

```
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

<i>blgnore</i>	ignore state
----------------	--------------

Referenced by `OGR_FD_SetStyleIgnored()`, and `OGRLayer::SetIgnoredFields()`.

## 11.66.3.31 ToHandle()

```
static OGRFeatureDefnH OGRFeatureDefn::ToHandle (
    OGRFeatureDefn * poFeatureDefn ) [inline], [static]
```

Convert a `OGRFeatureDefn*` to a `OGRFeatureDefnH`.

## Since

GDAL 2.3

Referenced by `OGR_F_GetDefnRef()`, `OGR_FD_Create()`, and `OGR_L_GetLayerDefn()`.

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

- `ogr_feature.h`
- `ogrfeaturedefn.cpp`

## 11.67 OGRField Union Reference

```
#include <ogr_core.h>
```

## 11.67.1 Detailed Description

**OGRFeature** (p. ??) field attribute value union.

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

- `ogr_core.h`

## 11.68 OGRFieldDefn Class Reference

```
#include <ogr_feature.h>
```

## Public Member Functions

- **OGRFieldDefn** (const char \*, **OGRFieldType**)  
*Constructor.*
- **OGRFieldDefn** (const **OGRFieldDefn** \*)  
*Constructor.*
- void **SetName** (const char \*)  
*Reset the name of this field.*
- const char \* **GetNameRef** () const  
*Fetch name of this field.*
- **OGRFieldType** **GetType** () const  
*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** () const  
*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** () const  
*Get the justification for this field.*
- void **SetJustify** ( **OGRJustification** eJustifyIn)  
*Set the justification for this field.*
- int **GetWidth** () const  
*Get the formatting width for this field.*
- void **SetWidth** (int nWidthIn)  
*Set the formatting width for this field in characters.*
- int **GetPrecision** () const  
*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** () const  
*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.*
- static OGRFieldDefnH **ToHandle** ( OGRFieldDefn \*poFieldDefn)
- static OGRFieldDefn \* **FromHandle** ( OGRFieldDefnH hFieldDefn)

### 11.68.1 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. ??)

### 11.68.2 Constructor & Destructor Documentation

#### 11.68.2.1 OGRFieldDefn() [1/2]

```
OGRFieldDefn::OGRFieldDefn (
    const char * pszNameIn,
    OGRFieldType eTypeIn )
```

Constructor.

By default, fields have no width, precision, are nullable and not ignored.

#### Parameters

<i>pszNameIn</i>	the name of the new field.
<i>eTypeIn</i>	the type of the new field.

## 11.68.2.2 OGRFieldDefn() [2/2]

```
OGRFieldDefn::OGRFieldDefn (
    const OGRFieldDefn * poPrototype ) [explicit]
```

Constructor.

Create by cloning an existing field definition.

## Parameters

<i>poPrototype</i>	the field definition to clone.
--------------------	--------------------------------

References GetDefault(), and SetDefault().

## 11.68.3 Member Function Documentation

## 11.68.3.1 FromHandle()

```
static OGRFieldDefn* OGRFieldDefn::FromHandle (
    OGRFieldDefnH hFieldDefn ) [inline], [static]
```

Convert a OGRFieldDefnH to a OGRFieldDefn\*.

Since

GDAL 2.3

Referenced by OGR\_FD\_AddFieldDefn(), OGR\_Fld\_Destroy(), OGR\_Fld\_GetDefault(), OGR\_Fld\_GetJustify(), OGR\_Fld\_GetNameRef(), OGR\_Fld\_GetPrecision(), OGR\_Fld\_GetSubType(), OGR\_Fld\_GetType(), OGR\_Fld\_GetWidth(), OGR\_Fld\_IsDefaultDriverSpecific(), OGR\_Fld\_IsIgnored(), OGR\_Fld\_IsNullable(), OGR\_Fld\_Set(), OGR\_Fld\_SetDefault(), OGR\_Fld\_SetIgnored(), OGR\_Fld\_SetJustify(), OGR\_Fld\_SetName(), OGR\_Fld\_SetNullable(), OGR\_Fld\_SetPrecision(), OGR\_Fld\_SetSubType(), OGR\_Fld\_SetType(), OGR\_Fld\_SetWidth(), OGR\_L\_AlterFieldDefn(), and OGR\_L\_CreateField().

## 11.68.3.2 GetDefault()

```
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 OGRFeature::FillUnsetWithDefault(), OGR\_Fld\_GetDefault(), and OGRFieldDefn().

### 11.68.3.3 GetFieldSubTypeName()

```
const char * OGRFieldDefn::GetFieldSubTypeName (
    OGRFieldType eSubType ) [static]
```

Fetch human readable name for a field subtype.

This static method is the same as the C function **OGR\_GetFieldSubTypeName()** (p. ??).

#### Parameters

<i>eSubType</i>	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().

### 11.68.3.4 GetFieldTypeNames()

```
const char * OGRFieldDefn::GetFieldTypeNames (
    OGRFieldType eType ) [static]
```

Fetch human readable name for a field type.

This static method is the same as the C function **OGR\_GetFieldTypeNames()** (p. ??).

#### Parameters

<i>eType</i>	the field type to get name for.
--------------	---------------------------------

#### Returns

pointer to an internal static name string. It should not be modified or freed.

References OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTInteger64, OFTInteger64List, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

Referenced by OGRFeature::DumpReadable(), and OGR\_GetFieldTypeNames().



### 11.68.3.5 GetJustify()

```
OGRJustification OGRFieldDefn::GetJustify ( ) const [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 **OGR\_Fld\_GetJustify()**.

### 11.68.3.6 GetNameRef()

```
const char * OGRFieldDefn::GetNameRef ( ) const [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 **OGRFeatureDefn::ComputeMapForSetFrom()**, **OGRFeature::DumpReadable()**, **OGRFeatureDefn::GetFieldIndex()**, and **OGR\_Fld\_GetNameRef()**.

### 11.68.3.7 GetPrecision()

```
int OGRFieldDefn::GetPrecision ( ) const [inline]
```

Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.

This method is the same as the C function **OGR\_Fld\_GetPrecision()** (p. ??).

#### Returns

the precision.

Referenced by **OGRFeature::GetFieldAsString()**, and **OGR\_Fld\_GetPrecision()**.

### 11.68.3.8 GetSubType()

```
OGRFieldType OGRFieldDefn::GetSubType ( ) const [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 `OGRFeature::DumpReadable()`, `OGR_Fld_GetSubType()`, and `OGRFeature::SetField()`.

### 11.68.3.9 GetType()

```
OGRFieldType OGRFieldDefn::GetType ( ) const [inline]
```

Fetch type of this field.

This method is the same as the C function **OGR\_Fld\_GetType()** (p. ??).

#### Returns

field type.

Referenced by `OGRFeature::DumpReadable()`, `OGRFeature::Equal()`, `OGRFeature::FillUnsetWithDefault()`, `OGRFeature::GetFieldAsBinary()`, `OGRFeature::GetFieldAsDateTime()`, `OGRFeature::GetFieldAsDouble()`, `OGRFeature::GetFieldAsDoubleList()`, `OGRFeature::GetFieldAsInteger64()`, `OGRFeature::GetFieldAsInteger64List()`, `OGRFeature::GetFieldAsIntegerList()`, `OGRFeature::GetFieldAsSerializedJSON()`, `OGRFeature::GetFieldAsString()`, `OGRFeature::GetFieldAsStringList()`, `OGR_Fld_GetType()`, `OGRFeature::SetField()`, `OGRFeature::SetFieldNull()`, `OGRFeature::SetFieldsFrom()`, and `OGRFeature::UnsetField()`.

### 11.68.3.10 GetWidth()

```
int OGRFieldDefn::GetWidth ( ) const [inline]
```

Get the formatting width for this field.

This method is the same as the C function **OGR\_Fld\_GetWidth()** (p. ??).

#### Returns

the width, zero means no specified width.

Referenced by `OGRFeature::GetFieldAsString()`, and `OGR_Fld_GetWidth()`.

### 11.68.3.11 IsDefaultDriverSpecific()

```
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 literal 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(), and EQUAL.

Referenced by OGR\_Fld\_IsDefaultDriverSpecific().

### 11.68.3.12 IsIgnored()

```
int OGRFieldDefn::IsIgnored ( ) const [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

Referenced by OGR\_Fld\_IsIgnored().

### 11.68.3.13 IsNullable()

```
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 OGRFeature::FillUnsetWithDefault(), and OGR\_Fld\_IsNullable().

**11.68.3.14 IsSame()**

```
int OGRFieldDefn::IsSame (
    const OGRFieldDefn * poOtherFieldDefn ) const
```

Test if the field definition is identical to the other one.

**Parameters**

<i>poOtherFieldDefn</i>	the other field definition to compare to.
-------------------------	---

**Returns**

TRUE if the field definition is identical to the other one.

Referenced by OGRFeatureDefn::IsSame().

**11.68.3.15 Set()**

```
void OGRFieldDefn::Set (
    const char * pszNameIn,
    OGRFieldType eTypeIn,
    int nWidthIn = 0,
    int nPrecisionIn = 0,
    OGRJustification eJustifyIn = OJUndefined )
```

Set defining parameters for a field in one call.

This method is the same as the C function **OGR\_Fld\_Set()** (p. ??).

**Parameters**

<i>pszNameIn</i>	the new name to assign.
<i>eTypeIn</i>	the new type (one of the OFT values like OFTInteger).
<i>nWidthIn</i>	the preferred formatting width. Defaults to zero indicating undefined.
<i>n</i> ↔ <i>PrecisionIn</i>	number of decimals places for formatting, defaults to zero indicating undefined.
<i>eJustifyIn</i>	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References SetJustify(), SetName(), SetPrecision(), SetType(), and SetWidth().

**11.68.3.16 SetDefault()**

```
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 literal value enclosed between single quote characters (and inner single quote characters escaped by repetition of the single quote character), **CURRENT\_TIMESTAMP**, **CURRENT\_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

<i>pszDefaultIn</i>	new default field value or NULL pointer.
---------------------	--

#### Since

GDAL 2.0

References **CPLError()**, and **CPLFree**.

Referenced by **OGR\_Fld\_SetDefault()**, and **OGRFieldDefn()**.

#### 11.68.3.17 SetIgnored()

```
void OGRFieldDefn::SetIgnored (
    int ignore ) [inline]
```

Set whether this field should be omitted when fetching features.

This method is the same as the C function **OGR\_Fld\_SetIgnored()** (p. ??).

#### Parameters

<i>ignore</i>	ignore state
---------------	--------------

Referenced by **OGR\_Fld\_SetIgnored()**, and **OGRLayer::SetIgnoredFields()**.

#### 11.68.3.18 SetJustify()

```
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

<i>eJustify</i>	the new justification.
-----------------	------------------------

Referenced by OGR\_Fld\_SetJustify(), and Set().

#### 11.68.3.19 SetName()

```
void OGRFieldDefn::SetName (
    const char * pszNameIn )
```

Reset the name of this field.

This method is the same as the C function **OGR\_Fld\_SetName()** (p. ??).

#### Parameters

<i>pszNameIn</i>	the new name to apply.
------------------	------------------------

References CPLFree, and CPLStrdup().

Referenced by OGR\_Fld\_SetName(), and Set().

#### 11.68.3.20 SetNullable()

```
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

<i>bNullableIn</i>	FALSE if the field must have a not-null constraint.
--------------------	---

Since

GDAL 2.0

Referenced by OGR\_Fld\_SetNullable().

### 11.68.3.21 SetPrecision()

```
void OGRFieldDefn::SetPrecision (
    int nPrecision ) [inline]
```

Set the formatting precision for this field in characters.

This should normally be zero for fields of types other than OFTReal.

This method is the same as the C function **OGR\_Fld\_SetPrecision()** (p. ??).

Parameters

<i>nPrecision</i>	the new precision.
-------------------	--------------------

Referenced by OGR\_Fld\_SetPrecision(), and Set().

### 11.68.3.22 SetSubType()

```
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

<i>eSubTypeIn</i>	the new field subtype.
-------------------	------------------------

Since

GDAL 2.0

References CPLError(), and OGR\_AreTypeSubTypeCompatible().

Referenced by OGR\_Fld\_SetSubType().

### 11.68.3.23 SetType()

```
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 **OGRFeatureDefn** (p. ??).

This method is the same as the C function **OGR\_Fld\_SetType()** (p. ??).

#### Parameters

<i>eTypeIn</i>	the new field type.
----------------	---------------------

References **CPL\_Error()**, and **OGR\_AreTypeSubTypeCompatible()**.

Referenced by **OGR\_Fld\_SetType()**, and **Set()**.

### 11.68.3.24 SetWidth()

```
void OGRFieldDefn::SetWidth (
    int nWidth ) [inline]
```

Set the formatting width for this field in characters.

This method is the same as the C function **OGR\_Fld\_SetWidth()** (p. ??).

#### Parameters

<i>nWidth</i>	the new width.
---------------	----------------

References **MAX**.

Referenced by **OGR\_Fld\_SetWidth()**, and **Set()**.

### 11.68.3.25 ToHandle()

```
static OGRFieldDefnH OGRFieldDefn::ToHandle (
    OGRFieldDefn * poFieldDefn ) [inline], [static]
```

Convert a **OGRFieldDefn\*** to a **OGRFieldDefnH**.

Since

GDAL 2.3

Referenced by **OGR\_F\_GetFieldDefnRef()**, **OGR\_FD\_GetFieldDefn()**, and **OGR\_Fld\_Create()**.

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

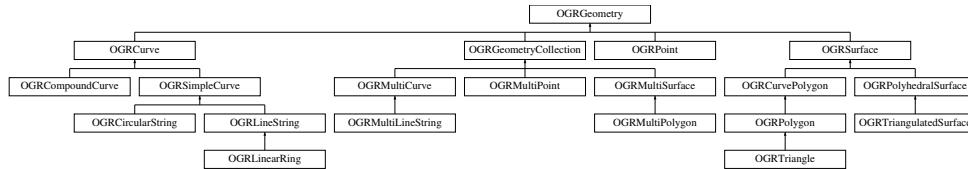
- **ogr\_feature.h**
- **ogrfielddefn.cpp**



## 11.69 OGRGeometry Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRGeometry:



### Public Member Functions

- **OGRGeometry** (const **OGRGeometry** &other)  
*Copy constructor.*
- **OGRGeometry** & **operator=** (const **OGRGeometry** &other)  
*Assignment operator.*
- bool **operator==** (const **OGRGeometry** &other) const
- bool **operator!=** (const **OGRGeometry** &other) const
- virtual int **getDimension** () const =0  
*Get the dimension of this object.*
- virtual int **getCoordinateDimension** () const  
*Get the dimension of the coordinates in this object.*
- int **CoordinateDimension** () 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.*
- **OGRBoolean** **Is3D** () const
- **OGRBoolean** **IsMeasured** () const
- virtual **OGRBoolean** **IsRing** () const  
*Test if the geometry is a ring.*
- virtual void **empty** ()=0  
*Clear geometry information. This restores the geometry to its initial state after construction, and before assignment of actual geometry.*
- virtual **OGRGeometry** \* **clone** () const **CPL\_WARN\_UNUSED\_RESULT=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.*
- **OGRErr** **importFromWkb** (const **GByte** \*, int=-1, **OGRwkbVariant**= **wkbVariantOldOgc**)  
*Assign geometry from well known binary data.*

- virtual **OGRERR** **importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumed↵  
Out)=0  
*Assign geometry from well known binary data.*
- virtual **OGRERR** **exportToWkb** ( **OGRwkbByteOrder**, unsigned char \*, **OGRwkbVariant**= **wkbVariant**↵  
**OldOgc**) const =0  
*Convert a geometry into well known binary format.*
- virtual **OGRERR** **importFromWkt** (const char \*\*ppszInput)=0  
*Assign geometry from well known text data.*
- **OGRERR** **importFromWkt** (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const  
char\*\*) instead")
- 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 \*=nullptr, char \*\*papszOptions=nullptr) 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=nullptr) const  
*Convert a geometry into GML format.*
- virtual char \* **exportToKML** () const  
*Convert a geometry into KML format.*
- virtual char \* **exportToJson** () const  
*Convert a geometry into GeoJSON format.*
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor)=0
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const =0
- virtual **GEOSGeom** **exportToGEOS** ( **GEOSContextHandle\_t** hGEOSCtx) const **CPL\_WARN\_UNUS↵  
ED\_RESULT**
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const  
*Returns if this geometry is or has curve geometry.*
- virtual **OGRGeometry** \* **getCurveGeometry** (const char \*const \*papszOptions=nullptr) const **CPL\_W↵  
ARN\_UNUSED\_RESULT**  
*Return curve version of this geometry.*
- virtual **OGRGeometry** \* **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char \*const  
\*papszOptions=nullptr) const **CPL\_WARN\_UNUSED\_RESULT**  
*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.*
- virtual void **set3D** ( **OGRBoolean** bls3D)  
*Add or remove the Z coordinate dimension.*
- virtual void **setMeasured** ( **OGRBoolean** blsMeasured)  
*Add or remove the M coordinate dimension.*
- virtual 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** (const **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 **CPL\_WARN\_UNUSED\_RESULT**  
*Compute boundary.*
- virtual double **Distance** (const **OGRGeometry** \*) const  
*Compute distance between two geometries.*
- virtual **OGRGeometry \* ConvexHull** () const **CPL\_WARN\_UNUSED\_RESULT**  
*Compute convex hull.*
- virtual **OGRGeometry \* Buffer** (double dfDist, int nQuadSegs=30) const **CPL\_WARN\_UNUSED\_RESULT**  
*Compute buffer of geometry.*
- virtual **OGRGeometry \* Intersection** (const **OGRGeometry** \*) const **CPL\_WARN\_UNUSED\_RESULT**  
*Compute intersection.*
- virtual **OGRGeometry \* Union** (const **OGRGeometry** \*) const **CPL\_WARN\_UNUSED\_RESULT**  
*Compute union.*
- virtual **OGRGeometry \* UnionCascaded** () const **CPL\_WARN\_UNUSED\_RESULT**  
*Compute union using cascading.*
- virtual **OGRGeometry \* Difference** (const **OGRGeometry** \*) const **CPL\_WARN\_UNUSED\_RESULT**  
*Compute difference.*
- virtual **OGRGeometry \* SymDifference** (const **OGRGeometry** \*) const **CPL\_WARN\_UNUSED\_RESULT**  
*Compute symmetric difference.*
- virtual **OGRErr Centroid** ( **OGRPoint** \*poPoint) const  
*Compute the geometry centroid.*
- virtual **OGRGeometry \* Simplify** (double dTolerance) const **CPL\_WARN\_UNUSED\_RESULT**  
*Simplify the geometry.*
- **OGRGeometry \* SimplifyPreserveTopology** (double dTolerance) const **CPL\_WARN\_UNUSED\_RESULT**  
*Simplify the geometry while preserving topology.*
- virtual **OGRGeometry \* DelaunayTriangulation** (double dfTolerance, int bOnlyEdges) const **CPL\_WARN\_UNUSED\_RESULT**  
*Return a Delaunay triangulation of the vertices of the geometry.*
- virtual **OGRGeometry \* Polygonize** () const **CPL\_WARN\_UNUSED\_RESULT**

*Polygonizes a set of sparse edges.*

- virtual double **Distance3D** (const **OGRGeometry** \*poOtherGeom) const

*Returns the 3D distance between two geometries.*

- virtual void **swapXY** ()

*Swap x and y coordinates.*

- **OGRPoint** \* **toPoint** ()
- const **OGRPoint** \* **toPoint** () const
- **OGRCurve** \* **toCurve** ()
- const **OGRCurve** \* **toCurve** () const
- **OGRSimpleCurve** \* **toSimpleCurve** ()
- const **OGRSimpleCurve** \* **toSimpleCurve** () const
- **OGRLineString** \* **toLineString** ()
- const **OGRLineString** \* **toLineString** () const
- **OGRLinearRing** \* **toLinearRing** ()
- const **OGRLinearRing** \* **toLinearRing** () const
- **OGRCircularString** \* **toCircularString** ()
- const **OGRCircularString** \* **toCircularString** () const
- **OGRCompoundCurve** \* **toCompoundCurve** ()
- const **OGRCompoundCurve** \* **toCompoundCurve** () const
- **OGRSurface** \* **toSurface** ()
- const **OGRSurface** \* **toSurface** () const
- **OGRPolygon** \* **toPolygon** ()
- const **OGRPolygon** \* **toPolygon** () const
- **OGRTriangle** \* **toTriangle** ()
- const **OGRTriangle** \* **toTriangle** () const
- **OGRCurvePolygon** \* **toCurvePolygon** ()
- const **OGRCurvePolygon** \* **toCurvePolygon** () const
- **OGRGeometryCollection** \* **toGeometryCollection** ()
- const **OGRGeometryCollection** \* **toGeometryCollection** () const
- **OGRMultiPoint** \* **toMultiPoint** ()
- const **OGRMultiPoint** \* **toMultiPoint** () const
- **OGRMultiLineString** \* **toMultiLineString** ()
- const **OGRMultiLineString** \* **toMultiLineString** () const
- **OGRMultiPolygon** \* **toMultiPolygon** ()
- const **OGRMultiPolygon** \* **toMultiPolygon** () const
- **OGRMultiCurve** \* **toMultiCurve** ()
- const **OGRMultiCurve** \* **toMultiCurve** () const
- **OGRMultiSurface** \* **toMultiSurface** ()
- const **OGRMultiSurface** \* **toMultiSurface** () const
- **OGRPolyhedralSurface** \* **toPolyhedralSurface** ()
- const **OGRPolyhedralSurface** \* **toPolyhedralSurface** () const
- **OGRTriangulatedSurface** \* **toTriangulatedSurface** ()
- const **OGRTriangulatedSurface** \* **toTriangulatedSurface** () const

## Static Public Member Functions

- static **GEOSContextHandle\_t** **createGEOSContext** ()
- static void **freeGEOSContext** ( **GEOSContextHandle\_t** hGEOSCtx)
- static **OGRGeometryH** **ToHandle** ( **OGRGeometry** \*poGeom)
- static **OGRGeometry** \* **FromHandle** ( **OGRGeometryH** hGeom)

### 11.69.1 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](http://www.opengeospatial.org/standards/sfa)

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. ??)), COMPOUNDCURVE (**OGRCompoundCurve** (p. ??)), CURVEPOLYGON (**OGRCurvePolygon** (p. ??)), MULTICURVE (**OGRMultiCurve** (p. ??)) and MULTISURFACE (**OGRMultiSurface** (p. ??)).

### 11.69.2 Constructor & Destructor Documentation

#### 11.69.2.1 OGRGeometry()

```
OGRGeometry::OGRGeometry (
    const OGRGeometry & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

References `OGRSpatialReference::Reference()`.

### 11.69.3 Member Function Documentation

#### 11.69.3.1 accept() [1/2]

```
virtual void OGRGeometry::accept (
    IOGRGeometryVisitor * visitor ) [pure virtual]
```

Accept a visitor.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRTriangulatedSurface** (p. ??), **OGRPolyhedralSurface** (p. ??), **OGRMultiPolygon** (p. ??), **OGRMultiSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRTriangle** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), **OGRLinearRing** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

### 11.69.3.2 accept() [2/2]

```
virtual void OGRGeometry::accept (
    IOGRConstGeometryVisitor * visitor ) const [pure virtual]
```

Accept a visitor.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRTriangulatedSurface** (p. ??), **OGRPolyhedralSurface** (p. ??), **OGRMultiPolygon** (p. ??), **OGRMultiSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRTriangle** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), **OGRLinearRing** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

### 11.69.3.3 assignSpatialReference()

```
void OGRGeometry::assignSpatialReference (
    OGRSpatialReference * poSR ) [virtual]
```

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.

Starting with GDAL 2.3, this will also assign the spatial reference to potential sub-geometries of the geometry (**OGRGeometryCollection** (p. ??), **OGRCurvePolygon**/**OGRPolygon**, **OGRCompoundCurve** (p. ??), **OGRPolyhedralSurface** (p. ??) and their derived classes).

This is similar to the SFCOM **IGeometry::put\_SpatialReference()** method.

This method is the same as the C function **OGR\_G\_AssignSpatialReference()** (p. ??).

#### Parameters

<i>poSR</i>	new spatial reference system to apply.
-------------	--

Reimplemented in **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), and **OGRCompoundCurve** (p. ??).

References **OGRSpatialReference::Reference()**, and **OGRSpatialReference::Release()**.

Referenced by **OGRGeometryCollection::assignSpatialReference()**, **OGRPolyhedralSurface::assignSpatialReference()**, **OGRPoint::clone()**, **OGRSimpleCurve::clone()**, **OGRLinearRing::clone()**, **OGRGeometryFactory::createFromWkb()**, **OGRGeometryFactory::createFromWkt()**, **OGRGeometryFactory::curveFromLineString()**, **OGRCircularString::CurveToLine()**, **OGRGeometryFactory::forceTo()**, **OGRGeometryFactory::forceToLineString()**, **OGRSimpleCurve::getSubLine()**, **operator=()**, and **OGRPoint::transform()**.

## 11.69.3.4 Boundary()

```
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 CPLError().

## 11.69.3.5 Buffer()

```
OGRGeometry * OGRGeometry::Buffer (
    double dfDist,
    int nQuadSegs = 30 ) const [virtual]
```

Compute buffer of geometry.

Builds a new geometry containing the buffer region around the geometry on which it is invoked. The buffer is a polygon containing the region within the buffer distance of the original geometry.

Some buffer sections are properly described as curves, but are converted to approximate polygons. The nQuadSegs parameter can be used to control how many segments should be used to define a 90 degree curve - a quadrant of a circle. A value of 30 is a reasonable default. Large values result in large numbers of vertices in the resulting buffer geometry while small numbers reduce the accuracy of the result.

This method is the same as the C function **OGR\_G\_Buffer()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>dfDist</i>	the buffer distance to be applied. Should be expressed into the same unit as the coordinates of the geometry.
<i>nQuadSegs</i>	the number of segments used to approximate a 90 degree (quadrant) of curvature.

**Returns**

the newly created geometry, or NULL if an error occurs.

References CPLError().

**11.69.3.6 Centroid()**

```
OGRERR 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. ??))

< Failure

< Failure

References CPLError(), and OGRERR\_FAILURE.

**11.69.3.7 clone()**

```
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 **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRLinearRing** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRCompoundCurve::addCurve(), OGRGeometryCollection::addGeometry(), OGRPolyhedralSurface::addGeometry(), OGRCurvePolygon::addRing(), OGRFeature::CopySelfTo(), OGRGeometryFactory::forceToPolygon(), getCurveGeometry(), OGRLayer::GetFeature(), getLinearGeometry(), OGRFeature::SetGeomField(), and OGRGeometryFactory::transformWithOptions().



## 11.69.3.8 closeRings()

```
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 OGRGeometryCollection::closeRings(), and OGR\_G\_CloseRings().

## 11.69.3.9 Contains()

```
OGRBoolean OGRGeometry::Contains (
    const OGRGeometry * ) const [virtual]
```

Test for containment.

Tests if actual geometry object contains the passed geometry.

This method is the same as the C function **OGR\_G\_Contains()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

## Returns

TRUE if poOtherGeom contains this geometry, otherwise FALSE.

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

References CPLError().

Referenced by OGRCurvePolygon::Contains().

## 11.69.3.10 ConvexHull()

```
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 CPLError().

**11.69.3.11 CoordinateDimension()**

```
int OGRGeometry::CoordinateDimension ( ) const
```

Get the dimension of the coordinates in this object.

This method is the same as the C function **OGR\_G\_CoordinateDimension()** (p. ??).

**Returns**

this will return 2 for XY, 3 for XYZ and XYM, and 4 for XYZM data.

**Since**

GDAL 2.1

Referenced by OGRSimpleCurve::exportToWkb(), OGRSimpleCurve::importFromWkb(), OGR\_G\_CoordinateDimension(), and OGRSimpleCurve::WkbSize().

**11.69.3.12 createGEOSContext()**

```
GEOSContextHandle_t OGRGeometry::createGEOSContext ( ) [static]
```

Create a new GEOS context.

**Returns**

a new GEOS context.

References CPLError().

Referenced by Intersects().

**11.69.3.13 Crosses()**

```
OGRBoolean OGRGeometry::Crosses (
    const OGRGeometry * ) const [virtual]
```

Test for crossing.

Tests if this geometry and the other passed into the method are crossing.

This method is the same as the C function **OGR\_G\_Crosses()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

## Returns

TRUE if they are crossing, otherwise FALSE.

References CPLError().

## 11.69.3.14 DelaunayTriangulation()

```
OGRGeometry * OGRGeometry::DelaunayTriangulation (
    double dfTolerance,
    int bOnlyEdges ) const [virtual]
```

Return a Delaunay triangulation of the vertices of the geometry.

This function is the same as the C function **OGR\_G\_DelaunayTriangulation()** (p. ??).

This function is built on the GEOS library, v3.4 or above. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>dfTolerance</i>	optional snapping tolerance to use for improved robustness
<i>bOnlyEdges</i>	if TRUE, will return a MULTILINESTRING, otherwise it will return a GEOMETRYCOLLECTION containing triangular POLYGONS.

## Returns

the geometry resulting from the Delaunay triangulation or NULL if an error occurs.

## Since

OGR 2.1

References CPLError().

## 11.69.3.15 Difference()

```
OGRGeometry * OGRGeometry::Difference (
    const OGRGeometry * ) const [virtual]
```

Compute difference.

Generates a new geometry which is the region of this geometry with the region of the second geometry removed.

This method is the same as the C function **OGR\_G\_Difference()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

**Parameters**

<i>poOtherGeom</i>	the other geometry removed from "this" geometry.
--------------------	--

**Returns**

a new geometry representing the difference or NULL if the difference is empty or an error occurs.

References CPLError().

**11.69.3.16 Disjoint()**

```
OGRBoolean OGRGeometry::Disjoint (
    const OGRGeometry * ) const [virtual]
```

Test for disjointness.

Tests if this geometry and the other passed into the method are disjoint.

This method is the same as the C function **OGR\_G\_Disjoint()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

**Parameters**

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

**Returns**

TRUE if they are disjoint, otherwise FALSE.

References CPLError().

**11.69.3.17 Distance()**

```
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

<i>poOtherGeom</i>	the other geometry to compare against.
--------------------	--

## Returns

the distance between the geometries or -1 if an error occurs.

References CPLDebug(), and CPLError().

## 11.69.3.18 Distance3D()

```
double OGRGeometry::Distance3D (
    const OGRGeometry * poOtherGeom ) const [virtual]
```

Returns the 3D distance between two geometries.

The distance is expressed into the same unit as the coordinates of the geometries.

This method is built on the SFCGAL library, check it for the definition of the geometry operation. If OGR is built without the SFCGAL library, this method will always return -1.0

This function is the same as the C function **OGR\_G\_Distance3D()** (p. ??).

## Returns

distance between the two geometries

## Since

GDAL 2.2

References CPLDebug(), CPLError(), and Is3D().

## 11.69.3.19 dumpReadable()

```
void OGRGeometry::dumpReadable (
    FILE * fp,
    const char * pszPrefix = nullptr,
    char ** ppszOptions = nullptr ) const [virtual]
```

Dump geometry in well known text format to indicated output file.

A few options can be defined to change the default dump :

- DISPLAY\_GEOMETRY=NO : to hide the dump of the geometry
- DISPLAY\_GEOMETRY=WKT or YES (default) : dump the geometry as a WKT
- DISPLAY\_GEOMETRY=SUMMARY : to get only a summary of the geometry

This method is the same as the C function **OGR\_G\_DumpReadable()** (p. ??).

## Parameters

<i>fp</i>	the text file to write the geometry to.
<i>pszPrefix</i>	the prefix to put on each line of output.
<i>papszOptions</i>	NULL terminated list of options (may be NULL)

&lt; Success

&lt; Success

References CPLFree, CPLTestBool(), CSLFetchNameValue(), dumpReadable(), EQUAL, exportToWkt(), OGRCompoundCurve::getCurve(), OGRCurvePolygon::getExteriorRingCurve(), getGeometryName(), getGeometryType(), OGRCurvePolygon::getInteriorRingCurve(), OGRCompoundCurve::getNumCurves(), OGRGeometryCollection::getNumGeometries(), OGRPolyhedralSurface::getNumGeometries(), OGRCurvePolygon::getNumInteriorRings(), OGRCurve::getNumPoints(), OGRSimpleCurve::getNumPoints(), OGRERR\_NONE, toCompoundCurve(), toCurvePolygon(), toGeometryCollection(), toPolyhedralSurface(), toSimpleCurve(), wkbCircularString, wkbCircularStringM, wkbCircularStringZ, wkbCircularStringZM, wkbCompoundCurve, wkbCompoundCurveM, wkbCompoundCurveZ, wkbCompoundCurveZM, wkbCurve, wkbCurveM, wkbCurvePolygon, wkbCurvePolygonM, wkbCurvePolygonZ, wkbCurvePolygonZM, wkbCurveZ, wkbCurveZM, wkbFlatten, wkbGeometryCollection, wkbGeometryCollection25D, wkbGeometryCollectionM, wkbGeometryCollectionZM, wkbLinearRing, wkbLineString, wkbLineString25D, wkbLineStringM, wkbLineStringZM, wkbMultiCurve, wkbMultiCurveM, wkbMultiCurveZ, wkbMultiCurveZM, wkbMultiLineString, wkbMultiLineString25D, wkbMultiLineStringM, wkbMultiLineStringZM, wkbMultiPoint, wkbMultiPoint25D, wkbMultiPointM, wkbMultiPointZM, wkbMultiPolygon, wkbMultiPolygon25D, wkbMultiPolygonM, wkbMultiPolygonZM, wkbMultiSurface, wkbMultiSurfaceM, wkbMultiSurfaceZ, wkbMultiSurfaceZM, wkbNone, wkbPoint, wkbPoint25D, wkbPointM, wkbPointZM, wkbPolygon, wkbPolygon25D, wkbPolygonM, wkbPolygonZM, wkbPolyhedralSurface, wkbPolyhedralSurfaceM, wkbPolyhedralSurfaceZ, wkbPolyhedralSurfaceZM, wkbSurface, wkbSurfaceM, wkbSurfaceZ, wkbSurfaceZM, wkbTIN, wkbTINM, wkbTINZ, wkbTINZM, wkbTriangle, wkbTriangleM, wkbTriangleZ, wkbTriangleZM, wkbUnknown, and wkbVariantIso.

Referenced by dumpReadable(), OGRFeature::DumpReadable(), and OGR\_G\_DumpReadable().

## 11.69.3.20 empty()

```
void OGRGeometry::empty ( ) [pure virtual]
```

Clear geometry information. This restores the geometry to its 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 **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGR\_G\_Empty().

## 11.69.3.21 Equals()

```
int OGRGeometry::Equals (
    const OGRGeometry * ) const [pure virtual]
```

Returns TRUE if two geometries are equivalent.

This operation implements the SQL/MM ST\_OrderingEquals() operation.

The comparison is done in a structural way, that is to say that the geometry types must be identical, as well as the number and ordering of sub-geometries and vertices. Or equivalently, two geometries are considered equal by this method if their WKT/WKB representation is equal. Note: this must be distinguished for equality in a spatial way (which is the purpose of the ST\_Equals() operation).

This method is the same as the C function **OGR\_G\_Equals()** (p. ??).

## Returns

TRUE if equivalent or FALSE otherwise.

Implemented in **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRFeature::Equal(), and OGRGeometryCollection::Equals().

## 11.69.3.22 exportToGEOS()

```
GEOSGeom OGRGeometry::exportToGEOS (
    GEOSContextHandle_t hGEOSctxt ) const [virtual]
```

Returns a GEOSGeom object corresponding to the geometry.

## Parameters

<i>hGEOSctxt</i>	GEOS context
------------------	--------------

## Returns

a GEOSGeom object corresponding to the geometry.

References CPLError().

Referenced by Intersects().

### 11.69.3.23 exportToGML()

```
char * OGRGeometry::exportToGML (
    const char *const * papszOptions = nullptr ) 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()** (p. ??) when no longer required.

The supported options in OGR 1.8.0 are :

- **FORMAT=GML3**. Otherwise it will default to GML 2.1.2 output.
- **GML3\_LINESTRING\_ELEMENT=curve**. (Only valid for **FORMAT=GML3**) To use gml:Curve element for linestrings. Otherwise gml:LineString will be used .
- **GML3\_LONGSRS=YES/NO**. (Only valid for **FORMAT=GML3**) Default to YES. If YES, SRS with EPSG authority will be written with the "urn:ogc:def:crs:EPSG::" prefix. In the case, if the SRS is a geographic SRS without explicit AXIS order, but that the same SRS authority code imported with **ImportFromEPSGA()** should be treated as lat/long, then the function will take care of coordinate order swapping. If set to NO, SRS with EPSG authority will be written with the "EPSG:" prefix, even if they are in lat/long order.

This method is the same as the C function **OGR\_G\_ExportToGMLEx()** (p. ??).

#### Parameters

<i>papszOptions</i>	NULL-terminated list of options.
---------------------	----------------------------------

#### Returns

A GML fragment or NULL in case of error.

References **OGR\_G\_ExportToGMLEx()**, and **ToHandle()**.

### 11.69.3.24 exportToJson()

```
char * OGRGeometry::exportToJson ( ) const [virtual]
```

Convert a geometry into GeoJSON format.

The returned string should be freed with **CPLFree()** (p. ??) 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 **OGR\_G\_ExportToJson()**, and **ToHandle()**.



## 11.69.3.25 exportToKML()

```
char * OGRGeometry::exportToKML ( ) const [virtual]
```

Convert a geometry into KML format.

The returned string should be freed with **CPLFree()** (p. ??) 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 **OGR\_G\_ExportToKML()**, and **ToHandle()**.

## 11.69.3.26 exportToWkb()

```
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

<i>eByteOrder</i>	One of <b>wkbXDR</b> or <b>wkbNDR</b> indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
<i>eWkbVariant</i>	What standard to use when exporting geometries with three dimensions (or more). The default <b>wkbVariantOldOgc</b> is the historical OGR variant. <b>wkbVariantIso</b> 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 **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), **OGRLinearRing** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by **OGRGeometryCollection::exportToWkb()**.

## 11.69.3.27 exportToWkt()

```

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

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

## Returns

Currently OGRERR\_NONE is always returned.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRPolyhedralSurface** (p. ??), **OGRMultiPolygon** (p. ??), **OGRMultiSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by dumpReadable(), and OGR\_G\_ExportToWkt().

## 11.69.3.28 flattenTo2D()

```

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 **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGR\_G\_FlattenTo2D().

## 11.69.3.29 freeGEOSContext()

```

void OGRGeometry::freeGEOSContext (
    GEOSContextHandle_t hGEOSctxt ) [static]

```

Destroy a GEOS context.

## Parameters

<i>hGEOSCtxt</i>	GEOS context
------------------	--------------

Referenced by Intersects().

## 11.69.3.30 FromHandle()

```
static OGRGeometry* OGRGeometry::FromHandle (
    OGRGeometryH hGeom ) [inline], [static]
```

Convert a OGRGeometryH to a OGRGeometry\*.

## Since

GDAL 2.3

Referenced by OGRGeometryFactory::createFromGML(), OGR\_DS\_ExecuteSQL(), OGR\_F\_SetGeometry(), OGR\_F\_SetGeometryDirectly(), OGR\_F\_SetGeomField(), OGR\_F\_SetGeomFieldDirectly(), OGR\_G\_AssignSpatialReference(), OGR\_G\_Boundary(), OGR\_G\_Buffer(), OGR\_G\_Centroid(), OGR\_G\_Clone(), OGR\_G\_CloseRings(), OGR\_G\_Contains(), OGR\_G\_ConvexHull(), OGR\_G\_CoordinateDimension(), OGR\_G\_Crosses(), OGR\_G\_DelaunayTriangulation(), OGR\_G\_Difference(), OGR\_G\_Disjoint(), OGR\_G\_Distance(), OGR\_G\_Distance3D(), OGR\_G\_DumpReadable(), OGR\_G\_Empty(), OGR\_G\_ExportToIsoWkb(), OGR\_G\_ExportToIsoWkt(), OGR\_G\_ExportToWkb(), OGR\_G\_ExportToWkt(), OGR\_G\_FlattenTo2D(), OGR\_G\_GetCoordinateDimension(), OGR\_G\_GetDimension(), OGR\_G\_GetEnvelope(), OGR\_G\_GetEnvelope3D(), OGR\_G\_GetGeometryName(), OGR\_G\_GetGeometryType(), OGR\_G\_GetSpatialReference(), OGR\_G\_ImportFromWkb(), OGR\_G\_ImportFromWkt(), OGR\_G\_Intersection(), OGR\_G\_Intersects(), OGR\_G\_Is3D(), OGR\_G\_IsEmpty(), OGR\_G\_IsMeasured(), OGR\_G\_IsRing(), OGR\_G\_IsSimple(), OGR\_G\_IsValid(), OGR\_G\_Overlaps(), OGR\_G\_Polygonize(), OGR\_G\_Set3D(), OGR\_G\_SetCoordinateDimension(), OGR\_G\_SetMeasured(), OGR\_G\_Simplify(), OGR\_G\_SimplifyPreserveTopology(), OGR\_G\_SwapXY(), OGR\_G\_SymDifference(), OGR\_G\_Touches(), OGR\_G\_Transform(), OGR\_G\_TransformTo(), OGR\_G\_Union(), OGR\_G\_UnionCascaded(), OGR\_G\_Within(), OGR\_G\_WkbSize(), OGR\_L\_SetSpatialFilter(), and OGR\_L\_SetSpatialFilterEx().

## 11.69.3.31 getCoordinateDimension()

```
int OGRGeometry::getCoordinateDimension ( ) const [virtual]
```

Get the dimension of the coordinates in this object.

This method is the same as the C function **OGR\_G\_GetCoordinateDimension()** (p. ??).

**Deprecated** use **CoordinateDimension()** (p. ??).

## Returns

this will return 2 or 3.

Referenced by OGRCurvePolygon::CastToPolygon(), OGRCircularString::CurveToLine(), OGRGeometryCollection::exportToWkb(), OGRSimpleCurve::getSubLine(), OGR\_G\_GetCoordinateDimension(), OGRSimpleCurve::setPoints(), OGRSimpleCurve::setZ(), OGRSimpleCurve::Value(), and OGRCircularString::Value().

### 11.69.3.32 `getCurveGeometry()`

```
OGRGeometry * OGRGeometry::getCurveGeometry (
    const char *const * papszOptions = nullptr ) 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

<i>papszOptions</i>	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 `OGRGeometryCollection::getCurveGeometry()`.

### 11.69.3.33 `getDimension()`

```
int OGRGeometry::getDimension ( ) const [pure virtual]
```

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. ??)).

This method is the same as the C function **OGR\_G\_GetDimension()** (p. ??).

#### Returns

0 for points, 1 for lines and 2 for surfaces.

Implemented in **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRPolyhedralSurface** (p. ??), **OGRMultiSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by `OGR_G_GetDimension()`.

11.69.3.34 `getEnvelope()` [1/2]

```
void OGRGeometry::getEnvelope (
    OGREnvelope * psEnvelope ) const [pure virtual]
```

Computes and returns the bounding envelope for this geometry in the passed `psEnvelope` structure.

This method is the same as the C function `OGR_G_GetEnvelope()` (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implemented in `OGRPolyhedralSurface` (p. ??), `OGRGeometryCollection` (p. ??), `OGRCurvePolygon` (p. ??), `OGRCompoundCurve` (p. ??), `OGRCircularString` (p. ??), `OGRSimpleCurve` (p. ??), and `OGRPoint` (p. ??).

Referenced by `OGRGeometryCollection::getEnvelope()`, `Intersects()`, `OGR_G_GetEnvelope()`, `OGR_G_GetEnvelope3D()`, and `OGRGeometryFactory::transformWithOptions()`.

11.69.3.35 `getEnvelope()` [2/2]

```
void OGRGeometry::getEnvelope (
    OGREnvelope3D * psEnvelope ) const [pure virtual]
```

Computes and returns the bounding envelope (3D) for this geometry in the passed `psEnvelope` structure.

This method is the same as the C function `OGR_G_GetEnvelope3D()` (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Since

OGR 1.9.0

Implemented in `OGRPolyhedralSurface` (p. ??), `OGRGeometryCollection` (p. ??), `OGRCurvePolygon` (p. ??), `OGRCompoundCurve` (p. ??), `OGRCircularString` (p. ??), `OGRSimpleCurve` (p. ??), and `OGRPoint` (p. ??).

11.69.3.36 `getGeometryName()`

```
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. ??), **OGRTriangulatedSurface** (p. ??), **OGRPolyhedralSurface** (p. ??), **OGRMultiPolygon** (p. ??), **OGRMultiSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRTriangle** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), **OGRLinearRing** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by **OGRTriangulatedSurface::addGeometry()**, **dumpReadable()**, **OGRSimpleCurve::exportToWkt()**, **OGRFeature::GetFieldAsString()**, and **OGR\_G\_GetGeometryName()**.

**11.69.3.37 getGeometryType()**

```
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. ??), **OGRTriangulatedSurface** (p. ??), **OGRPolyhedralSurface** (p. ??), **OGRMultiPolygon** (p. ??), **OGRMultiSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRTriangle** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by **OGRPolyhedralSurface::addGeometry()**, **OGRGeometryCollection::addGeometryDirectly()**, **OGRPolyhedralSurface::addGeometryDirectly()**, **OGRCurve::CastToCompoundCurve()**, **OGRSimpleCurve::clone()**, **OGRGeometryCollection::closeRings()**, **OGRCurvePolygon::Contains()**, **dumpReadable()**, **OGRPoint::Equals()**, **OGRSimpleCurve::Equals()**, **OGRCompoundCurve::Equals()**, **OGRCurvePolygon::Equals()**, **OGRGeometryCollection::Equals()**, **OGRPolyhedralSurface::Equals()**, **OGRSimpleCurve::exportToWkb()**, **OGRGeometryFactory::forceTo()**, **OGRGeometryFactory::forceToLineString()**, **OGRGeometryFactory::forceToMultiLineString()**, **OGRGeometryFactory::forceToMultiPoint()**, **OGRGeometryFactory::forceToMultiPolygon()**, **OGRGeometryFactory::forceToPolygon()**, **OGRGeometryCollection::get\_Area()**, **OGRGeometryCollection::get\_Length()**, **getIsoGeometryType()**, **OGRPoint::Intersects()**, **OGRCurvePolygon::Intersects()**, **OGR\_F\_GetGeometryRef()**, **OGR\_F\_GetGeomFieldRef()**, **OGR\_G\_Centroid()**, **OGR\_G\_GetGeometryType()**, **OGRGeometryFactory::transformWithOptions()**, and **OGRPoint::Within()**.

11.69.3.38 `getIsoGeometryType()`

```
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 `getGeometryType()`, and `wkbFlatten`.

Referenced by `OGRPoint::exportToWkb()`, `OGRSimpleCurve::exportToWkb()`, `OGRPolygon::exportToWkb()`, `OGRGeometryCollection::exportToWkb()`, and `OGRPolyhedralSurface::exportToWkb()`.

11.69.3.39 `getLinearGeometry()`

```
OGRGeometry * OGRGeometry::getLinearGeometry (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) 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**

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings. See <b><code>OGRGeometryFactory::curveToLineString()</code></b> (p. ??) for valid options.

**Returns**

a new geometry.

**Since**

GDAL 2.0

Reimplemented in **`OGRGeometryCollection`** (p. ??), **`OGRPolygon`** (p. ??), **`OGRCurvePolygon`** (p. ??), **`OGRCompoundCurve`** (p. ??), and **`OGRCircularString`** (p. ??).

References clone().

Referenced by OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRPolygon::getLinearGeometry(), and OGRGeometryCollection::getLinearGeometry().

#### 11.69.3.40 getSpatialReference()

```
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 OGRCurve::CastToCompoundCurve(), OGRCurvePolygon::CastToPolygon(), OGRTriangulatedSurface::CastToPolyhedralSurface(), OGRPoint::clone(), OGRSimpleCurve::clone(), OGRLinearRing::clone(), OGRCompoundCurve::clone(), OGRCurvePolygon::clone(), OGRGeometryCollection::clone(), OGRPolyhedralSurface::clone(), OGRGeometryFactory::curveFromLineString(), OGRCurvePolygon::CurvePolyToPoly(), OGRCircularString::CurveToLine(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRPolygon::getCurveGeometry(), OGRGeometryCollection::getCurveGeometry(), OGRGeometryCollection::getLinearGeometry(), OGRSimpleCurve::getSubLine(), operator=(), OGRTriangulatedSurface::operator=(), and transformTo().

#### 11.69.3.41 hasCurveGeometry()

```
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

<i>bLookForNonLinear</i>	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. ??), **OGRPolyhedralSurface** (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()**.

11.69.3.42 **importFromWkb()** [1/2]

```
OGRERR OGRGeometry::importFromWkb (
    const GByte * pabyData,
    int nSize = -1,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc )
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	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.

Referenced by `OGRGeometryFactory::createFromWkb()`.

### 11.69.3.43 `importFromWkb()` [2/2]

```
OGRERR OGRGeometry::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [pure virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM `IWks::ImportFromWKB()` method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

#### Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of <i>pabyData</i> in bytes, or -1 if not known.
<i>eWkbVariant</i>	if <code>wkbVariantPostGIS1</code> , special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

#### 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.

#### Since

GDAL 2.3

Implemented in **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRTriangle** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), **OGRLinearRing** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

### 11.69.3.44 `importFromWkt()` [1/2]

```
OGRERR OGRGeometry::importFromWkt (
    const char ** ppszInput ) [pure virtual]
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM `IWks::ImportFromWKT()` method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppsInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
-----------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

Implemented in **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRPolyhedralSurface** (p. ??), **OGRMultiSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRGeometryFactory::createFromWkt(), OGR\_G\_ImportFromWkt(), and OSRImportFromWkt().

## 11.69.3.45 importFromWkt() [2/2]

```
OGRERR OGRGeometry::importFromWkt (
    char ** ppsInput ) const [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

## 11.69.3.46 Intersection()

```
OGRGeometry * OGRGeometry::Intersection (
    const OGRGeometry * ) const [virtual]
```

Compute intersection.

Generates a new geometry which is the region of intersection of the two geometries operated on. The **Intersects()** (p. ??) method can be used to test if two geometries intersect.

This method is the same as the C function **OGR\_G\_Intersection()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the other geometry intersected with "this" geometry.
--------------------	--

**Returns**

a new geometry representing the intersection or NULL if there is no intersection or an error occurs.

References CPLError().

**11.69.3.47 Intersects()**

```
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 rigorous fashion otherwise TRUE is returned if the envelopes (bounding boxes) of the two geometries overlap.

The *poOtherGeom* argument may be safely NULL, but in this case the method will always return TRUE. That is, a NULL geometry is treated as being everywhere.

This method is the same as the C function **OGR\_G\_Intersects()** (p. ??).

**Parameters**

<i>poOtherGeom</i>	the other geometry to test against.
--------------------	-------------------------------------

**Returns**

TRUE if the geometries intersect, otherwise FALSE.

Reimplemented in **OGRCurvePolygon** (p. ??), and **OGRPoint** (p. ??).

References createGEOSContext(), exportToGEOS(), freeGEOSContext(), and getEnvelope().

Referenced by OGRPoint::Intersects(), and OGRCurvePolygon::Intersects().

**11.69.3.48 Is3D()**

```
OGRBoolean OGRGeometry::Is3D ( ) const [inline]
```

Returns whether the geometry has a Z component.

Referenced by OGRSimpleCurve::addPoint(), Distance3D(), OGRPoint::exportToWkb(), OGRSimpleCurve::exportToWkb(), OGRPolygon::exportToWkb(), OGRPoint::exportToWkt(), OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), OGRCurve::get\_IsClosed(), OGR\_G\_Is3D(), and OGRTriangulatedSurface::operator=().

## 11.69.3.49 isEmpty()

```
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 **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRCurve::CastToCompoundCurve(), OGRSimpleCurve::Equals(), OGRCurvePolygon::Equals(), OGRGeometryCollection::Equals(), OGRPolyhedralSurface::Equals(), OGRGeometryFactory::forceTo(), OGRGeometryCollection::getEnvelope(), OGRGeometryCollection::IsEmpty(), and OGR\_G\_IsEmpty().

## 11.69.3.50 IsMeasured()

```
OGRBoolean OGRGeometry::IsMeasured ( ) const [inline]
```

Returns whether the geometry has a M component.

Referenced by OGRSimpleCurve::addPoint(), OGRPoint::exportToWkb(), OGRSimpleCurve::exportToWkb(), OGRCompoundCurve::exportToWkb(), OGRPoint::exportToWkt(), OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), OGR\_G\_IsMeasured(), and OGRTriangulatedSurface::operator=().

## 11.69.3.51 IsRing()

```
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.

References CPL\_Error().

Referenced by OGR\_G\_IsRing().

### 11.69.3.52 IsSimple()

```
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.

References CPLError().

Referenced by OGR\_G\_IsSimple().

### 11.69.3.53 IsValid()

```
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. ??).

References CPLError().

Referenced by OGRCircularString::IsValid(), and OGR\_G\_IsValid().

### 11.69.3.54 operator!=( )

```
bool OGRGeometry::operator!= (
    const OGRGeometry & other ) const [inline]
```

Returns if two geometries are different.

## 11.69.3.55 operator=()

```
OGRGeometry & OGRGeometry::operator= (
    const OGRGeometry & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

Since

GDAL 2.1

References assignSpatialReference(), and getSpatialReference().

Referenced by OGRPoint::operator=(), OGRSimpleCurve::operator=(), OGRCompoundCurve::operator=(), OGRCurvePolygon::operator=(), OGRGeometryCollection::operator=(), OGRPolyhedralSurface::operator=(), and OGRTriangulatedSurface::operator=().

## 11.69.3.56 operator==()

```
bool OGRGeometry::operator== (
    const OGRGeometry & other ) const [inline]
```

Returns if two geometries are equal.

## 11.69.3.57 Overlaps()

```
OGRBoolean OGRGeometry::Overlaps (
    const OGRGeometry * ) const [virtual]
```

Test for overlap.

Tests if this geometry and the other passed into the method overlap, that is their intersection has a non-zero area.

This method is the same as the C function **OGR\_G\_Overlaps()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

Returns

TRUE if they are overlapping, otherwise FALSE.

References CPLError().

**11.69.3.58 Polygonize()**

```
OGRGeometry * OGRGeometry::Polygonize ( ) const [virtual]
```

Polygonizes a set of sparse edges.

A new geometry object is created and returned containing a collection of reassembled Polygons: NULL will be returned if the input collection doesn't corresponds to a MultiLinestring, or when reassembling Edges into Polygons is impossible due to topological inconsistencies.

This method is the same as the C function **OGR\_G\_Polygonize()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

**Returns**

a newly allocated geometry now owned by the caller, or NULL on failure.

**Since**

OGR 1.9.0

References CPLError().

**11.69.3.59 segmentize()**

```
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**

<i>dfMaxLength</i>	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 consecutive vertices is lower than dfMaxLength.

**Parameters**

<i>dfMaxLength</i>	maximum length between 2 consecutive vertices.
--------------------	--



Reimplemented in **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRCircularString** (p. ??), and **OGRSimpleCurve** (p. ??).

Referenced by OGRGeometryCollection::segmentize().

### 11.69.3.60 set3D()

```
void OGRGeometry::set3D (
    OGRBoolean bIs3D ) [virtual]
```

Add or remove the Z coordinate dimension.

This method adds or removes the explicit Z coordinate dimension. Removing the Z coordinate dimension of a geometry will remove any existing Z values. Adding the Z dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

#### Parameters

<i>bIs3D</i>	Should the geometry have a Z dimension, either TRUE or FALSE.
--------------	---

#### Since

GDAL 2.1

Reimplemented in **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), and **OGRSimpleCurve** (p. ??).

Referenced by OGR\_G\_Set3D(), OGRGeometryCollection::set3D(), and OGRPolyhedralSurface::set3D().

### 11.69.3.61 setCoordinateDimension()

```
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, a compound curve, a polygon, etc. will affect the children geometries. This will also remove the M dimension if present before this call.

**Deprecated** use **set3D()** (p. ??) or **setMeasured()** (p. ??).

#### Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

Reimplemented in **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

References `setMeasured()`.

Referenced by `OGRGeometryCollection::setCoordinateDimension()`, and `OGRPolyhedralSurface::setCoordinateDimension()`.

### 11.69.3.62 `setMeasured()`

```
void OGRGeometry::setMeasured (
    OGRBoolean bIsMeasured ) [virtual]
```

Add or remove the M coordinate dimension.

This method adds or removes the explicit M coordinate dimension. Removing the M coordinate dimension of a geometry will remove any existing M values. Adding the M dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

#### Parameters

<i>bIsMeasured</i>	Should the geometry have a M dimension, either TRUE or FALSE.
--------------------	---

Since

GDAL 2.1

Reimplemented in **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), and **OGRSimpleCurve** (p. ??).

Referenced by `OGRPoint::flattenTo2D()`, `OGR_G_SetMeasured()`, `setCoordinateDimension()`, `OGRPoint::setCoordinateDimension()`, `OGRGeometryCollection::setMeasured()`, and `OGRPolyhedralSurface::setMeasured()`.

### 11.69.3.63 `Simplify()`

```
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

<i>dTolerance</i>	the distance tolerance for the simplification.
-------------------	--

**Returns**

the simplified geometry or NULL if an error occurs.

**Since**

OGR 1.8.0

References CPLError().

**11.69.3.64 SimplifyPreserveTopology()**

```
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**

<i>dTolerance</i>	the distance tolerance for the simplification.
-------------------	--

**Returns**

the simplified geometry or NULL if an error occurs.

**Since**

OGR 1.9.0

References CPLError().

**11.69.3.65 swapXY()**

```
void OGRGeometry::swapXY ( ) [virtual]
```

Swap x and y coordinates.

**Since**

OGR 1.8.0

Reimplemented in **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGR\_G\_SwapXY(), and OGRGeometryCollection::swapXY().

## 11.69.3.66 SymDifference()

```
OGRGeometry * OGRGeometry::SymDifference (
    const OGRGeometry * ) 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

<i>poOtherGeom</i>	the other geometry.
--------------------	---------------------

## Returns

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

## Since

OGR 1.8.0

References CPLError().

## 11.69.3.67 toCircularString() [1/2]

```
OGRCircularString* OGRGeometry::toCircularString ( ) [inline]
```

Down-cast to OGRCircularString\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkb↔CircularString.

## Since

GDAL 2.3

## 11.69.3.68 toCircularString() [2/2]

```
const OGRCircularString* OGRGeometry::toCircularString ( ) const [inline]
```

Down-cast to OGRCircularString\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkb↔CircularString.

## Since

GDAL 2.3

## 11.69.3.69 toCompoundCurve() [1/2]

```
OGRCompoundCurve* OGRGeometry::toCompoundCurve ( ) [inline]
```

Down-cast to OGRCompoundCurve\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p.??) == wkbCompoundCurve.

Since

GDAL 2.3

Referenced by dumpReadable(), and OGRCompoundCurve::Equals().

## 11.69.3.70 toCompoundCurve() [2/2]

```
const OGRCompoundCurve* OGRGeometry::toCompoundCurve ( ) const [inline]
```

Down-cast to OGRCompoundCurve\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p.??) == wkbCompoundCurve.

Since

GDAL 2.3

## 11.69.3.71 toCurve() [1/2]

```
OGRCurve* OGRGeometry::toCurve ( ) [inline]
```

Down-cast to OGRCurve\*. Implies prior checking that OGR\_GT\_IsSubClass(getGeometryType(), wkbCurve).

Since

GDAL 2.3

Referenced by OGRCompoundCurve::addCurve(), OGRCurvePolygon::addRing(), OGRGeometryFactory::curve↔FromLineString(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometry↔Factory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), OGRGeometryCollection::get\_Area(), OGRGeometryCollection::get\_Length(), and OGR\_G\_GetPointCount().

**11.69.3.72 toCurve()** [2/2]

```
const OGRCurve* OGRGeometry::toCurve ( ) const [inline]
```

Down-cast to OGRCurve\*. Implies prior checking that `OGR_GT_IsSubClass(getGeometryType(), wkbCurve)`.

Since

GDAL 2.3

**11.69.3.73 toCurvePolygon()** [1/2]

```
OGRCurvePolygon* OGRGeometry::toCurvePolygon ( ) [inline]
```

Down-cast to OGRCurvePolygon\*. Implies prior checking that `wkbFlatten(getGeometryType()) (p. ??) == wkbCurvePolygon or wkbPolygon or wkbTriangle`.

Since

GDAL 2.3

Referenced by `OGRCurvePolygon::clone()`, `OGRGeometryCollection::closeRings()`, `dumpReadable()`, `OGRCurvePolygon::Equals()`, `OGRGeometryFactory::forceToLineString()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRGeometryFactory::forceToMultiPolygon()`, `OGRGeometryFactory::forceToPolygon()`, `OGRPoint::Intersects()`, and `OGRPoint::Within()`.

**11.69.3.74 toCurvePolygon()** [2/2]

```
const OGRCurvePolygon* OGRGeometry::toCurvePolygon ( ) const [inline]
```

Down-cast to OGRCurvePolygon\*. Implies prior checking that `wkbFlatten(getGeometryType()) (p. ??) == wkbCurvePolygon or wkbPolygon or wkbTriangle`.

Since

GDAL 2.3

## 11.69.3.75 toGeometryCollection() [1/2]

```
OGRGeometryCollection* OGRGeometry::toGeometryCollection ( ) [inline]
```

Down-cast to OGRGeometryCollection\*. Implies prior checking that OGR\_GT\_IsSubClass(getGeometryType(), wkbGeometryCollection).

Since

GDAL 2.3

Referenced by OGRGeometryCollection::clone(), dumpReadable(), OGRGeometryCollection::Equals(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRGeometryCollection::get\_Area(), OGRGeometryCollection::get\_Length(), OGRGeometryCollection::getCurveGeometry(), OGRGeometryCollection::getLinearGeometry(), and OGRGeometryFactory::transformWithOptions().

## 11.69.3.76 toGeometryCollection() [2/2]

```
const OGRGeometryCollection* OGRGeometry::toGeometryCollection ( ) const [inline]
```

Down-cast to OGRGeometryCollection\*. Implies prior checking that OGR\_GT\_IsSubClass(getGeometryType(), wkbGeometryCollection).

Since

GDAL 2.3

## 11.69.3.77 ToHandle()

```
static OGRGeometryH OGRGeometry::ToHandle (
    OGRGeometry * poGeom ) [inline], [static]
```

Convert a OGRGeometry\* to a OGRGeometryH.

Since

GDAL 2.3

Referenced by exportToGML(), exportToJson(), exportToKML(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsInteger64(), OGRFeature::GetFieldAsString(), OGRFeature::IsFieldSet(), OGR\_F\_GetGeometryRef(), OGR\_F\_GetGeomFieldRef(), OGR\_F\_StealGeometry(), OGR\_G\_Boundary(), OGR\_G\_Buffer(), OGR\_G\_Clone(), OGR\_G\_ConvexHull(), OGR\_G\_DelaunayTriangulation(), OGR\_G\_Difference(), OGR\_G\_Intersection(), OGR\_G\_Polygonize(), OGR\_G\_Simplify(), OGR\_G\_SimplifyPreserveTopology(), OGR\_G\_SymDifference(), OGR\_G\_Union(), OGR\_G\_UnionCascaded(), and OGR\_L\_GetSpatialFilter().

**11.69.3.78 toLinearRing()** [1/2]

```
OGRLinearRing* OGRGeometry::toLinearRing ( ) [inline]
```

Down-cast to OGRLinearRing\*. Implies prior checking that EQUAL(**getGeometryName()** (p. ??), "LINEARRING").

Since

GDAL 2.3

Referenced by OGRPolygon::exportToWkt(), OGRPolygon::getExteriorRing(), OGRPolygon::getInteriorRing(), OGRPolygon::stealExteriorRing(), and OGRPolygon::stealInteriorRing().

**11.69.3.79 toLinearRing()** [2/2]

```
const OGRLinearRing* OGRGeometry::toLinearRing ( ) const [inline]
```

Down-cast to OGRLinearRing\*. Implies prior checking that EQUAL(**getGeometryName()** (p. ??), "LINEARRING").

Since

GDAL 2.3

**11.69.3.80 toLineString()** [1/2]

```
OGRLineString* OGRGeometry::toLineString ( ) [inline]
```

Down-cast to OGRLineString\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbLineString.

Since

GDAL 2.3

Referenced by OGRLineString::CurveToLine(), and OGRGeometryFactory::forceToLineString().

**11.69.3.81 toLineString()** [2/2]

```
const OGRLineString* OGRGeometry::toLineString ( ) const [inline]
```

Down-cast to OGRLineString\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbLineString.

Since

GDAL 2.3



## 11.69.3.82 toMultiCurve() [1/2]

```
OGRMultiCurve* OGRGeometry::toMultiCurve ( ) [inline]
```

Down-cast to OGRMultiCurve\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbMulti↔Curve and derived types.

Since

GDAL 2.3

Referenced by OGRGeometryFactory::forceToMultiLineString().

## 11.69.3.83 toMultiCurve() [2/2]

```
const OGRMultiCurve* OGRGeometry::toMultiCurve ( ) const [inline]
```

Down-cast to OGRMultiCurve\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbMulti↔Curve and derived types.

Since

GDAL 2.3

## 11.69.3.84 toMultiLineString() [1/2]

```
OGRMultiLineString* OGRGeometry::toMultiLineString ( ) [inline]
```

Down-cast to OGRMultiLineString\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkb↔MultiLineString.

Since

GDAL 2.3

Referenced by OGRGeometryFactory::forceToMultiLineString().

## 11.69.3.85 toMultiLineString() [2/2]

```
const OGRMultiLineString* OGRGeometry::toMultiLineString ( ) const [inline]
```

Down-cast to OGRMultiLineString\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkb↔MultiLineString.

Since

GDAL 2.3

**11.69.3.86 toMultiPoint()** [1/2]

```
OGRMultiPoint* OGRGeometry::toMultiPoint ( ) [inline]
```

Down-cast to OGRMultiPoint\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbMulti↔Point.

Since

GDAL 2.3

**11.69.3.87 toMultiPoint()** [2/2]

```
const OGRMultiPoint* OGRGeometry::toMultiPoint ( ) const [inline]
```

Down-cast to OGRMultiPoint\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbMulti↔Point.

Since

GDAL 2.3

**11.69.3.88 toMultiPolygon()** [1/2]

```
OGRMultiPolygon* OGRGeometry::toMultiPolygon ( ) [inline]
```

Down-cast to OGRMultiPolygon\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkb↔MultiPolygon.

Since

GDAL 2.3

Referenced by OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToMultiLineString(), and OGR↔GeometryFactory::forceToMultiPolygon().

**11.69.3.89 toMultiPolygon()** [2/2]

```
const OGRMultiPolygon* OGRGeometry::toMultiPolygon ( ) const [inline]
```

Down-cast to OGRMultiPolygon\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkb↔MultiPolygon.

Since

GDAL 2.3

**11.69.3.90 toMultiSurface()** [1/2]

```
OGRMultiSurface* OGRGeometry::toMultiSurface ( ) [inline]
```

Down-cast to OGRMultiSurface\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbMultiSurface and derived types.

Since

GDAL 2.3

Referenced by OGRGeometryFactory::forceToMultiPolygon().

**11.69.3.91 toMultiSurface()** [2/2]

```
const OGRMultiSurface* OGRGeometry::toMultiSurface ( ) const [inline]
```

Down-cast to OGRMultiSurface\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbMultiSurface and derived types.

Since

GDAL 2.3

**11.69.3.92 toPoint()** [1/2]

```
OGRPoint* OGRGeometry::toPoint ( ) [inline]
```

Down-cast to OGRPoint\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbPoint.

Since

GDAL 2.3

Referenced by OGRCurvePolygon::Contains(), OGRPoint::Equals(), OGRCurvePolygon::Intersects(), OGR\_G\_AddPoint(), OGR\_G\_AddPoint\_2D(), OGR\_G\_AddPointM(), OGR\_G\_AddPointZM(), OGR\_G\_GetPoint(), OGR\_G\_GetPoints(), OGR\_G\_GetPointsZM(), OGR\_G\_GetPointZM(), OGR\_G\_SetPoint(), OGR\_G\_SetPoint\_2D(), OGR\_G\_SetPointM(), OGR\_G\_SetPointZM(), and OGRGeometryFactory::transformWithOptions().

**11.69.3.93 toPoint()** [2/2]

```
const OGRPoint* OGRGeometry::toPoint ( ) const [inline]
```

Down-cast to OGRPoint\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbPoint.

Since

GDAL 2.3

**11.69.3.94 toPolygon()** [1/2]

```
OGRPolygon* OGRGeometry::toPolygon ( ) [inline]
```

Down-cast to OGRPolygon\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbPolygon or wkbTriangle.

Since

GDAL 2.3

Referenced by OGRTriangulatedSurface::addGeometry(), OGRPolygon::CurvePolyToPoly(), OGRGeometry↔Factory::forceTo(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), and OGRPolyhedralSurface::importFromWkt().

**11.69.3.95 toPolygon()** [2/2]

```
const OGRPolygon* OGRGeometry::toPolygon ( ) const [inline]
```

Down-cast to OGRPolygon\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbPolygon or wkbTriangle.

Since

GDAL 2.3

**11.69.3.96 toPolyhedralSurface()** [1/2]

```
OGRPolyhedralSurface* OGRGeometry::toPolyhedralSurface ( ) [inline]
```

Down-cast to OGRPolyhedralSurface\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbPolyhedralSurface or wkbTIN.

Since

GDAL 2.3

Referenced by OGRPolyhedralSurface::clone(), dumpReadable(), OGRPolyhedralSurface::Equals(), OGR↔GeometryFactory::forceTo(), OGRGeometryFactory::forceToMultiPolygon(), and OGRGeometryFactory::force↔ToPolygon().

**11.69.3.97 toPolyhedralSurface()** [2/2]

```
const OGRPolyhedralSurface* OGRGeometry::toPolyhedralSurface ( ) const [inline]
```

Down-cast to OGRPolyhedralSurface\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbPolyhedralSurface or wkbTIN.

Since

GDAL 2.3

**11.69.3.98 toSimpleCurve()** [1/2]

```
OGRSimpleCurve* OGRGeometry::toSimpleCurve ( ) [inline]
```

Down-cast to OGRSimpleCurve\*. Implies prior checking that **getGeometryType()** (p. ??) is wkbLineString, wkbCircularString or a derived type.

Since

GDAL 2.3

Referenced by OGRSimpleCurve::clone(), dumpReadable(), OGRSimpleCurve::Equals(), OGR\_G\_AddPoint(), OGR\_G\_AddPoint\_2D(), OGR\_G\_AddPointM(), OGR\_G\_AddPointZM(), OGR\_G\_GetPoints(), OGR\_G\_GetPointsZM(), and OGR\_G\_SetPointCount().

**11.69.3.99 toSimpleCurve()** [2/2]

```
const OGRSimpleCurve* OGRGeometry::toSimpleCurve ( ) const [inline]
```

Down-cast to OGRSimpleCurve\*. Implies prior checking that **getGeometryType()** (p. ??) is wkbLineString, wkbCircularString or a derived type.

Since

GDAL 2.3

**11.69.3.100 toSurface()** [1/2]

```
OGRSurface* OGRGeometry::toSurface ( ) [inline]
```

Down-cast to OGRSurface\*. Implies prior checking that OGR\_GT\_IsSubClass(getGeometryType(), wkbSurface).

Since

GDAL 2.3

Referenced by OGRGeometryFactory::forceToPolygon(), OGRGeometryCollection::get\_Area(), and OGRMultiSurface::importFromWkt().

**11.69.3.101 toSurface()** [2/2]

```
const OGRSurface* OGRGeometry::toSurface ( ) const [inline]
```

Down-cast to OGRSurface\*. Implies prior checking that `OGR_GT_IsSubClass(getGeometryType(), wkbSurface)`.

Since

GDAL 2.3

**11.69.3.102 toTriangle()** [1/2]

```
OGRTriangle* OGRGeometry::toTriangle ( ) [inline]
```

Down-cast to OGRTriangle\*. Implies prior checking that `wkbFlatten(getGeometryType()) (p. ??) == wkbTriangle`.

Since

GDAL 2.3

**11.69.3.103 toTriangle()** [2/2]

```
const OGRTriangle* OGRGeometry::toTriangle ( ) const [inline]
```

Down-cast to OGRTriangle\*. Implies prior checking that `wkbFlatten(getGeometryType()) (p. ??) == wkbTriangle`.

Since

GDAL 2.3

**11.69.3.104 toTriangulatedSurface()** [1/2]

```
OGRTriangulatedSurface* OGRGeometry::toTriangulatedSurface ( ) [inline]
```

Down-cast to OGRTriangulatedSurface\*. Implies prior checking that `wkbFlatten(getGeometryType()) (p. ??) == wkbTIN`.

Since

GDAL 2.3

Referenced by OGRGeometryFactory::forceTo().

## 11.69.3.105 toTriangulatedSurface() [2/2]

```
const OGRTriangulatedSurface* OGRGeometry::toTriangulatedSurface ( ) const [inline]
```

Down-cast to OGRTriangulatedSurface\*. Implies prior checking that **wkbFlatten(getGeometryType())** (p. ??) == wkbTIN.

Since

GDAL 2.3

## 11.69.3.106 Touches()

```
OGRBoolean OGRGeometry::Touches (
    const OGRGeometry * ) const [virtual]
```

Test for touching.

Tests if this geometry and the other passed into the method are touching.

This method is the same as the C function **OGR\_G\_Touches()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

## Returns

TRUE if they are touching, otherwise FALSE.

References CPLError().

## 11.69.3.107 transform()

```
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

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

## Returns

OGRERR\_NONE on success or an error code.

Implemented in **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRLinearRing** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRGeometryCollection::transform(), and OGRGeometryFactory::transformWithOptions().

## 11.69.3.108 transformTo()

```
OGRErr OGRGeometry::transformTo (
    OGRSpatialReference * poSR )
```

Transform geometry to new spatial reference system.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

This method will only work if the geometry already has an assigned spatial reference system, and if it is transformable to the target coordinate system.

Because this method requires internal creation and initialization of an **OGRCoordinateTransformation** (p. ??) object it is significantly more expensive to use this method to transform many geometries than it is to create the **OGRCoordinateTransformation** (p. ??) in advance, and call **transform()** (p. ??) with that transformation. This method exists primarily for convenience when only transforming a single geometry.

This method is the same as the C function **OGR\_G\_TransformTo()** (p. ??).

## Parameters

<i>poSR</i>	spatial reference system to transform to.
-------------	---

## Returns

OGRERR\_NONE on success, or an error code.

< Failure

< Failure

< Failure

References CPLError(), and getSpatialReference().



## 11.69.3.109 Union()

```
OGRGeometry * OGRGeometry::Union (
    const OGRGeometry * ) const [virtual]
```

Compute union.

Generates a new geometry which is the region of union of the two geometries operated on.

This method is the same as the C function **OGR\_G\_Union()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the other geometry unioned with "this" geometry.
--------------------	--

## Returns

a new geometry representing the union or NULL if an error occurs.

References CPLError().

## 11.69.3.110 UnionCascaded()

```
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 CPLError().

## 11.69.3.111 Within()

```
OGRBoolean OGRGeometry::Within (
    const OGRGeometry * ) const [virtual]
```

Test for containment.

Tests if actual geometry object is within the passed geometry.

This method is the same as the C function **OGR\_G\_Within()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

**Parameters**

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

**Returns**

TRUE if poOtherGeom is within this geometry, otherwise FALSE.

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

References CPLError().

Referenced by OGRPoint::Within().

**11.69.3.112 WkbSize()**

```
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 **OGRPolyhedralSurface** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRCompoundCurve** (p. ??), **OGRLinearRing** (p. ??), **OGRSimpleCurve** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGR\_G\_WkbSize().

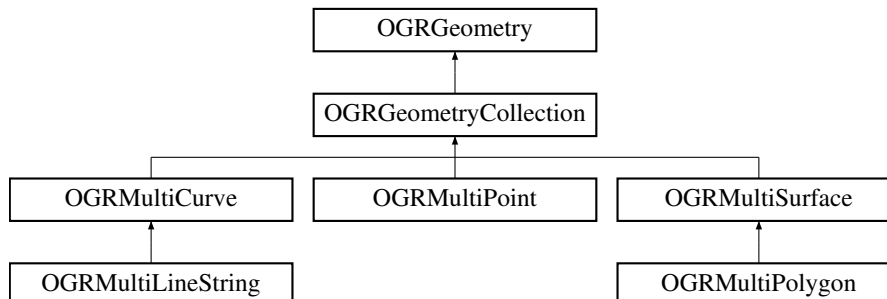
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrgeometry.cpp

## 11.70 OGRGeometryCollection Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRGeometryCollection:



### Public Types

- typedef **OGRGeometry** **ChildType**

### Public Member Functions

- **OGRGeometryCollection** ()  
*Create an empty geometry collection.*
- **OGRGeometryCollection** (const **OGRGeometryCollection** &other)  
*Copy constructor.*
- **OGRGeometryCollection** & **operator=** (const **OGRGeometryCollection** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Fetch geometry type.*
- virtual **OGRGeometry** \* **clone** () const override  
*Make a copy of this object.*
- virtual void **empty** () override  
*Clear geometry information. This restores the geometry to its initial state after construction, and before assignment of actual geometry.*
- virtual **OGRErr** **transform** (**OGRCoordinateTransformation** \*poCT) override  
*Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** () override  
*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual **OGRBoolean** **isEmpty** () const override  
*Returns TRUE (non-zero) if the object has no points.*
- virtual void **segmentize** (double dfMaxLength) override  
*Modify the geometry such it has no segment longer then the given distance.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override

- Returns if this geometry is or has curve geometry.*

  - virtual **OGRGeometry \* getCurveGeometry** (const char \*const \*papszOptions=nullptr) const override

*Return curve version of this geometry.*
- virtual **OGRGeometry \* getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*papszOptions=nullptr) const override

*Return, possibly approximate, non-curve version of this geometry.*
- virtual int **WkbSize** () const override

*Returns size of related binary representation.*
- virtual **OGRErr importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumedOut) override

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

*Convert a geometry into well known binary format.*
- **OGRErr importFromWkt** (const char \*\*) override
- virtual **OGRErr exportToWkt** (char \*\*papszDstText, **OGRwkbVariant**= **wkbVariantOldOgc**) const override

*Convert a geometry into well known text format.*
- virtual double **get\_Length** () const

*Compute the length of a multcurve.*
- virtual double **get\_Area** () const

*Compute area of geometry collection.*
- virtual int **getDimension** () const override

*Get the dimension of this object.*
- virtual void **getEnvelope** (OGREnvelope \*psEnvelope) const override

*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (OGREnvelope3D \*psEnvelope) const override

*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.*
- const **OGRGeometry \* getGeometryRef** (int) const

*Fetch geometry from container.*
- virtual **OGRBoolean Equals** (const **OGRGeometry \***) const override

*Returns TRUE if two geometries are equivalent.*
- virtual void **setCoordinateDimension** (int nDimension) override

*Set the coordinate dimension.*
- virtual void **set3D** ( **OGRBoolean** bls3D) override

*Add or remove the Z coordinate dimension.*
- virtual void **setMeasured** ( **OGRBoolean** blsMeasured) override

*Add or remove the M 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.*
- virtual void **assignSpatialReference** ( **OGRSpatialReference** \*poSR) override

*Assign spatial reference to this object.*
- void **closeRings** () override

*Force rings to be closed.*

- virtual void **swapXY** () override  
*Swap x and y coordinates.*
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override
- virtual **OGRErr** **importFromWkt** (const char \*\*ppszInput)=0  
*Assign geometry from well known text data.*
- **OGRErr** **importFromWkt** (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

### Static Public Member Functions

- static **OGRGeometryCollection** \* **CastToGeometryCollection** ( **OGRGeometryCollection** \*poSrc)  
*Cast to geometry collection.*

### Protected Member Functions

- virtual **OGRBoolean** **isCompatibleSubType** ( **OGRwkbGeometryType**) const

#### 11.70.1 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.

#### 11.70.2 Member Typedef Documentation

##### 11.70.2.1 ChildType

```
typedef OGRGeometry OGRGeometryCollection::ChildType
```

Type of child elements.

#### 11.70.3 Constructor & Destructor Documentation

### 11.70.3.1 OGRGeometryCollection()

```
OGRGeometryCollection::OGRGeometryCollection (
    const OGRGeometryCollection & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

References `clone()`, and `VSI_CALLOC_VERBOSE`.

## 11.70.4 Member Function Documentation

### 11.70.4.1 `accept()` [1/2]

```
virtual void OGRGeometryCollection::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRMulti**↔**Polygon** (p. ??), and **OGRMultiSurface** (p. ??).

References `IOGRGeometryVisitor::visit()`.

### 11.70.4.2 `accept()` [2/2]

```
virtual void OGRGeometryCollection::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRMulti**↔**Polygon** (p. ??), and **OGRMultiSurface** (p. ??).

References `IOGRConstGeometryVisitor::visit()`.

### 11.70.4.3 `addGeometry()`

```
OGRERR OGRGeometryCollection::addGeometry (
    const OGRGeometry * poNewGeom ) [virtual]
```

Add a geometry to the container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. The passed geometry is cloned to make an internal copy.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_AddGeometry()** (p. ??).

## Parameters

<i>poNewGeom</i>	geometry to add to the container.
------------------	-----------------------------------

## Returns

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

< Failure  
< Success

References addGeometryDirectly(), OGRGeometry::clone(), OGRERR\_FAILURE, and OGRERR\_NONE.

Referenced by clone(), and operator=().

## 11.70.4.4 addGeometryDirectly()

```
OGRERR OGRGeometryCollection::addGeometryDirectly (
    OGRGeometry * poNewGeom ) [virtual]
```

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. ??) does.

This method is the same as the C function **OGR\_G\_AddGeometryDirectly()** (p. ??).

There is no SFCOM analog to this method.

## Parameters

<i>poNewGeom</i>	geometry to add to the container.
------------------	-----------------------------------

## Returns

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

< Unsupported geometry type

< Failure  
< Success

References OGRGeometry::getGeometryType(), isCompatibleSubType(), OGRERR\_FAILURE, OGRERR\_NONE, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, and VSI\_REALLOC\_VERBOSE.

Referenced by addGeometry(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), getCurveGeometry(), getLinearGeometry(), and OGRMultiPoint::importFromWkt().

#### 11.70.4.5 assignSpatialReference()

```
void OGRGeometryCollection::assignSpatialReference (
    OGRSpatialReference * poSR ) [override], [virtual]
```

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.

Starting with GDAL 2.3, this will also assign the spatial reference to potential sub-geometries of the geometry (**OGRGeometryCollection** (p. ??), **OGRCurvePolygon**/**OGRPolygon**, **OGRCompoundCurve** (p. ??), **OGRPolyhedralSurface** (p. ??) and their derived classes).

This is similar to the SFCOM IGeometry::put\_SpatialReference() method.

This method is the same as the C function **OGR\_G\_AssignSpatialReference()** (p. ??).

##### Parameters

<i>poSR</i>	new spatial reference system to apply.
-------------	--

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

References **OGRGeometry::assignSpatialReference()**.

Referenced by **clone()**, **OGRGeometryFactory::forceTo()**, **OGRGeometryFactory::forceToMultiLineString()**, **OGRGeometryFactory::forceToMultiPoint()**, **OGRGeometryFactory::forceToMultiPolygon()**, **getCurveGeometry()**, **getLinearGeometry()**, and **transform()**.

#### 11.70.4.6 begin() [1/2]

```
ChildType** OGRGeometryCollection::begin ( ) [inline]
```

Return begin of sub-geometry iterator.

Since

GDAL 2.3

#### 11.70.4.7 begin() [2/2]

```
const ChildType* const* OGRGeometryCollection::begin ( ) const [inline]
```

Return begin of sub-geometry iterator.

Since

GDAL 2.3



## 11.70.4.8 CastToGeometryCollection()

```
OGRGeometryCollection * OGRGeometryCollection::CastToGeometryCollection (
    OGRGeometryCollection * poSrc ) [static]
```

Cast to geometry collection.

This methods cast a derived class of geometry collection to a plain geometry collection.

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

## Parameters

<i>poSrc</i>	the input geometry - ownership is passed to the method.
--------------	---

## Returns

new geometry.

## Since

GDAL 2.2

References [getGeometryType\(\)](#), [OGRGeometryCollection\(\)](#), [wkbFlatten](#), and [wkbGeometryCollection](#).

Referenced by [OGRGeometryFactory::forceTo\(\)](#).

## 11.70.4.9 clone()

```
OGRGeometry * OGRGeometryCollection::clone ( ) const [override], [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.

< Success

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

References [addGeometry\(\)](#), [assignSpatialReference\(\)](#), [OGRGeometryFactory::createGeometry\(\)](#), [getGeometryType\(\)](#), [OGRGeometry::getSpatialReference\(\)](#), [OGRERR\\_NONE](#), and [OGRGeometry::toGeometryCollection\(\)](#).

Referenced by [getCurveGeometry\(\)](#), and [OGRGeometryCollection\(\)](#).

**11.70.4.10 closeRings()**

```
void OGRGeometryCollection::closeRings ( ) [override], [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(), OGRGeometry::getGeometryType(), OGR\_GT\_IsSubClassOf(), OGRGeometry::toCurvePolygon(), wkbCurvePolygon, and wkbFlatten.

**11.70.4.11 empty()**

```
void OGRGeometryCollection::empty ( ) [override], [virtual]
```

Clear geometry information. This restores the geometry to its 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 CPLFree.

Referenced by operator=().

**11.70.4.12 end()** [1/2]

```
ChildType** OGRGeometryCollection::end ( ) [inline]
```

Return end of sub-geometry iterator.

**11.70.4.13 end()** [2/2]

```
const ChildType* const* OGRGeometryCollection::end ( ) const [inline]
```

Return end of sub-geometry iterator.

## 11.70.4.14 Equals()

```
OGRBoolean OGRGeometryCollection::Equals (
    const OGRGeometry * ) const [override], [virtual]
```

Returns TRUE if two geometries are equivalent.

This operation implements the SQL/MM ST\_OrderingEquals() operation.

The comparison is done in a structural way, that is to say that the geometry types must be identical, as well as the number and ordering of sub-geometries and vertices. Or equivalently, two geometries are considered equal by this method if their WKT/WKB representation is equal. Note: this must be distinguished for equality in a spatial way (which is the purpose of the ST\_Equals() operation).

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()**, **IsEmpty()**, and **OGRGeometry::toGeometryCollection()**.

## 11.70.4.15 exportToWkb()

```
OGRErr OGRGeometryCollection::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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

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

**Returns**

Currently OGRERR\_NONE is always returned.

< Success

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

References CPL\_SWAP32, CPL\_Error(), OGRGeometry::exportToWkb(), OGRGeometry::getCoordinateDimension(), getGeometryType(), OGRGeometry::getIsoGeometryType(), wkbFlatten, wkbHasZ, wkbMultiCurve, wkbMultiSurface, wkbVariantIso, wkbVariantOldOgc, and wkbVariantPostGIS1.

**11.70.4.16 exportToWkt()**

```
OGRERR OGRGeometryCollection::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

**Parameters**

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

**Returns**

Currently OGRERR\_NONE is always returned.

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

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRMultiPolygon** (p. ??), and **OGRMultiSurface** (p. ??).

**11.70.4.17 flattenTo2D()**

```
void OGRGeometryCollection::flattenTo2D ( ) [override], [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. ??).

11.70.4.18 `get_Area()`

```
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 `OGRCurve::get_Area()`, `OGRSurface::get_Area()`, `get_Area()`, `OGRGeometry::getGeometryType()`, `OGR_GT_IsCurve()`, `OGR_GT_IsSubClassOf()`, `OGR_GT_IsSurface()`, `OGRGeometry::toCurve()`, `OGRGeometry::toGeometryCollection()`, `OGRGeometry::toSurface()`, `wkbFlatten`, `wkbGeometryCollection`, and `wkbMultiSurface`.

Referenced by `get_Area()`.

11.70.4.19 `get_Length()`

```
double OGRGeometryCollection::get_Length ( ) const [virtual]
```

Compute the length of a multcurve.

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 length.

References `OGRCurve::get_Length()`, `get_Length()`, `OGRGeometry::getGeometryType()`, `OGR_GT_IsCurve()`, `OGR_GT_IsSubClassOf()`, `OGRGeometry::toCurve()`, `OGRGeometry::toGeometryCollection()`, `wkbFlatten`, `wkbGeometryCollection`, and `wkbMultiCurve`.

Referenced by `get_Length()`.

11.70.4.20 `getCurveGeometry()`

```
OGRGeometry * OGRGeometryCollection::getCurveGeometry (
    const char *const * papszOptions = nullptr ) const [override], [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

<i>papszOptions</i>	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()`, `assignSpatialReference()`, `clone()`, `OGRGeometryFactory::createGeometry()`, `OGRGeometry::getCurveGeometry()`, `getGeometryType()`, `OGRGeometry::getSpatialReference()`, `OGRGeometry::hasCurveGeometry()`, `OGR_GT_GetCurve()`, and `OGRGeometry::toGeometryCollection()`.

11.70.4.21 `getDimension()`

```
int OGRGeometryCollection::getDimension ( ) const [override], [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. ??).

11.70.4.22 `getEnvelope()` [1/2]

```
void OGRGeometryCollection::getEnvelope (
    OGREnvelope * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope for this geometry in the passed `psEnvelope` structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

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

## 11.70.4.23 getEnvelope() [2/2]

```
void OGRGeometryCollection::getEnvelope (
    OGREnvelope3D * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope (3D) for this geometry in the passed *psEnvelope* structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

## Since

OGR 1.9.0

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

References **OGRGeometry::getEnvelope()**, and **OGRGeometry::IsEmpty()**.

## 11.70.4.24 getGeometryName()

```
const char * OGRGeometryCollection::getGeometryName ( ) const [override], [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. ??), **OGRMultiPolyline** (p. ??), **OGRMultiPolygon** (p. ??), and **OGRMultiSurface** (p. ??).

11.70.4.25 `getGeometryRef()` [1/2]

```
OGRGeometry * OGRGeometryCollection::getGeometryRef (
    int i )
```

Fetch geometry from container.

This method returns a pointer to a geometry within the container. The returned geometry remains owned by the container, and should not be modified. The pointer is only valid until the next change to the geometry container. Use `IGeometry::clone()` to make a copy.

This method relates to the SFCOM `IGeometryCollection::get_Geometry()` method.

## Parameters

<i>i</i>	the index of the geometry to fetch, between 0 and <b><code>getNumGeometries()</code></b> (p. ??) - 1.
----------	---

## Returns

pointer to requested geometry.

Referenced by `Equals()`, `OGRGeometryFactory::forceTo()`, `OGRGeometryFactory::forceToLineString()`, `OGRGeometryFactory::forceToMultiPolygon()`, `OGRGeometryFactory::forceToPolygon()`, and `OGRGeometryFactory::transformWithOptions()`.

11.70.4.26 `getGeometryRef()` [2/2]

```
const OGRGeometry * OGRGeometryCollection::getGeometryRef (
    int i ) const
```

Fetch geometry from container.

This method returns a pointer to a geometry within the container. The returned geometry remains owned by the container, and should not be modified. The pointer is only valid until the next change to the geometry container. Use `IGeometry::clone()` to make a copy.

This method relates to the SFCOM `IGeometryCollection::get_Geometry()` method.

## Parameters

<i>i</i>	the index of the geometry to fetch, between 0 and <b><code>getNumGeometries()</code></b> (p. ??) - 1.
----------	---

## Returns

pointer to requested geometry.



11.70.4.27 `getGeometryType()`

```
OGRwkbGeometryType OGRGeometryCollection::getGeometryType ( ) const [override], [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 **wkbGeometryCollection**, **wkbGeometryCollection25D**, **wkbGeometryCollectionM**, and **wkbGeometryCollectionZM**.

Referenced by **CastToGeometryCollection()**, **clone()**, **Equals()**, **exportToWkb()**, **getCurveGeometry()**, and **getLinearGeometry()**.

11.70.4.28 `getLinearGeometry()`

```
OGRGeometry * OGRGeometryCollection::getLinearGeometry (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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**

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings. See <b>OGRGeometryFactory::curveToLineString()</b> (p. ??) for valid options.

**Returns**

a new geometry.

**Since**

GDAL 2.0

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

References `addGeometryDirectly()`, `assignSpatialReference()`, `OGRGeometryFactory::createGeometry()`, `getGeometryType()`, `OGRGeometry::getLinearGeometry()`, `OGRGeometry::getSpatialReference()`, `OGR_GT_GetLinear()`, and `OGRGeometry::toGeometryCollection()`.

Referenced by `OGRGeometryFactory::forceToLineString()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRGeometryFactory::forceToMultiPolygon()`, and `OGRGeometryFactory::forceToPolygon()`.

**11.70.4.29 getNumGeometries()**

```
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()`, `OGRGeometryFactory::forceTo()`, `OGRGeometryFactory::forceToLineString()`, `OGRGeometryFactory::forceToMultiPolygon()`, `OGRGeometryFactory::forceToPolygon()`, and `OGRGeometryFactory::transformWithOptions()`.

**11.70.4.30 hasCurveGeometry()**

```
OGRBoolean OGRGeometryCollection::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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

<i>bLookForNonLinear</i>	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. ??), **OGRMultiPolygon** (p. ??), and **OGRMultiSurface** (p. ??).

Referenced by OGRMultiSurface::hasCurveGeometry(), and OGRMultiCurve::hasCurveGeometry().

## 11.70.4.31 importFromWkb()

```
OGRERR OGRGeometryCollection::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	if wkbVariantPostGIS1, special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

Since

GDAL 2.3

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

#### 11.70.4.32 importFromWkt() [1/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

#### 11.70.4.33 importFromWkt() [2/3]

```
OGRERR OGRGeometry::importFromWkt
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

#### Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

#### Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

#### 11.70.4.34 importFromWkt() [3/3]

```
OGRERR OGRGeometryCollection::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated

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

Reimplemented in **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), and **OGRMultiSurface** (p. ??).

## 11.70.4.35 isCompatibleSubType()

```
OGRBoolean OGRGeometryCollection::isCompatibleSubType (
    OGRwkbGeometryType ) const [protected], [virtual]
```

Returns whether a geometry of the specified geometry type can be a member of this collection.

## Parameters

<i>eSubType</i>	type of the potential member
-----------------	------------------------------

## Returns

TRUE or FALSE

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiCurve** (p. ??), **OGRMultiPoint** (p. ??), **OGRMultiPolygon** (p. ??), and **OGRMultiSurface** (p. ??).

Referenced by addGeometryDirectly().

## 11.70.4.36 isEmpty()

```
OGRBoolean OGRGeometryCollection::IsEmpty ( ) const [override], [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. ??).

References OGRGeometry::IsEmpty().

Referenced by Equals(), and OGRMultiPoint::exportToWkt().

## 11.70.4.37 operator=()

```
OGRGeometryCollection & OGRGeometryCollection::operator= (
    const OGRGeometryCollection & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

## Since

GDAL 2.1

References addGeometry(), empty(), and OGRGeometry::operator=().

Referenced by OGRMultiSurface::operator=(), OGRMultiPoint::operator=(), and OGRMultiCurve::operator=().

**11.70.4.38 removeGeometry()**

```
OGRERR OGRGeometryCollection::removeGeometry (
    int iGeom,
    int bDelete = TRUE ) [virtual]
```

Remove a geometry from the container.

Removing a geometry will cause the geometry count to drop by one, and all "higher" geometries will shuffle down one in index.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_RemoveGeometry()** (p. ??).

**Parameters**

<i>iGeom</i>	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
<i>bDelete</i>	if TRUE the geometry will be deallocated, otherwise it will not. The default is TRUE as the container is considered to own the geometries in it.

**Returns**

OGRERR\_NONE if successful, or OGRERR\_FAILURE if the index is out of range.

< Failure

< Success

< Success

References OGRERR\_FAILURE, and OGRERR\_NONE.

Referenced by OGRGeometryFactory::forceToLineString(), and OGRGeometryFactory::forceToMultiPolygon().

**11.70.4.39 segmentize()**

```
void OGRGeometryCollection::segmentize (
    double dfMaxLength ) [override], [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**

<i>dfMaxLength</i>	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 consecutive vertices is lower than `dfMaxLength`.

#### Parameters

<code>dfMaxLength</code>	maximum length between 2 consecutive vertices.
--------------------------	--

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

References `OGRGeometry::segmentize()`.

#### 11.70.4.40 `set3D()`

```
void OGRGeometryCollection::set3D (
    OGRBoolean bIs3D ) [override], [virtual]
```

Add or remove the Z coordinate dimension.

This method adds or removes the explicit Z coordinate dimension. Removing the Z coordinate dimension of a geometry will remove any existing Z values. Adding the Z dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

#### Parameters

<code>bIs3D</code>	Should the geometry have a Z dimension, either TRUE or FALSE.
--------------------	---

#### Since

GDAL 2.1

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

References `OGRGeometry::set3D()`.

#### 11.70.4.41 `setCoordinateDimension()`

```
void OGRGeometryCollection::setCoordinateDimension (
    int nNewDimension ) [override], [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, a compound curve, a polygon, etc. will affect the children geometries. This will also remove the M dimension if present before this call.

**Deprecated** use `set3D()` (p. ??) or `setMeasured()` (p. ??).

## Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

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

References OGRGeometry::setCoordinateDimension().

11.70.4.42 **setMeasured()**

```
void OGRGeometryCollection::setMeasured (
    OGRBoolean bIsMeasured ) [override], [virtual]
```

Add or remove the M coordinate dimension.

This method adds or removes the explicit M coordinate dimension. Removing the M coordinate dimension of a geometry will remove any existing M values. Adding the M dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

## Parameters

<i>bIsMeasured</i>	Should the geometry have a M dimension, either TRUE or FALSE.
--------------------	---

## Since

GDAL 2.1

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

References OGRGeometry::setMeasured().

11.70.4.43 **swapXY()**

```
void OGRGeometryCollection::swapXY ( ) [override], [virtual]
```

Swap x and y coordinates.

## Since

OGR 1.8.0

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

References OGRGeometry::swapXY().



## 11.70.4.44 transform()

```
OGRERR OGRGeometryCollection::transform (
    OGRCoordinateTransformation * poCT ) [override], [virtual]
```

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGR↔SpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

## Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

## Returns

OGRERR\_NONE on success or an error code.

< Success

< Failure

< Success

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

References assignSpatialReference(), CPLDebug(), OGRCoordinateTransformation::GetTargetCS(), OGRERR\_↔FAILURE, OGRERR\_NONE, and OGRGeometry::transform().

## 11.70.4.45 WkbSize()

```
int OGRGeometryCollection::WkbSize ( ) const [override], [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
- ogrgeometrycollection.cpp

## 11.71 OGRGeometryFactory Class Reference

```
#include <ogr_geometry.h>
```

### Static Public Member Functions

- static **OGRERR** **createFromWkb** (const void \*, **OGRSpatialReference** \*, **OGRGeometry** \*\*, int=-1, **OGRWkbVariant**= **wkbVariantOldGcc**)  
*Create a geometry object of the appropriate type from its well known binary representation.*
- static **OGRERR** **createFromWkb** (const void \*pabyData, **OGRSpatialReference** \*, **OGRGeometry** \*\*, int nSize, **OGRWkbVariant** eVariant, int &nBytesConsumedOut)  
*Create a geometry object of the appropriate type from its well known binary representation.*
- static **OGRERR** **createFromWkt** (const char \*, **OGRSpatialReference** \*, **OGRGeometry** \*\*)  
*Create a geometry object of the appropriate type from its well known text representation.*
- static **OGRERR** **createFromWkt** (const char \*\*, **OGRSpatialReference** \*, **OGRGeometry** \*\*)  
*Create a geometry object of the appropriate type from its well known text representation.*
- static **OGRERR** **createFromWkt** (char \*\*pszInput, **OGRSpatialReference** \*poSRS, **OGRGeometry** \*\*poGeom)  
*Create a geometry object of the appropriate type from its well known text representation.*
- static **OGRERR** **createFromFgf** (const void \*, **OGRSpatialReference** \*, **OGRGeometry** \*\*, int=-1, int \*p=NULLPTR)  
*Create a geometry object of the appropriate type from its FGF (FDO Geometry Format) binary representation.*
- static **OGRGeometry** \* **createFromGML** (const char \*)  
*Create geometry from GML.*
- static **OGRGeometry** \* **createFromGEOS** ( **GEOSContextHandle\_t** hGEOSCtx, **GEOSGeom**)
- static **OGRGeometry** \* **createFromGeoJson** (const char \*)  
*Create geometry from GeoJson fragment.*
- static **OGRGeometry** \* **createFromGeoJson** (const **CPLJSONObject** &oJSONObject)  
*Create geometry from GeoJson fragment.*
- 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 multiline string.*
- static **OGRGeometry** \* **forceTo** ( **OGRGeometry** \*poGeom, **OGRWkbGeometryType** eTargetType, const char \*const \*pszOptions=NULLPTR)  
*Convert to another geometry type.*
- static **OGRGeometry** \* **organizePolygons** ( **OGRGeometry** \*\*papoPolygons, int nPolygonCount, int \*pbResultValidGeometry, const char \*\*pszOptions=NULLPTR)  
*Organize polygons based on geometries.*
- static bool **haveGEOS** ()  
*Test if GEOS enabled.*

- static **OGRGeometry \* transformWithOptions** (const **OGRGeometry** \*poSrcGeom, **OGRCoordinateTransformation** \*poCT, char \*\*papszOptions)
- static **OGRGeometry \* approximateArcAngles** (double dfX, double dfY, double dfZ, double dfPrimaryRadius, double dfSecondaryAxis, double dfRotation, double dfStartAngle, double dfEndAngle, double dfMaxAngleStepSizeDegrees)
- static int **GetCurveParameters** (double x0, double y0, double x1, double y1, double x2, double y2, double &R, double &cx, double &cy, double &alpha0, double &alpha1, double &alpha2)  
*Returns the parameter of an arc circle.*
- 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=nullptr)  
*Converts an arc circle into an approximate line string.*
- static **OGRCurve \* curveFromLineString** (const **OGRLineString** \*poLS, const char \*const \*papszOptions=nullptr)  
*Try to convert a linestring approximating curves into a curve.*

### 11.71.1 Detailed Description

Create geometry objects from well known text/binary.

### 11.71.2 Member Function Documentation

#### 11.71.2.1 approximateArcAngles()

```
OGRGeometry * OGRGeometryFactory::approximateArcAngles (
    double dfCenterX,
    double dfCenterY,
    double dfZ,
    double dfPrimaryRadius,
    double dfSecondaryRadius,
    double dfRotation,
    double dfStartAngle,
    double dfEndAngle,
    double dfMaxAngleStepSizeDegrees ) [static]
```

Stroke arc to linestring.

Stroke an arc of a circle to a linestring based on a center point, radius, start angle and end angle, all angles in degrees.

If the dfMaxAngleStepSizeDegrees is zero, then a default value will be used. This is currently 4 degrees unless the user has overridden the value with the OGR\_ARC\_STEPSIZE configuration variable.

See also

**CPLSetConfigOption()** (p. ??)

## Parameters

<i>dfCenterX</i>	center X
<i>dfCenterY</i>	center Y
<i>dfZ</i>	center Z
<i>dfPrimaryRadius</i>	X radius of ellipse.
<i>dfSecondaryRadius</i>	Y radius of ellipse.
<i>dfRotation</i>	rotation of the ellipse clockwise.
<i>dfStartAngle</i>	angle to first point on arc (clockwise of X-positive)
<i>dfEndAngle</i>	angle to last point on arc (clockwise of X-positive)
<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.

## Returns

**OGRLineString** (p. ??) geometry representing an approximation of the arc.

## Since

OGR 1.8.0

References **OGRSimpleCurve::getPoint()**, **M\_PI**, and **OGRSimpleCurve::setPoint()**.

Referenced by **OGR\_G\_ApproximateArcAngles()**.

## 11.71.2.2 createFromFgf()

```
OGRERR OGRGeometryFactory::createFromFgf (
    const void * pabyData,
    OGRSpatialReference * poSR,
    OGRGeometry ** ppoReturn,
    int nBytes = -1,
    int * pnBytesConsumed = nullptr ) [static]
```

Create a geometry object of the appropriate type from its 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()** (p. ??) is the same as this method.

## Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure, but NULL might be a valid return for a NULL shape.
<i>nBytes</i>	the number of bytes available in pabyData.
<i>pnBytesConsumed</i>	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.

Referenced by OGR\_G\_CreateFromFgf().

**11.71.2.3 createFromGeoJson()** [1/2]

```
OGRGeometry * OGRGeometryFactory::createFromGeoJson (
    const char * pszJsonString ) [static]
```

Create geometry from GeoJson fragment.

**Parameters**

<i>pszJsonString</i>	The GeoJSON fragment for the geometry.
----------------------	--

**Returns**

a geometry on success, or NULL on error.

**Since**

GDAL 2.3

References CPLJSONDocument::GetRoot(), and CPLJSONDocument::LoadMemory().

**11.71.2.4 createFromGeoJson()** [2/2]

```
OGRGeometry * OGRGeometryFactory::createFromGeoJson (
    const CPLJSONObject & oJsonObject ) [static]
```

Create geometry from GeoJson fragment.

**Parameters**

<i>oJsonObject</i>	The JSONObject class describes the GeoJSON geometry.
--------------------	--

**Returns**

a geometry on success, or NULL on error.

Since

GDAL 2.3

References CPLJSONObject::IsValid().

#### 11.71.2.5 createFromGEOS()

```
OGRGeometry * OGRGeometryFactory::createFromGEOS (
    GEOSContextHandle_t hGEOSCtxt,
    GEOSGeom ) [static]
```

Builds a OGRGeometry\* from a GEOSGeom.

Parameters

<i>hGEOSCtxt</i>	GEOS context
<i>geosGeom</i>	GEOS geometry

Returns

a OGRGeometry\*

References CPLError().

#### 11.71.2.6 createFromGML()

```
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 **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, LineStringSegment, Arc, Circle, CompositeSurface, OrientableSurface, Solid, Tin, TriangulatedSurface.

Arc and Circle elements are stroked to linestring, by using a 4 degrees step, unless the user has overridden the value with the OGR\_ARC\_STEPSIZE configuration variable.

The C function **OGR\_G\_CreateFromGML()** (p. ??) is the same as this method.

## Parameters

<i>pszData</i>	The GML fragment for the geometry.
----------------	------------------------------------

## Returns

a geometry on success, or NULL on error.

References OGRGeometry::FromHandle(), and OGR\_G\_CreateFromGML().

## 11.71.2.7 createFromWkb() [1/2]

```
OGRERR OGRGeometryFactory::createFromWkb (
    const void * pabyData,
    OGRSpatialReference * poSR,
    OGRGeometry ** ppoReturn,
    int nBytes = -1,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) [static]
```

Create a geometry object of the appropriate type from its well known binary representation.

Note that if *nBytes* is passed as zero, no checking can be done on whether the *pabyData* is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **OGRGeometry::WkbSize()** (p. ??) on the returned geometry to establish the number of bytes it required in WKB format.

Also note that this is a static method, and that there is no need to instantiate an **OGRGeometryFactory** (p. ??) object.

The C function **OGR\_G\_CreateFromWkb()** (p. ??) is the same as this method.

## Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure. If not NULL, *ppoReturn should be freed with <b>OGRGeometryFactory::destroyGeometry()</b> (p. ??) after use.
<i>nBytes</i>	the number of bytes available in <i>pabyData</i> , or -1 if it isn't known.
<i>eWkbVariant</i>	WKB variant.

## 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.

Referenced by OGRPolyhedralSurface::importFromWkb(), and OGR\_G\_CreateFromWkb().

11.71.2.8 `createFromWkb()` [2/2]

```

OGRERR OGRGeometryFactory::createFromWkb (
    const void * pabyData,
    OGRSpatialReference * poSR,
    OGRGeometry ** ppoReturn,
    int nBytes,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [static]

```

Create a geometry object of the appropriate type from its well known binary representation.

Note that if `nBytes` is passed as zero, no checking can be done on whether the `pabyData` is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **`OGRGeometry::WkbSize()`** (p. ??) on the returned geometry to establish the number of bytes it required in WKB format.

Also note that this is a static method, and that there is no need to instantiate an **`OGRGeometryFactory`** (p. ??) object.

The C function **`OGR_G_CreateFromWkb()`** (p. ??) is the same as this method.

## Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure. If not NULL, *ppoReturn should be freed with <b><code>OGRGeometryFactory::destroyGeometry()</code></b> (p. ??) after use.
<i>nBytes</i>	the number of bytes available in <code>pabyData</code> , or -1 if it isn't known.
<i>eWkbVariant</i>	WKB variant.
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

## Since

GDAL 2.3

< Not enough data to deserialize

< Corrupt data

< Success



< Unsupported geometry type

< Success

< Success

References OGRGeometry::assignSpatialReference(), CPLDebug(), CPLGetConfigOption(), CPLTestBool(), createGeometry(), OGRGeometry::getLinearGeometry(), OGRGeometry::hasCurveGeometry(), OGRGeometry::importFromWkb(), OGRERR\_CORRUPT\_DATA, OGRERR\_NONE, OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, wkbNDR, wkbUnknown, and wkbXDR.

#### 11.71.2.9 createFromWkt() [1/3]

```
OGRERR OGRGeometryFactory::createFromWkt (
    const char * pszData,
    OGRSpatialReference * poSR,
    OGRGeometry ** ppoReturn ) [static]
```

Create a geometry object of the appropriate type from its well known text representation.

The C function **OGR\_G\_CreateFromWkt()** (p. ??) is the same as this method.

##### Parameters

<i>pszData</i>	input zero terminated string containing well known text representation of the geometry to be created.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL if the method fails. If not NULL, *ppoReturn should be freed with <b>OGRGeometryFactory::destroyGeometry()</b> (p. ??) after use.

##### Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

##### Since

GDAL 2.3

Referenced by OGRMultiSurface::importFromWkt(), and OGR\_G\_CreateFromWkt().

#### 11.71.2.10 createFromWkt() [2/3]

```
OGRERR OGRGeometryFactory::createFromWkt (
    const char ** ppszData,
    OGRSpatialReference * poSR,
    OGRGeometry ** ppoReturn ) [static]
```

Create a geometry object of the appropriate type from its well known text representation.

The C function **OGR\_G\_CreateFromWkt()** (p. ??) is the same as this method.

## Parameters

<i>ppszData</i>	input zero terminated string containing well known text representation of the geometry to be created. The pointer is updated to point just beyond that last character consumed.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL if the method fails. If not NULL, *ppoReturn should be freed with <b>OGRGeometryFactory::destroyGeometry()</b> (p. ??) after use.

## Example:

```
const char* wkt= "POINT(0 0)";

// cast because OGR_G_CreateFromWkt will move the pointer
char* pszWkt = (char*) wkt;
OGRSpatialReferenceH ref = OSRNewSpatialReference(NULL);
OGRGeometryH new_geom;
OGRERR err = OGR_G_CreateFromWkt(&pszWkt, ref, &new_geom);
```

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

< Corrupt data

< Unsupported geometry type

< Success

References OGRGeometry::assignSpatialReference(), CPLGetConfigOption(), CPLTestBool(), OGRGeometry::getLinearGeometry(), OGRGeometry::hasCurveGeometry(), OGRGeometry::importFromWkt(), OGRERR\_CORRUPT\_DATA, OGRERR\_NONE, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, and STARTS\_WITH\_CI.

## 11.71.2.11 createFromWkt() [3/3]

```
static OGRERR OGRGeometryFactory::createFromWkt (
    char ** ppszInput,
    OGRSpatialReference * poSR,
    OGRGeometry ** ppoGeom ) [inline], [static]
```

Deprecated.

**Deprecated** in GDAL 2.3

## 11.71.2.12 createGeometry()

```
OGRGeometry * OGRGeometryFactory::createGeometry (
    OGRwkbGeometryType eGeometryType ) [static]
```

Create an empty geometry of desired type.

This is equivalent to allocating the desired geometry with new, but the allocation is guaranteed to take place in the context of the GDAL/OGR heap.

This method is the same as the C function **OGR\_G\_CreateGeometry()** (p. ??).

## Parameters

<i>eGeometryType</i>	the type code of the geometry class to be instantiated.
----------------------	---

## Returns

the newly create geometry or NULL on failure. Should be freed with **OGRGeometryFactory::destroyGeometry()** (p. ??) after use.

References `wkbCircularString`, `wkbCompoundCurve`, `wkbCurvePolygon`, `wkbFlatten`, `wkbGeometryCollection`, `wkbLinearRing`, `wkbLineString`, `wkbMultiCurve`, `wkbMultiLineString`, `wkbMultiPoint`, `wkbMultiPolygon`, `wkbMultiSurface`, `wkbPoint`, `wkbPolygon`, `wkbPolyhedralSurface`, `wkbTIN`, and `wkbTriangle`.

Referenced by `OGRSimpleCurve::clone()`, `OGRCurvePolygon::clone()`, `OGRGeometryCollection::clone()`, `OGRPolyhedralSurface::clone()`, `createFromWkb()`, `forceTo()`, `OGRGeometryCollection::getCurveGeometry()`, `OGRGeometryCollection::getLinearGeometry()`, `OGRPolyhedralSurface::importFromWkt()`, `OGR_G_CreateGeometry()`, and `transformWithOptions()`.

## 11.71.2.13 curveFromLineString()

```
OGRCurve * OGRGeometryFactory::curveFromLineString (
    const OGRLineString * poLS,
    const char *const * papszOptions = nullptr ) [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

<i>poLS</i>	handle to the geometry to convert.
<i>papszOptions</i>	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::assignSpatialReference()`, `OGRSimpleCurve::clone()`, `OGRCompoundCurve::getNumCurves()`, `OGRSimpleCurve::getNumPoints()`, `OGRSimpleCurve::getPoint()`, `OGRGeometry::getSpatialReference()`, `OGRPoint::getX()`, `OGRSimpleCurve::getX()`, `OGRPoint::getY()`, `OGRSimpleCurve::getY()`, `OGRCompoundCurve::stealCurve()`, and `OGRGeometry::toCurve()`.

Referenced by `OGRLineString::getCurveGeometry()`.

## 11.71.2.14 curveToLineString()

```

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 = nullptr ) [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

<i>x0</i>	x of first point
<i>y0</i>	y of first point
<i>z0</i>	z of first point
<i>x1</i>	x of intermediate point
<i>y1</i>	y of intermediate point
<i>z1</i>	z of intermediate point
<i>x2</i>	x of final point
<i>y2</i>	y of final point
<i>z2</i>	z of final point
<i>bHasZ</i>	TRUE if z must be taken into account
<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings or NULL. Recognized options: <ul style="list-style-type: none"> <li>• <b>ADD_INTERMEDIATE_POINT=STEALTH/YES/NO</b> (Default to STEALTH). Determine if and how the intermediate point must be output in the linestring. If set to STEALTH, no explicit intermediate point is added but its properties are encoded in low significant bits of intermediate points and <b>OGRGeometryFactory::curveFromLineString()</b> (p. ??) can decode them. This is the best compromise for round-tripping in OGR and better results with PostGIS <b>ST_LineToCurve()</b> If set to YES, the intermediate point is explicitly added to the linestring. If set to NO, the intermediate point is not explicitly added.</li> </ul>

## Returns

the converted geometry (ownership to caller).

Since

GDAL 2.0

References OGRSimpleCurve::addPoint(), CPLDebug(), CPLError(), CPLParseNameValue(), EQUAL, GetCurveParameters(), M\_PI, and OGRSimpleCurve::reversePoints().

Referenced by OGRCircularString::CurveToLine().

#### 11.71.2.15 destroyGeometry()

```
void OGRGeometryFactory::destroyGeometry (
    OGRGeometry * poGeom ) [static]
```

Destroy geometry object.

Equivalent to invoking delete on a geometry, but it guaranteed to take place within the context of the GDAL/OGR heap.

This method is the same as the C function **OGR\_G\_DestroyGeometry()** (p. ??).

Parameters

<i>poGeom</i>	the geometry to deallocate.
---------------	-----------------------------

Referenced by OGR\_G\_DestroyGeometry().

#### 11.71.2.16 forceTo()

```
OGRGeometry * OGRGeometryFactory::forceTo (
    OGRGeometry * poGeom,
    OGRwkbGeometryType eTargetType,
    const char *const * papszOptions = nullptr ) [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 "wrapping" 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

<i>poGeom</i>	the input geometry - ownership is passed to the method.
<i>eTargetType</i>	target output geometry type.
<i>papszOptions</i>	options as a null-terminated list of strings or NULL.

**Returns**

new geometry.

**Since**

GDAL 2.0

< Success

< Success

< Success

< Success

< Success

References OGRPolyhedralSurface::addGeometry(), OGRGeometryCollection::addGeometryDirectly(), OGRPolyhedralSurface::addGeometryDirectly(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::assignSpatialReference(), OGRPolyhedralSurface::assignSpatialReference(), OGRGeometryCollection::CastToGeometryCollection(), OGRCurve::CastToLineString(), OGRCurvePolygon::CastToPolygon(), OGRTriangulatedSurface::CastToPolyhedralSurface(), createGeometry(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRPolyhedralSurface::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRPolyhedralSurface::getNumGeometries(), OGRCurvePolygon::getNumInteriorRings(), OGRSimpleCurve::getNumPoints(), OGRGeometry::getSpatialReference(), OGRGeometry::IsEmpty(), OGR\_GT\_GetCollection(), OGR\_GT\_IsSubClassOf(), OGRERR\_NONE, OGRGeometry::toCurve(), OGRGeometry::toGeometryCollection(), OGRGeometry::toMultiPolygon(), OGRGeometry::toPolygon(), OGRGeometry::toPolyhedralSurface(), OGRGeometry::toTriangulatedSurface(), wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, wkbPolyhedralSurface, wkbTIN, wkbTriangle, and wkbUnknown.

Referenced by OGR\_F\_GetGeometryRef(), OGR\_F\_GetGeomFieldRef(), and OGR\_G\_ForceTo().

**11.71.2.17 forceToLineString()**

```
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 curvopolygons 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**

<i>poGeom</i>	the input geometry - ownership is passed to the method.
<i>bOnlyInOrder</i>	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(), OGRCurve::CurveToLine(), 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(), OGRGeometry::toCurve(), OGRGeometry::toCurvePolygon(), OGRGeometry::toGeometryCollection(), OGRGeometry::toLineString(), wkbCircularString, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, and wkbPolygon.

Referenced by OGR\_G\_ForceToLineString().

## 11.71.2.18 forceToMultiLineString()

```
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(), OGRGeometryCollection::assignSpatialReference(), OGRMultiCurve::CastToMultiLineString(), CPLAssert, OGRCurvePolygon::CurvePolyToPoly(), OGRCurve::CurveToLine(), forceToMultiPolygon(), OGRPolygon::getExteriorRing(), OGRGeometry::getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometry::getLinearGeometry(), OGRGeometryCollection::getLinearGeometry(), OGRCurvePolygon::getNumInteriorRings(), OGRSimpleCurve::getNumPoints(), OGRGeometry::getSpatialReference(), OGRGeometry::hasCurveGeometry(), OGRMultiCurve::hasCurveGeometry(), OGR\_GT\_IsSubClassOf(), OGRGeometry::toCurve(), OGRGeometry::toCurvePolygon(), OGRGeometry::toGeometryCollection(), OGRGeometry::toMultiCurve(), OGRGeometry::toMultiLineString(), OGRGeometry::toMultiPolygon(), OGRGeometry::toPolygon(), wkbCircularString, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, and wkbPolyhedralSurface.

Referenced by OGR\_G\_ForceToMultiLineString().

## 11.71.2.19 forceToMultiPoint()

```
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(), OGRGeometryCollection::assignSpatialReference(), OGRGeometry::getGeometryType(), OGRGeometry::getSpatialReference(), OGRGeometry::toGeometryCollection(), wkbFlatten, wkbGeometryCollection, wkbMultiPoint, and wkbPoint.

Referenced by OGR\_G\_ForceToMultiPoint().

## 11.71.2.20 forceToMultiPolygon()

```
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(), OGRGeometryCollection::assignSpatialReference(), OGRMultiSurface::CastToMultiPolygon(), OGRPolyhedralSurface::CastToMultiPolygon(), OGRCurvePolygon::CurvePolyToPoly(), forceToPolygon(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getLinearGeometry(), OGRGeometryCollection::getNumGeometries(), OGRGeometry::getSpatialReference(), OGRGeometry::hasCurveGeometry(), OGRMultiSurface::hasCurveGeometry(), OGR\_GT\_IsSubClassOf(), OGRGeometryCollection::removeGeometry(), OGRGeometry::toCurvePolygon(), OGRGeometry::toGeometryCollection(), OGRGeometry::toMultiPolygon(), OGRGeometry::toMultiSurface(), OGRGeometry::toPolyhedralSurface(), wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, wkbPolyhedralSurface, wkbTIN, and wkbTriangle.

Referenced by forceToMultiLineString(), and OGR\_G\_ForceToMultiPolygon().

## 11.71.2.21 forceToPolygon()

```
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).

Note: the resulting polygon may break the Simple Features rules for polygons, for example when converting from a multi-part multipolygon.



## Parameters

<i>poGeom</i>	the input geometry - ownership is passed to the method.
---------------	---

## Returns

new geometry.

References OGRCurvePolygon::addRingDirectly(), OGRCurvePolygon::assignSpatialReference(), OGRCurve::CastToLinearRing(), OGRGeometry::clone(), OGRCurvePolygon::CurvePolyToPoly(), OGRCurve::CurveToLine(), OGRCurve::get\_IsClosed(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRPolyhedralSurface::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getLinearGeometry(), OGRGeometryCollection::getNumGeometries(), OGRPolyhedralSurface::getNumGeometries(), OGRCurvePolygon::getNumInteriorRings(), OGRCurve::getNumPoints(), OGRGeometry::getSpatialReference(), OGRGeometry::hasCurveGeometry(), OGR\_GT\_IsCurve(), OGR\_GT\_IsSubClassOf(), OGRPolygon::stealExteriorRing(), OGRPolygon::stealInteriorRing(), OGRGeometry::toCurve(), OGRGeometry::toCurvePolygon(), OGRGeometry::toGeometryCollection(), OGRGeometry::toPolygon(), OGRGeometry::toPolyhedralSurface(), OGRGeometry::toSurface(), wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, and wkbPolyhedralSurface.

Referenced by forceToMultiPolygon(), and OGR\_G\_ForceToPolygon().

## 11.71.2.22 GetCurveParameters()

```
int OGRGeometryFactory::GetCurveParameters (
    double x0,
    double y0,
    double x1,
    double y1,
    double x2,
    double y2,
    double & R,
    double & cx,
    double & cy,
    double & alpha0,
    double & alpha1,
    double & alpha2 ) [static]
```

Returns the parameter of an arc circle.

Angles are return in radians, with trigonometric convention (counter clock wise)

## Parameters

<i>x0</i>	x of first point
<i>y0</i>	y of first point
<i>x1</i>	x of intermediate point
<i>y1</i>	y of intermediate point
<i>x2</i>	x of final point
<i>y2</i>	y of final point
<i>R</i>	radius (output)
<i>cx</i>	x of arc center (output)
<i>cy</i>	y of arc center (output)
<i>alpha0</i>	angle between center and first point, in radians (output)
<i>alpha1</i>	angle between center and intermediate point, in radians (output)
<i>alpha2</i>	angle between center and final point, in radians (output)

**Returns**

TRUE if the points are not aligned and define an arc circle.

**Since**

GDAL 2.0

Referenced by `curveToLineString()`, `OGRCircularString::get_Length()`, `OGRCircularString::segmentize()`, and `OGRCircularString::Value()`.

**11.71.2.23 haveGEOS()**

```
bool 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()`.

**11.71.2.24 organizePolygons()**

```
OGRGeometry * OGRGeometryFactory::organizePolygons (
    OGRGeometry ** papoPolygons,
    int nPolygonCount,
    int * pbIsValidGeometry,
    const char ** papszOptions = nullptr ) [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 `OGRPolygon`/`OGRCurvePolygon` 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

<i>papoPolygons</i>	array of geometry pointers - should all be OGRPolygons. Ownership of the geometries is passed, but not of the array itself.
<i>nPolygonCount</i>	number of items in papoPolygons
<i>pblsValidGeometry</i>	value will be set TRUE if result is valid or FALSE otherwise.
<i>papszOptions</i>	a list of strings for passing options

## Returns

a single resulting geometry (either **OGRPolygon** (p. ??), **OGRCurvePolygon** (p. ??), **OGRMultiPolygon** (p. ??), **OGRMultiSurface** (p. ??) or **OGRGeometryCollection** (p. ??)). Returns a POLYGON EMPTY in the case of nPolygonCount being 0.

References CPLDebug(), CPLError(), CPLGetConfigOption(), CPLTestBool(), CSLFetchNameValue(), EQUAL, and haveGEOS().

## 11.71.2.25 transformWithOptions()

```
OGRGeometry * OGRGeometryFactory::transformWithOptions (
    const OGRGeometry * poSrcGeom,
    OGRCoordinateTransformation * poCT,
    char ** papszOptions ) [static]
```

Transform a geometry.

## Parameters

<i>poSrcGeom</i>	source geometry
<i>poCT</i>	coordinate transformation object.
<i>papszOptions</i>	options. Including WRAPDATELINE=YES.

## Returns

(new) transformed geometry.

< Success

References OGRGeometry::clone(), CPLAtofM(), CPLTestBool(), createGeometry(), CSLFetchNameValueDef(), OGRGeometry::getEnvelope(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRCoordinateTransformation::GetSourceCS(), OGRCoordinateTransformation::GetTargetCS(), OGRPoint::getX(), OGRSpatialReference::IsSame(), OGRCreateCoordinateTransformation(), OGRERR\_NONE, OGRSpatialReference::SetWellKnownGeogCS(), OGRPoint::setX(), OGRGeometry::toGeometryCollection(), OGRGeometry::toPoint(), OGRGeometry::transform(), wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiLineString, wkbMultiPolygon, wkbPoint, and wkbPolygon.

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

- **ogr\_geometry.h**
- **ogrgeometryfactory.cpp**

## 11.72 OGRGeomFieldDefn Class Reference

```
#include <ogr_feature.h>
```

### Public Member Functions

- **OGRGeomFieldDefn** (const char \*pszNameIn, **OGRwkbGeometryType** eGeomTypeIn)  
*Constructor.*
- **OGRGeomFieldDefn** (const **OGRGeomFieldDefn** \*)  
*Constructor.*
- void **SetName** (const char \*)  
*Reset the name of this field.*
- const char \* **GetNameRef** () const  
*Fetch name of this field.*
- **OGRwkbGeometryType** **GetType** () const  
*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** () const  
*Fetch spatial reference system of this field.*
- void **SetSpatialRef** ( **OGRSpatialReference** \*poSRSIn)  
*Set the spatial reference of this field.*
- int **IsIgnored** () const  
*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** (const **OGRGeomFieldDefn** \*) const  
*Test if the geometry field definition is identical to the other one.*

### Static Public Member Functions

- static **OGRGeomFieldDefnH** **ToHandle** ( **OGRGeomFieldDefn** \*poGeomFieldDefn)
- static **OGRGeomFieldDefn** \* **FromHandle** ( **OGRGeomFieldDefnH** hGeomFieldDefn)

#### 11.72.1 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

## 11.72.2 Constructor & Destructor Documentation

### 11.72.2.1 OGRGeomFieldDefn() [1/2]

```
OGRGeomFieldDefn::OGRGeomFieldDefn (
    const char * pszNameIn,
    OGRwkbGeometryType eGeomTypeIn )
```

Constructor.

#### Parameters

<i>pszNameIn</i>	the name of the new field.
<i>eGeomTypeIn</i>	the type of the new field.

Since

GDAL 1.11

### 11.72.2.2 OGRGeomFieldDefn() [2/2]

```
OGRGeomFieldDefn::OGRGeomFieldDefn (
    const OGRGeomFieldDefn * poPrototype ) [explicit]
```

Constructor.

Create by cloning an existing geometry field definition.

#### Parameters

<i>poPrototype</i>	the geometry field definition to clone.
--------------------	---

Since

GDAL 1.11

References [GetNameRef\(\)](#), [GetSpatialRef\(\)](#), [GetType\(\)](#), [IsNullable\(\)](#), [SetNullable\(\)](#), and [SetSpatialRef\(\)](#).

## 11.72.3 Member Function Documentation

### 11.72.3.1 FromHandle()

```
static OGRGeomFieldDefn* OGRGeomFieldDefn::FromHandle (
    OGRGeomFieldDefnH hGeomFieldDefn ) [inline], [static]
```

Convert a OGRGeomFieldDefnH to a OGRGeomFieldDefn\*.

Since

GDAL 2.3

Referenced by OGR\_FD\_AddGeomFieldDefn(), OGR\_GFld\_Destroy(), OGR\_GFld\_GetNameRef(), OGR\_GFld\_GetSpatialRef(), OGR\_GFld\_GetType(), OGR\_GFld\_IsIgnored(), OGR\_GFld\_IsNullable(), OGR\_GFld\_SetIgnored(), OGR\_GFld\_SetName(), OGR\_GFld\_SetNullable(), OGR\_GFld\_SetSpatialRef(), OGR\_GFld\_SetType(), and OGR\_L\_CreateGeomField().

### 11.72.3.2 GetNameRef()

```
const char * OGRGeomFieldDefn::GetNameRef ( ) const [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(), OGRFeatureDefn::GetGeomFieldIndex(), IsSame(), OGR\_GFld\_GetNameRef(), OGRGeomFieldDefn(), and OGRFeature::SetFrom().

### 11.72.3.3 GetSpatialRef()

```
OGRSpatialReference * OGRGeomFieldDefn::GetSpatialRef ( ) const [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 OGRLayer::GetSpatialRef(), IsSame(), OGR\_GFld\_GetSpatialRef(), and OGRGeomFieldDefn().

## 11.72.3.4 GetType()

```
OGRwkbGeometryType OGRGeomFieldDefn::GetType ( ) const [inline]
```

Fetch geometry type of this field.

This method is the same as the C function **OGR\_GFId\_GetType()** (p. ??).

## Returns

field geometry type.

## Since

GDAL 1.11

Referenced by OGRFeatureDefn::GetGeomType(), IsSame(), OGR\_GFId\_GetType(), and OGRGeomFieldDefn().

## 11.72.3.5 IsIgnored()

```
int OGRGeomFieldDefn::IsIgnored ( ) const [inline]
```

Return whether this field should be omitted when fetching features.

This method is the same as the C function **OGR\_GFId\_IsIgnored()** (p. ??).

## Returns

ignore state

## Since

GDAL 1.11

Referenced by OGRFeatureDefn::IsGeometryIgnored(), and OGR\_GFId\_IsIgnored().

## 11.72.3.6 IsNullable()

```
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\_GFId\_IsNullable()** (p. ??).

## Returns

TRUE if the field is authorized to be null.

## Since

GDAL 2.0

Referenced by IsSame(), OGR\_GFId\_IsNullable(), OGRGeomFieldDefn(), and OGRFeature::Validate().

### 11.72.3.7 IsSame()

```
int OGRGeomFieldDefn::IsSame (
    const OGRGeomFieldDefn * poOtherFieldDefn ) const
```

Test if the geometry field definition is identical to the other one.

#### Parameters

<i>poOtherFieldDefn</i>	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().

### 11.72.3.8 SetIgnored()

```
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\_GFid\_SetIgnored()** (p. ??).

#### Parameters

<i>ignore</i>	ignore state
---------------	--------------

#### Since

GDAL 1.11

Referenced by OGR\_GFid\_SetIgnored(), OGRFeatureDefn::SetGeometryIgnored(), and OGRLayer::SetIgnoredFields().



## 11.72.3.9 SetName()

```
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

<i>pszNameIn</i>	the new name to apply.
------------------	------------------------

## Since

GDAL 1.11

References CPLFree, and CPLStrdup().

Referenced by OGR\_GFld\_SetName().

## 11.72.3.10 SetNullable()

```
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

<i>bNullableIn</i>	FALSE if the field must have a not-null constraint.
--------------------	---

## Since

GDAL 2.0

Referenced by OGR\_GFld\_SetNullable(), and OGRGeomFieldDefn().

### 11.72.3.11 SetSpatialRef()

```
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

<i>poSRSIn</i>	the new SRS to apply.
----------------	-----------------------

#### Since

GDAL 1.11

References OGRSpatialReference::Reference().

Referenced by OGRGeomFieldDefn().

### 11.72.3.12 SetType()

```
void OGRGeomFieldDefn::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. ??).

This method is the same as the C function **OGR\_GFld\_SetType()** (p. ??).

#### Parameters

<i>eTypeIn</i>	the new field geometry type.
----------------	------------------------------

#### Since

GDAL 1.11

Referenced by OGR\_GFld\_SetType(), and OGRFeatureDefn::SetGeomType().

## 11.72.3.13 ToHandle()

```
static OGRGeomFieldDefnH OGRGeomFieldDefn::ToHandle (
    OGRGeomFieldDefn * poGeomFieldDefn ) [inline], [static]
```

Convert a OGRGeomFieldDefn\* to a OGRGeomFieldDefnH.

Since

GDAL 2.3

Referenced by OGR\_F\_GetGeomFieldDefnRef(), OGR\_FD\_GetGeomFieldDefn(), and OGR\_GFld\_Create().

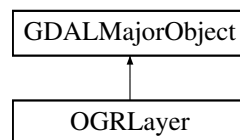
The documentation for this class was generated from the following files:

- **ogr\_feature.h**
- ogrgeomfielddefn.cpp

## 11.73 OGRLayer Class Reference

```
#include <ogr_sfrmts.h>
```

Inheritance diagram for OGRLayer:



### Classes

- struct **Private**

### Public Member Functions

- FeatureIterator **begin** ()
- FeatureIterator **end** ()
- virtual **OGRGeometry** \* **GetSpatialFilter** ()  
*This method returns the current spatial filter for this layer.*
- virtual void **SetSpatialFilter** ( **OGRGeometry** \*)  
*Set a new spatial filter.*
- virtual void **SetSpatialFilterRect** (double dfMinX, double dfMinY, double dfMaxX, double dfMaxY)  
*Set a new rectangular spatial filter.*
- virtual void **SetSpatialFilter** (int iGeomField, **OGRGeometry** \*)  
*Set a new spatial filter.*
- virtual void **SetSpatialFilterRect** (int iGeomField, double dfMinX, double dfMinY, double dfMaxX, double dfMaxY)  
*Set a new rectangular spatial filter.*

- virtual **OGRErr SetAttributeFilter** (const char \*)  
*Set a new attribute query.*
- virtual void **ResetReading** ()=0  
*Reset feature reading to start on the first feature.*
- virtual **OGRFeature \* GetNextFeature** () **CPL\_WARN\_UNUSED\_RESULT**  
*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) **CPL\_WARN\_UNUSED\_RESULT**  
*Fetch a feature by its identifier.*
- **OGRErr SetFeature** ( **OGRFeature \*poFeature**) **CPL\_WARN\_UNUSED\_RESULT**  
*Rewrite an existing feature.*
- **OGRErr CreateFeature** ( **OGRFeature \*poFeature**) **CPL\_WARN\_UNUSED\_RESULT**  
*Create and write a new feature within a layer.*
- virtual **OGRErr DeleteFeature** ( **GIntBig** nFID) **CPL\_WARN\_UNUSED\_RESULT**  
*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) **CPL\_WARN\_UNUSED\_RESULT**  
*Fetch the extent of this layer.*
- virtual **OGRErr GetExtent** (int iGeomField, OGREnvelope \*psExtent, int bForce=TRUE) **CPL\_WARN\_UNUSED\_RESULT**  
*Fetch the extent of this layer, on the specified geometry field.*
- 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 OGRErr **StartTransaction** () CPL\_WARN\_UNUSED\_RESULT  
*For datasources which support transactions, StartTransaction creates a transaction.*
- virtual OGRErr **CommitTransaction** () CPL\_WARN\_UNUSED\_RESULT  
*For datasources which support transactions, CommitTransaction commits a transaction.*
- virtual OGRErr **RollbackTransaction** ()  
*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.*
- 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=nullptr, GDALProgressFunc pfnProgress=nullptr, void \*pProgressArg=nullptr)  
*Intersection of two layers.*
- OGRErr **Union** ( OGRLayer \*pLayerMethod, OGRLayer \*pLayerResult, char \*\*papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void \*pProgressArg=nullptr)  
*Union of two layers.*
- OGRErr **SymDifference** ( OGRLayer \*pLayerMethod, OGRLayer \*pLayerResult, char \*\*papszOptions, GDALProgressFunc pfnProgress, void \*pProgressArg)  
*Symmetrical difference of two layers.*
- OGRErr **Identity** ( OGRLayer \*pLayerMethod, OGRLayer \*pLayerResult, char \*\*papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void \*pProgressArg=nullptr)  
*Identify the features of this layer with the ones from the identity layer.*
- OGRErr **Update** ( OGRLayer \*pLayerMethod, OGRLayer \*pLayerResult, char \*\*papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void \*pProgressArg=nullptr)  
*Update this layer with features from the update layer.*
- OGRErr **Clip** ( OGRLayer \*pLayerMethod, OGRLayer \*pLayerResult, char \*\*papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void \*pProgressArg=nullptr)  
*Clip off areas that are not covered by the method layer.*
- OGRErr **Erase** ( OGRLayer \*pLayerMethod, OGRLayer \*pLayerResult, char \*\*papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void \*pProgressArg=nullptr)  
*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.*

### Static Public Member Functions

- static OGRLayerH **ToHandle** ( OGRLayer \*poLayer)
- static OGRLayer \* **FromHandle** ( OGRLayerH hLayer)

## Protected Member Functions

- virtual **OGRERR** **ISetFeature** ( **OGRFeature** \*poFeature) **CPL\_WARN\_UNUSED\_RESULT**  
*Rewrite an existing feature.*
- virtual **OGRERR** **ICreateFeature** ( **OGRFeature** \*poFeature) **CPL\_WARN\_UNUSED\_RESULT**  
*Create and write a new feature within a layer.*

## Friends

- FeatureIterator **begin** ( **OGRLayer** \*poLayer)
- FeatureIterator **end** ( **OGRLayer** \*poLayer)

### 11.73.1 Detailed Description

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

### 11.73.2 Member Function Documentation

#### 11.73.2.1 AlterFieldDefn()

```
OGRERR OGRLayer::AlterFieldDefn (
    int iField,
    OGRFieldDefn * poNewFieldDefn,
    int nFlags ) [virtual]
```

Alter the definition of an existing field on a layer.

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

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

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

This function is the same as the C function **OGR\_L\_AlterFieldDefn()** (p. ??).

#### Parameters

<i>iField</i>	index of the field whose definition must be altered.
<i>poNewFieldDefn</i>	new field definition
<i>nFlags</i>	combination of <b>ALTER_NAME_FLAG</b> , <b>ALTER_TYPE_FLAG</b> , <b>ALTER_WIDTH_PRECISION_FLAG</b> , <b>ALTER_NULLABLE_FLAG</b> and <b>ALTER_DEFAULT_FLAG</b> 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

References CPLError().

Referenced by OGR\_L\_AlterFieldDefn().

**11.73.2.2 begin()**

```
OGRLayer::FeatureIterator OGRLayer::begin ( )
```

Return begin of feature iterator.

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

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

**Since**

GDAL 2.3

**11.73.2.3 Clip()**

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

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

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

**Note**

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

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

The recognized list of options is :

- SKIP\_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

This method is the same as the C function **OGR\_L\_Clip()** (p. ??).

## Parameters

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

## Returns

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

## Note

The first geometry field is always used.

## Since

OGR 1.10

References CPLError(), CPLTestBool(), CSLFetchNameValueDef(), GetFeatureCount(), GetLayerDefn(), OGRGeometryFactory::haveGEOS(), OGRERR\_NONE, and OGRERR\_UNSUPPORTED\_OPERATION.

Referenced by OGR\_L\_Clip().

## 11.73.2.4 CommitTransaction()

```
OGRERR OGRLayer::CommitTransaction ( ) [virtual]
```

For datasources which support transactions, CommitTransaction commits a transaction.

If no transaction is active, or the commit fails, will return OGRERR\_FAILURE. Datasources which do not support transactions will always return OGRERR\_NONE.

This function is the same as the C function **OGR\_L\_CommitTransaction()** (p. ??).

## Returns

OGRERR\_NONE on success.

References OGRERR\_NONE.

Referenced by OGR\_L\_CommitTransaction().

## 11.73.2.5 CreateFeature()

```
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

<i>poFeature</i>	the feature to write to disk.
------------------	-------------------------------

## Returns

OGRERR\_NONE on success.

References ICreateFeature().

Referenced by OGR\_L\_CreateFeature().

## 11.73.2.6 CreateField()

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

Create a new field on a layer.

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

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

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

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

This function is the same as the C function **OGR\_L\_CreateField()** (p. ??).

## Parameters

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

## Returns

OGRERR\_NONE on success.

References CPLError().

Referenced by OGR\_L\_CreateField().

### 11.73.2.7 CreateGeomField()

```
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 existence that were obtained or created with the previous layer definition.

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

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

This function is the same as the C function **OGR\_L\_CreateGeomField()** (p. ??).

#### Parameters

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

#### Returns

OGRERR\_NONE on success.

#### Since

OGR 1.11

References CPLError().

Referenced by OGR\_L\_CreateGeomField().

### 11.73.2.8 DeleteFeature()

```
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

<i>nFID</i>	the feature id to be deleted from the layer
-------------	---

## Returns

OGRERR\_NONE if the operation works, otherwise an appropriate error code (e.g OGRERR\_NON\_EXISTING\_FEATURE if the feature does not exist).

References OGRERR\_UNSUPPORTED\_OPERATION.

Referenced by OGR\_L\_DeleteFeature().

## 11.73.2.9 DeleteField()

```
OGRERR OGRLayer::DeleteField (
    int iField ) [virtual]
```

Delete an existing field on a layer.

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

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

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

This function is the same as the C function **OGR\_L\_DeleteField()** (p. ??).

## Parameters

<i>iField</i>	index of the field to delete.
---------------	-------------------------------

## Returns

OGRERR\_NONE on success.

## Since

OGR 1.9.0

References CPLError().

Referenced by OGR\_L\_DeleteField().

**11.73.2.10 Derefence()**

```
int OGRLayer::Derefence ( )
```

Decrement layer reference count.

This method is the same as the C function `OGR_L_Derefence()`.

**Returns**

the reference count after decrementing.

**11.73.2.11 end()**

```
OGRLayer::FeatureIterator OGRLayer::end ( )
```

Return end of feature iterator.

**11.73.2.12 Erase()**

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

Remove areas that are covered by the method layer.

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

**Note**

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

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

The recognized list of options is :

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

This method is the same as the C function `OGR_L_Erase()` (p. ??).

## Parameters

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

## Returns

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

## Note

The first geometry field is always used.

## Since

OGR 1.10

References CPLError(), CPLTestBool(), CSLFetchNameValueDef(), GetFeatureCount(), GetLayerDefn(), OGRGeometryFactory::haveGEOS(), OGRERR\_NONE, and OGRERR\_UNSUPPORTED\_OPERATION.

Referenced by OGR\_L\_Erase().

## 11.73.2.13 FindFieldIndex()

```
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().

Referenced by OGR\_L\_FindFieldIndex().

## 11.73.2.14 FromHandle()

```
static OGRLayer* OGRLayer::FromHandle (
    OGRLayerH hLayer ) [inline], [static]
```

Convert a OGRLayerH to a OGRLayer\*.

Since

GDAL 2.3

Referenced by OGR\_DS\_CopyLayer(), OGR\_DS\_ReleaseResultSet(), OGR\_L\_AlterFieldDefn(), OGR\_L\_Clip(), OGR\_L\_CommitTransaction(), OGR\_L\_CreateFeature(), OGR\_L\_CreateField(), OGR\_L\_CreateGeomField(), OGR\_L\_DeleteFeature(), OGR\_L\_DeleteField(), OGR\_L\_Erase(), OGR\_L\_FindFieldIndex(), OGR\_L\_GetExtent(), OGR\_L\_GetExtentEx(), OGR\_L\_GetFeature(), OGR\_L\_GetFeatureCount(), OGR\_L\_GetFIDColumn(), OGR\_L\_GetGeometryColumn(), OGR\_L\_GetGeomType(), OGR\_L\_GetLayerDefn(), OGR\_L\_GetName(), OGR\_L\_GetNextFeature(), OGR\_L\_GetSpatialFilter(), OGR\_L\_GetSpatialRef(), OGR\_L\_GetStyleTable(), OGR\_L\_GetIdentity(), OGR\_L\_Intersection(), OGR\_L\_ReorderField(), OGR\_L\_ReorderFields(), OGR\_L\_ResetReading(), OGR\_L\_RollbackTransaction(), OGR\_L\_SetAttributeFilter(), OGR\_L\_SetFeature(), OGR\_L\_SetIgnoredFields(), OGR\_L\_SetNextByIndex(), OGR\_L\_SetSpatialFilter(), OGR\_L\_SetSpatialFilterEx(), OGR\_L\_SetSpatialFilterRect(), OGR\_L\_SetSpatialFilterRectEx(), OGR\_L\_SetStyleTable(), OGR\_L\_SetStyleTableDirectly(), OGR\_L\_StartTransaction(), OGR\_L\_SymDifference(), OGR\_L\_SyncToDisk(), OGR\_L\_TestCapability(), OGR\_L\_Union(), and OGR\_L\_Update().

## 11.73.2.15 GetExtent() [1/2]

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

Fetch the extent of this layer.

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

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

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

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

This method is the same as the C function **OGR\_L\_GetExtent()** (p. ??).

## Parameters

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

**Returns**

OGRERR\_NONE on success, OGRERR\_FAILURE if extent not known.

Referenced by GetExtent(), Intersection(), OGR\_L\_GetExtent(), and OGR\_L\_GetExtentEx().

**11.73.2.16 GetExtent()** [2/2]

```
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 implementer: 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**

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

**Returns**

OGRERR\_NONE on success, OGRERR\_FAILURE if extent not known.

References GetExtent().

**11.73.2.17 GetFeature()**

```
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

<i>nFID</i>	the feature id of the feature to read.
-------------	--

#### Returns

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

References OGRGeometry::clone(), CPLFree, CPLStrdup(), SetAttributeFilter(), and SetSpatialFilter().

Referenced by OGR\_L\_GetFeature().

#### 11.73.2.18 GetFeatureCount()

```
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

<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.
---------------	---

## Returns

feature count, -1 if count not known.

Referenced by Clip(), Erase(), Identity(), Intersection(), OGR\_L\_GetFeatureCount(), SymDifference(), Union(), and Update().

## 11.73.2.19 GetFIDColumn()

```
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.

Referenced by OGR\_L\_GetFIDColumn().

## 11.73.2.20 GetGeometryColumn()

```
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.

References OGRFeatureDefn::GetGeomFieldDefn(), GetLayerDefn(), and OGRGeomFieldDefn::GetNameRef().

Referenced by OGR\_L\_GetGeometryColumn().

### 11.73.2.21 GetGeomType()

```
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

References **CPLDebug()**, **OGRFeatureDefn::GetGeomType()**, **GetLayerDefn()**, and **wkbUnknown**.

Referenced by **Identity()**, **Intersection()**, **OGR\_L\_GetGeomType()**, and **Union()**.

### 11.73.2.22 GetLayerDefn()

```
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.

Referenced by **Clip()**, **Erase()**, **FindFieldIndex()**, **GetGeometryColumn()**, **GetGeomType()**, **GetName()**, **GetSpatialRef()**, **Identity()**, **Intersection()**, **OGR\_L\_GetLayerDefn()**, **ReorderField()**, **SetIgnoredFields()**, **SetSpatialFilter()**, **SymDifference()**, **Union()**, and **Update()**.

### 11.73.2.23 GetName()

```
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 be freed)

#### Since

OGR 1.8.0

References **GetLayerDefn()**, and **OGRFeatureDefn::GetName()**.

Referenced by **OGR\_L\_GetName()**.

### 11.73.2.24 GetNextFeature()

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

Fetch the next available feature from this layer.

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

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

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

Features returned by **GetNextFeature()** (p. ??) may or may not be affected by concurrent modifications depending on drivers. A guaranteed way of seeing 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.

Referenced by **OGR\_L\_GetNextFeature()**, and **SetNextByIndex()**.

### 11.73.2.25 GetRefCount()

```
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.

### 11.73.2.26 GetSpatialFilter()

```
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.

Referenced by `OGR_L_GetSpatialFilter()`.

### 11.73.2.27 GetSpatialRef()

```
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.

References `OGRFeatureDefn::GetGeomFieldDefn()`, `GetLayerDefn()`, and `OGRGeomFieldDefn::GetSpatialRef()`.

Referenced by `OGR_L_GetSpatialRef()`.

## 11.73.2.28 GetStyleTable()

```
OGRStyleTable * OGRLayer::GetStyleTable ( ) [virtual]
```

Returns layer style table.

This method is the same as the C function **OGR\_L\_GetStyleTable()** (p. ??).

## Returns

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

Referenced by **OGR\_L\_GetStyleTable()**.

## 11.73.2.29 ICreateFeature()

```
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

<i>poFeature</i>	the feature to write to disk.
------------------	-------------------------------

## Returns

OGRERR\_NONE on success.

## Since

GDAL 2.0

References **OGRERR\_UNSUPPORTED\_OPERATION**.

Referenced by **CreateFeature()**.

## 11.73.2.30 Identity()

```

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

```

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

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

## Note

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

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

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

The recognized list of options is :

- SKIP\_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.
- USE\_PREPARED\_GEOMETRIES=YES/NO. Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- KEEP\_LOWER\_DIMENSION\_GEOMETRIES=YES/NO. Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

This method is the same as the C function **OGR\_L\_Identity()** (p. ??).

## Parameters

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

**Returns**

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

**Note**

The first geometry field is always used.

**Since**

OGR 1.10

References CPLDebug(), CPLError(), CPLTestBool(), CSLFetchNameValueDef(), GetFeatureCount(), GetGeom↵  
Type(), GetLayerDefn(), OGRGeometryFactory::haveGEOS(), OGRERR\_NONE, OGRERR\_UNSUPPORTED\_↵  
OPERATION, OGRHasPreparedGeometrySupport(), and wkbUnknown.

Referenced by OGR\_L\_Identity().

**11.73.2.31 Intersection()**

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

Intersection of two layers.

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

**Note**

If the schema of the result is set by user and contains fields that have the same name as a field in input and in method layer, then the attribute in the result feature will get the value from the feature of the method layer. For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

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

The recognized list of options is:

- SKIP\_FAILURES=YES/NO. Set to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.

- **METHOD\_PREFIX**=string. Set a prefix for the field names that will be created from the fields of the method layer.
- **USE\_PREPARED\_GEOMETRIES**=YES/NO. Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- **PRETEST\_CONTAINMENT**=YES/NO. Set to YES to pretest the containment of features of method layer within the features of this layer. This will speed up the method significantly in some cases. Requires that the prepared geometries are in effect.
- **KEEP\_LOWER\_DIMENSION\_GEOMETRIES**=YES/NO. Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

This method is the same as the C function **OGR\_L\_Intersection()** (p. ??).

#### Parameters

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

#### Returns

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

#### Note

The first geometry field is always used.

#### Since

OGR 1.10

References CPLDebug(), CPLError(), CPLTestBool(), CSLFetchNameValueDef(), GetExtent(), GetFeatureCount(), GetGeomType(), GetLayerDefn(), OGRGeometryFactory::haveGEOS(), OGRERR\_NONE, OGRERR\_UNSUPPORTED\_OPERATION, OGRHasPreparedGeometrySupport(), and wkbUnknown.

Referenced by OGR\_L\_Intersection().

#### 11.73.2.32 ISetFeature()

```
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

<i>poFeature</i>	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

References OGRERR\_UNSUPPORTED\_OPERATION.

Referenced by SetFeature().

## 11.73.2.33 Reference()

```
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.

## 11.73.2.34 ReorderField()

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

Reorder an existing field on a layer.

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

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

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

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

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

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

This function is the same as the C function **OGR\_L\_ReorderField()** (p. ??).

## Parameters

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

## Returns

OGRERR\_NONE on success.

## Since

OGR 1.9.0

References CPLError(), OGRFeatureDefn::GetFieldCount(), and GetLayerDefn().

Referenced by OGR\_L\_ReorderField().

## 11.73.2.35 ReorderFields()

```
OGRERR OGRLayer::ReorderFields (
    int * panMap ) [virtual]
```

Reorder all the fields of a layer.

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

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

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

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

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

This function is the same as the C function **OGR\_L\_ReorderFields()** (p. ??).

## Parameters

<i>panMap</i>	an array of <b>GetLayerDefn()</b> (p. ??)-> <b>OGRFeatureDefn::GetFieldCount()</b> (p. ??) elements which is a permutation of [0, <b>GetLayerDefn()</b> (p. ??)-> <b>OGRFeatureDefn::GetFieldCount()</b> (p. ??)-1].
---------------	--

**Returns**

OGRERR\_NONE on success.

**Since**

OGR 1.9.0

References CPLError().

Referenced by OGR\_L\_ReorderFields().

**11.73.2.36 ResetReading()**

```
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. ??).

Referenced by OGR\_L\_ResetReading(), SetAttributeFilter(), SetNextByIndex(), and SetSpatialFilter().

**11.73.2.37 RollbackTransaction()**

```
OGRERR OGRLayer::RollbackTransaction ( ) [virtual]
```

For datasources which support transactions, RollbackTransaction will roll back a datasource to its state before the start of the current transaction. If no transaction is active, or the rollback fails, will return OGRERR\_FAILURE. Datasources which do not support transactions will always return OGRERR\_NONE.

This function is the same as the C function **OGR\_L\_RollbackTransaction()** (p. ??).

**Returns**

OGRERR\_NONE on success.

References OGRERR\_UNSUPPORTED\_OPERATION.

Referenced by OGR\_L\_RollbackTransaction().

**11.73.2.38 SetAttributeFilter()**

```
OGRERR OGRLayer::SetAttributeFilter (
    const char * pszQuery ) [virtual]
```

Set a new attribute query.

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

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

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

This method is the same as the C function **OGR\_L\_SetAttributeFilter()** (p. ??).

## Parameters

<i>pszQuery</i>	query in restricted SQL WHERE format, or NULL to clear the current query.
-----------------	---

## Returns

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

References CPLFree, CPLStrdup(), OGRERR\_NONE, and ResetReading().

Referenced by GetFeature(), and OGR\_L\_SetAttributeFilter().

## 11.73.2.39 SetFeature()

```
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

<i>poFeature</i>	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 OGR\_L\_SetFeature().

## 11.73.2.40 SetIgnoredFields()

```
OGRERR OGRLayer::SetIgnoredFields (
    const char ** papszFields ) [virtual]
```

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

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

Besides field names of the layers, the following special fields can be passed: "OGR\_GEOMETRY" to ignore geometry and "OGR\_STYLE" to ignore layer style.

By default, no fields are ignored.

This method is the same as the C function **OGR\_L\_SetIgnoredFields()** (p. ??)

#### Parameters

<i>papszFields</i>	an array of field names terminated by NULL item. If NULL is passed, the ignored list is cleared.
--------------------	--

#### Returns

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

References EQUAL, OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetFieldIndex(), OGRFeatureDefn::GetGeomFieldCount(), OGRFeatureDefn::GetGeomFieldDefn(), OGRFeatureDefn::GetGeomFieldIndex(), GetLayerDefn(), OGRERR\_FAILURE, OGRERR\_NONE, OGRFeatureDefn::SetGeometryIgnored(), OGRFieldDefn::SetIgnored(), OGRGeomFieldDefn::SetIgnored(), and OGRFeatureDefn::SetStyleIgnored().

Referenced by OGR\_L\_SetIgnoredFields().

#### 11.73.2.41 SetNextByIndex()

```
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

<i>nIndex</i>	the index indicating how many steps into the result set to seek.
---------------	--

**Returns**

OGRERR\_NONE on success or an error code.

References `GetNextFeature()`, `OGRERR_FAILURE`, `OGRERR_NONE`, and `ResetReading()`.

Referenced by `OGR_L_SetNextByIndex()`.

**11.73.2.42 SetSpatialFilter()** [1/2]

```
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 whose 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.

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

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

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

This method is the same as the C function **OGR\_L\_SetSpatialFilter()** (p. ??).

**Parameters**

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

References `ResetReading()`.

Referenced by `GetFeature()`, `OGR_L_SetSpatialFilter()`, `OGR_L_SetSpatialFilterEx()`, `SetSpatialFilter()`, and `SetSpatialFilterRect()`.

**11.73.2.43 SetSpatialFilter()** [2/2]

```
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 implementer: 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

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

#### Since

GDAL 1.11

References **CPLError()**, **GetLayerDefn()**, and **SetSpatialFilter()**.

#### 11.73.2.44 SetSpatialFilterRect() [1/2]

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

Set a new rectangular spatial filter.

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

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

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

This method is the same as the C function **OGR\_L\_SetSpatialFilterRect()** (p. ??).

## Parameters

<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.
<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

Referenced by `OGR_L_SetSpatialFilterRect()`, and `OGR_L_SetSpatialFilterRectEx()`.

11.73.2.45 **SetSpatialFilterRect()** [2/2]

```
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

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

## Since

GDAL 1.11

References `OGRSimpleCurve::addPoint()`, `OGRCurvePolygon::addRing()`, and `SetSpatialFilter()`.



## 11.73.2.46 SetStyleTable()

```
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()** (p. ??).

## Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

References OGRStyleTable::Clone().

Referenced by OGR\_L\_SetStyleTable().

## 11.73.2.47 SetStyleTableDirectly()

```
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()** (p. ??).

## Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

Referenced by OGR\_L\_SetStyleTableDirectly().

## 11.73.2.48 StartTransaction()

```
OGRERR OGRLayer::StartTransaction ( ) [virtual]
```

For datasources which support transactions, StartTransaction creates a transaction.

If starting the transaction fails, will return OGRERR\_FAILURE. Datasources which do not support transactions will always return OGRERR\_NONE.

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

This function is the same as the C function **OGR\_L\_StartTransaction()** (p. ??).

**Returns**

OGRERR\_NONE on success.

References OGRERR\_NONE.

Referenced by OGR\_L\_StartTransaction().

**11.73.2.49 SymDifference()**

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

Symmetrical difference of two layers.

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

**Note**

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

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

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

The recognized list of options is :

- SKIP\_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

This method is the same as the C function **OGR\_L\_SymDifference()** (p. ??).

**Parameters**

<i>pLayerMethod</i>	the method layer. Should not be NULL.
<i>pLayerResult</i>	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
<i>papszOptions</i>	NULL terminated list of options (may be NULL). <span style="float: right;">Generated by Doxygen</span>
<i>pfnProgress</i>	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
<i>pProgressArg</i>	argument to be passed to pfnProgress. May be NULL.

**Returns**

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

**Note**

The first geometry field is always used.

**Since**

OGR 1.10

References CPLError(), CPLTestBool(), CSLFetchNameValueDef(), GetFeatureCount(), GetLayerDefn(), OGRGeometryFactory::haveGEOS(), OGRERR\_NONE, and OGRERR\_UNSUPPORTED\_OPERATION.

Referenced by OGR\_L\_SymDifference().

**11.73.2.50 SyncToDisk()**

```
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.

References OGRERR\_NONE.

Referenced by OGR\_L\_SyncToDisk().

## 11.73.2.51 TestCapability()

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

Test if this layer supported the named capability.

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

- **OLCRandomRead** / "RandomRead": TRUE if the **GetFeature()** (p. ??) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. ??) and **GetNextFeature()** (p. ??) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. ??) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. ??) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCFastSpatialFilter** / "FastSpatialFilter": TRUE if this layer implements spatial filtering efficiently. Layers that effectively read all features, and test them with the **OGRFeature** (p. ??) intersection methods should return FALSE. This can be used as a clue by the application whether it should build and maintain its own spatial index for features in this layer.
- **OLCFastFeatureCount** / "FastFeatureCount": TRUE if this layer can return a feature count (via **GetFeatureCount()** (p. ??)) efficiently. i.e. without counting the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastGetExtent** / "FastGetExtent": TRUE if this layer can return its data extent (via **GetExtent()** (p. ??)) efficiently, i.e. without scanning all the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastSetNextByIndex** / "FastSetNextByIndex": TRUE if this layer can perform the **SetNextByIndex()** (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()** (p. ??), **CommitTransaction()** (p. ??) and **RollbackTransaction()** (p. ??) 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

<i>pszCap</i>	the name of the capability to test.
---------------	-------------------------------------

#### Returns

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

Referenced by **OGR\_L\_TestCapability()**.

#### 11.73.2.52 ToHandle()

```
static OGRLayerH OGRLayer::ToHandle (
    OGRLayer * pOLayer ) [inline], [static]
```

Convert a **OGRLayer\*** to a **OGRLayerH**.

#### Since

GDAL 2.3

Referenced by **OGR\_DS\_CopyLayer()**, **OGR\_DS\_ExecuteSQL()**, **OGR\_DS\_GetLayer()**, and **OGR\_DS\_GetLayerBy←ByName()**.

#### 11.73.2.53 Union()

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

Union of two layers.

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

**Note**

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

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

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

The recognized list of options is :

- **SKIP\_FAILURES=**YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- **PROMOTE\_TO\_MULTI=**YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- **INPUT\_PREFIX=**string. Set a prefix for the field names that will be created from the fields of the input layer.
- **METHOD\_PREFIX=**string. Set a prefix for the field names that will be created from the fields of the method layer.
- **USE\_PREPARED\_GEOMETRIES=**YES/NO. Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- **KEEP\_LOWER\_DIMENSION\_GEOMETRIES=**YES/NO. Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

This method is the same as the C function **OGR\_L\_Union()** (p. ??).

**Parameters**

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

**Returns**

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

**Note**

The first geometry field is always used.

**Since**

OGR 1.10

References CPLDebug(), CPLError(), CPLTestBool(), CSLFetchNameValueDef(), GetFeatureCount(), GetGeomType(), GetLayerDefn(), OGRGeometryFactory::haveGEOS(), OGRERR\_NONE, OGRERR\_UNSUPPORTED\_OPERATION, OGRHasPreparedGeometrySupport(), and wkbUnknown.

Referenced by OGR\_L\_Union().

## 11.73.2.54 Update()

```

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

```

Update this layer with features from the update layer.

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

## Note

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

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

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

The recognized list of options is :

- SKIP\_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

This method is the same as the C function **OGR\_L\_Update()** (p. ??).

## Parameters

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

## Returns

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

**Note**

The first geometry field is always used.

**Since**

OGR 1.10

References CPLError(), CPLTestBool(), CSLFetchNameValueDef(), GetFeatureCount(), GetLayerDefn(), OGRGeometryFactory::haveGEOS(), OGRERR\_NONE, and OGRERR\_UNSUPPORTED\_OPERATION.

Referenced by OGR\_L\_Update().

### 11.73.3 Friends And Related Function Documentation

#### 11.73.3.1 begin

```
FeatureIterator begin (
    OGRLayer * poLayer ) [friend]
```

Return begin of feature iterator.

Using this iterator for standard range-based loops is safe, but due to implementation limitations, you shouldn't try to access (dereference) more than one iterator step at a time, since the std::unique\_ptr<OGRFeature (p. ??)> reference is reused.

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

**Since**

GDAL 2.3

**See also**

**OGRLayer::begin()** (p. ??)

Referenced by begin().

#### 11.73.3.2 end

```
FeatureIterator end (
    OGRLayer * poLayer ) [friend]
```

Return end of feature iterator.

**See also**

**OGRLayer::end()** (p. ??)

Referenced by end().

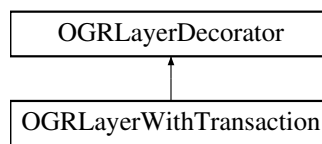
The documentation for this class was generated from the following files:

- ogrsf\_frmts.h
- ogrsf\_frmts.dox
- ogrlayer.cpp



## 11.74 OGRLayerWithTransaction Class Reference

Inheritance diagram for OGRLayerWithTransaction:



### Friends

- class **OGRDataSourceWithTransaction**

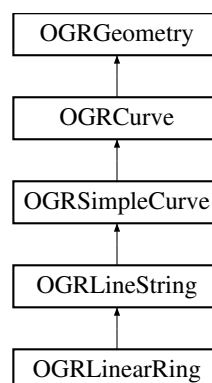
The documentation for this class was generated from the following file:

- ogremulatedtransaction.cpp

## 11.75 OGRLinearRing Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRLinearRing:



### Public Member Functions

- **OGRLinearRing** ()
- **OGRLinearRing** (const **OGRLinearRing** &other)  
*Copy constructor.*
- **OGRLinearRing** ( **OGRLinearRing** \*)
- **OGRLinearRing** & **operator=** (const **OGRLinearRing** &other)  
*Assignment operator.*
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRGeometry** \* **clone** () const override

- 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 **reverseWindingOrder** ()
- virtual void **closeRings** () override
- Force rings to be closed.*

  - **OGRBoolean isPointInRing** (const **OGRPoint** \*pt, int bTestEnvelope=TRUE) const
  - **OGRBoolean isPointOnRingBoundary** (const **OGRPoint** \*pt, int bTestEnvelope=TRUE) const
- virtual **OGRERR** **transform** ( **OGRCoordinateTransformation** \*poCT) override
- Apply arbitrary coordinate transformation to geometry.*

  - **OGRLineString** \* **toUpperClass** ()
  - const **OGRLineString** \* **toUpperClass** () const
  - virtual void **accept** ( **OGRGeometryVisitor** \*visitor) override
  - virtual void **accept** ( **OGRConstGeometryVisitor** \*visitor) const override
  - virtual int **WkbSize** () const override

*Returns size of related binary representation.*
- virtual **OGRERR** **importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumedOut) override
- Assign geometry from well known binary data.*

  - virtual **OGRERR** **exportToWkb** ( **OGRwkbByteOrder**, unsigned char \*, **OGRwkbVariant**= **wkbVariant**↔**OldOgc**) const override

*Convert a geometry into well known binary format.*

## Static Protected Member Functions

- static **OGRLineString** \* **CastToLineString** ( **OGRLinearRing** \*poLR)
- Cast to line string.*

## Additional Inherited Members

### 11.75.1 Detailed Description

Concrete representation of a closed ring.

This class is functionally equivalent to an **OGRLineString** (p. ??), but has a separate identity to maintain alignment with the OpenGIS simple feature data model. It exists to serve as a component of an **OGRPolygon** (p. ??).

The **OGRLinearRing** (p. ??) has no corresponding free standing well known binary representation, so **import↔FromWkb()** (p. ??) and **exportToWkb()** (p. ??) will not actually work. There is a non-standard GDAL WKT representation though.

Because **OGRLinearRing** (p. ??) is not a "proper" free standing simple features object, it cannot be directly used on a feature via **SetGeometry()**, and cannot generally be used with GEOS for operations like **Intersects()** (p. ??). Instead the polygon should be used, or the **OGRLinearRing** (p. ??) should be converted to an **OGRLineString** (p. ??) for such operations.

Note: this class exists in SFSQL 1.2, but not in ISO SQL/MM Part 3.

### 11.75.2 Constructor & Destructor Documentation

**11.75.2.1 OGRLinearRing()** [1/3]

```
OGRLinearRing::OGRLinearRing ( )
```

Constructor

Referenced by clone().

**11.75.2.2 OGRLinearRing()** [2/3]

```
OGRLinearRing::OGRLinearRing (
    const OGRLinearRing & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

**11.75.2.3 OGRLinearRing()** [3/3]

```
OGRLinearRing::OGRLinearRing (
    OGRLinearRing * poSrcRing ) [explicit]
```

Constructor

Parameters

<i>poSrcRing</i>	source ring.
------------------	--------------

References CPLDebug(), OGRSimpleCurve::getNumPoints(), and OGRSimpleCurve::setNumPoints().

**11.75.3 Member Function Documentation****11.75.3.1 accept()** [1/2]

```
virtual void OGRLinearRing::accept (
    OGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRGeometryVisitor::visit().

### 11.75.3.2 accept() [2/2]

```
virtual void OGRLinearRing::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRConstGeometryVisitor::visit().

### 11.75.3.3 CastToLineString()

```
OGRLineString * OGRLinearRing::CastToLineString (
    OGRLinearRing * poLR ) [static], [protected]
```

Cast to line string.

The passed in geometry is consumed and a new one returned .

#### Parameters

<i>poLR</i>	the input geometry - ownership is passed to the method.
-------------	---

#### Returns

new geometry.

References OGRLineString::OGRLineString().

### 11.75.3.4 clone()

```
OGRGeometry * OGRLinearRing::clone ( ) const [override], [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(), OGRLinearRing(), and OGRSimpleCurve::setPoints().

**11.75.3.5 closeRings()**

```
void OGRLinearRing::closeRings ( ) [override], [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(), OGRSimpleCurve::getY(), and OGRSimpleCurve::getZ().

**11.75.3.6 exportToWkb()**

```
OGRERR OGRLinearRing::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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**

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
<i>eWkbVariant</i>	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.

< Unsupported operation

< Success

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

References OGRERR\_UNSUPPORTED\_OPERATION.

**11.75.3.7 getGeometryName()**

```
const char * OGRLinearRing::getGeometryName ( ) const [override], [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. ??).

**11.75.3.8 importFromWkb()**

```
OGRERR OGRLinearRing::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

**Parameters**

<i>pabyData</i>	the binary input data.	Generated by Doxygen
<i>nSize</i>	the size of <i>pabyData</i> in bytes, or -1 if not known.	
<i>eWkbVariant</i>	if <i>wkbVariantPostGIS1</i> , special interpretation is done for curve geometries.	
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.	

**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.

**Since**

GDAL 2.3

< Unsupported operation

< Success

< Corrupt data

< Not enough data to deserialize

< Failure

< Success

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

References OGRERR\_UNSUPPORTED\_OPERATION.

**11.75.3.9 isClockwise()**

```
int OGRLinearRing::isClockwise ( ) const [virtual]
```

Returns TRUE if the ring has clockwise winding (or less than 2 points)

**Returns**

TRUE if clockwise otherwise FALSE.

**11.75.3.10 isPointInRing()**

```
OGRBoolean OGRLinearRing::isPointInRing (
    const OGRPoint * poPoint,
    int bTestEnvelope = TRUE ) const
```

Returns whether the point is inside the ring.

**Parameters**

<i>poPoint</i>	point
<i>bTestEnvelope</i>	set to TRUE if the presence of the point inside the ring envelope must be checked first.

**Returns**

TRUE or FALSE.

References CPLDebug(), OGRSimpleCurve::getEnvelope(), OGRSimpleCurve::getNumPoints(), OGRPoint::getX(), OGRSimpleCurve::getX(), OGRPoint::getY(), OGRSimpleCurve::getY(), and OGRPoint::isEmpty().

**11.75.3.11 isPointOnRingBoundary()**

```
OGRBoolean OGRLinearRing::isPointOnRingBoundary (
    const  OGRPoint * poPoint,
    int bTestEnvelope = TRUE ) const
```

Returns whether the point is on the ring boundary.

**Parameters**

<i>poPoint</i>	point
<i>bTestEnvelope</i>	set to TRUE if the presence of the point inside the ring envelope must be checked first.

**Returns**

TRUE or FALSE.

References CPLDebug(), OGRSimpleCurve::getEnvelope(), OGRSimpleCurve::getNumPoints(), OGRPoint::getX(), OGRSimpleCurve::getX(), OGRPoint::getY(), and OGRSimpleCurve::getY().

**11.75.3.12 operator=()**

```
OGRLinearRing & OGRLinearRing::operator= (
    const  OGRLinearRing & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

**Since**

GDAL 2.1

References OGRLinearString::operator=().



## 11.75.3.13 reverseWindingOrder()

```
void OGRLinearRing::reverseWindingOrder ( ) [virtual]
```

Reverse order of points.

References OGRSimpleCurve::getPoint(), and OGRSimpleCurve::setPoint().

## 11.75.3.14 toUpperClass() [1/2]

```
OGRLineString* OGRLinearRing::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

## 11.75.3.15 toUpperClass() [2/2]

```
const OGRLineString* OGRLinearRing::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

## 11.75.3.16 transform()

```
OGRERR OGRLinearRing::transform (
    OGRCoordinateTransformation * poCT ) [override], [virtual]
```

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGR↔SpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

## Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

**Returns**

OGRERR\_NONE on success or an error code.

< Success

< Not enough memory

< Failure

< Failure

< Failure

< Success

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

References OGRSimpleCurve::getNumPoints().

**11.75.3.17 WkbSize()**

```
int OGRLinearRing::WkbSize ( ) const [override], [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. ??).

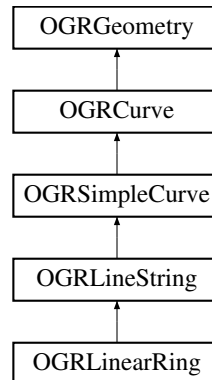
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrlinearring.cpp

## 11.76 OGRLineString Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRLineString:



### Public Member Functions

- **OGRLineString** ()  
*Create an empty line string.*
- **OGRLineString** (const **OGRLineString** &other)  
*Copy constructor.*
- **OGRLineString & operator=** (const **OGRLineString** &other)  
*Assignment operator.*
- virtual **OGRLineString \* CurveToLine** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*papszOptions=nullptr) const override  
*Return a linestring from a curve geometry.*
- virtual **OGRGeometry \* getCurveGeometry** (const char \*const \*papszOptions=nullptr) const override  
*Return curve version of this geometry.*
- virtual double **get\_Area** () const override  
*Get the area of the (closed) curve.*
- virtual **OGRwkbGeometryType getGeometryType** () const override  
*Fetch geometry type.*
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- **OGRSimpleCurve \* toUpperClass** ()
- const **OGRSimpleCurve \* toUpperClass** () const
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override

### Static Protected Member Functions

- static **OGRLinearRing \* CastToLinearRing** ( **OGRLineString** \*poLS)  
*Cast to linear ring.*

## Additional Inherited Members

### 11.76.1 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. ??).

### 11.76.2 Constructor & Destructor Documentation

#### 11.76.2.1 OGRLineString()

```
OGRLineString::OGRLineString (
    const OGRLineString & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

### 11.76.3 Member Function Documentation

#### 11.76.3.1 accept() [1/2]

```
virtual void OGRLineString::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

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

References **IOGRGeometryVisitor::visit()**.

11.76.3.2 `accept()` [2/2]

```
virtual void OGRLineString::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

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

References `IOGRConstGeometryVisitor::visit()`.

11.76.3.3 `CastToLinearRing()`

```
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

<i>poLS</i>	the input geometry - ownership is passed to the method.
-------------	---

## Returns

new geometry.

References `CPLError()`, and `OGRCurve::get_IsClosed()`.

11.76.3.4 `CurveToLine()`

```
OGRLineString * OGRLineString::CurveToLine (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings or NULL. See <b>OGRGeometryFactory::curveToLineString()</b> (p. ??) for valid options.

## Returns

a line string approximating the curve

## Since

GDAL 2.0

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

References **OGRSimpleCurve::clone()**, and **OGRGeometry::toLineString()**.

Referenced by **OGRCurvePolygon::CurvePolyToPoly()**.

11.76.3.5 **get\_Area()**

```
virtual double OGRLineString::get_Area ( ) const [override], [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. ??).

Referenced by **OGRCircularString::get\_Area()**, and **OGRCompoundCurve::get\_Area()**.

11.76.3.6 **getCurveGeometry()**

```
OGRGeometry * OGRLineString::getCurveGeometry (
    const char *const * papszOptions = nullptr ) const [override], [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

<i>papszOptions</i>	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().

## 11.76.3.7 getGeometryName()

```
const char * OGRLineString::getGeometryName ( ) const [override], [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. ??).

## 11.76.3.8 getGeometryType()

```
OGRwkbGeometryType OGRLineString::getGeometryType ( ) const [override], [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, wkbLineString25D, wkbLineStringM, and wkbLineStringZM.

### 11.76.3.9 operator=()

```
OGRLineString & OGRLineString::operator= (
    const OGRLineString & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

Since

GDAL 2.1

References OGRSimpleCurve::operator=().

Referenced by OGRLinearRing::operator=().

### 11.76.3.10 toUpperClass() [1/2]

```
OGRSimpleCurve* OGRLineString::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

### 11.76.3.11 toUpperClass() [2/2]

```
const OGRSimpleCurve* OGRLineString::toUpperClass ( ) const [inline]
```

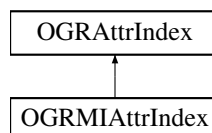
Return pointer of this in upper class

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

- **ogr\_geometry.h**
- ogrlinestring.cpp

## 11.77 OGRMAttrIndex Class Reference

Inheritance diagram for OGRMAttrIndex:



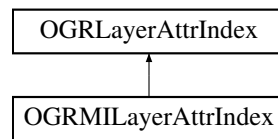
The documentation for this class was generated from the following file:

- ogr\_miattrind.cpp



## 11.78 OGRMILayerAttrIndex Class Reference

Inheritance diagram for OGRMILayerAttrIndex:



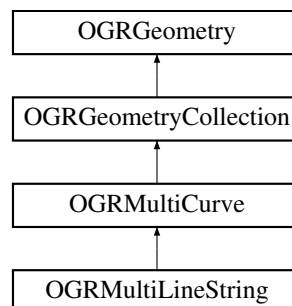
The documentation for this class was generated from the following file:

- ogr\_miattrind.cpp

## 11.79 OGRMultiCurve Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRMultiCurve:



### Public Types

- typedef **OGRCurve** **ChildType**

### Public Member Functions

- **OGRMultiCurve** ()  
*Create an empty multi curve collection.*
- **OGRMultiCurve** (const **OGRMultiCurve** &other)  
*Copy constructor.*
- **OGRMultiCurve** & **operator=** (const **OGRMultiCurve** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual const char \* **getGeometryName** () const override

- Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override
- Fetch geometry type.*
- **OGRErr** **importFromWkt** (const char \*\*) override
- virtual **OGRErr** **exportToWkt** (char \*\*, **OGRwkbVariant**= **wkbVariantOldOgc**) const override
- Convert a geometry into well known text format.*
- virtual int **getDimension** () const override
- Get the dimension of this object.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override
- Returns if this geometry is or has curve geometry.*
- **OGRGeometryCollection** \* **toUpperClass** ()
- const **OGRGeometryCollection** \* **toUpperClass** () const
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override
- virtual **OGRErr** **importFromWkt** (const char \*\*ppszInput)=0
- Assign geometry from well known text data.*
- **OGRErr** **importFromWkt** (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

## Static Public Member Functions

- static **OGRMultiLineString** \* **CastToMultiLineString** ( **OGRMultiCurve** \*poMC)
- Cast to multi line string.*

## Protected Member Functions

- virtual **OGRBoolean** **isCompatibleSubType** ( **OGRwkbGeometryType**) const override

### 11.79.1 Detailed Description

A collection of **OGRCurve** (p. ??).

Since

GDAL 2.0

### 11.79.2 Member Typedef Documentation

#### 11.79.2.1 ChildType

```
typedef OGRCurve OGRMultiCurve::ChildType
```

Type of child elements.

### 11.79.3 Constructor & Destructor Documentation

#### 11.79.3.1 OGRMultiCurve()

```
OGRMultiCurve::OGRMultiCurve (
    const OGRMultiCurve & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

### 11.79.4 Member Function Documentation

#### 11.79.4.1 accept() [1/2]

```
virtual void OGRMultiCurve::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

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

References **IOGRGeometryVisitor::visit()**.

#### 11.79.4.2 accept() [2/2]

```
virtual void OGRMultiCurve::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

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

References **IOGRConstGeometryVisitor::visit()**.

**11.79.4.3 begin()** [1/2]

```
ChildType** OGRMultiCurve::begin ( ) [inline]
```

Return begin of iterator.

Since

GDAL 2.3

**11.79.4.4 begin()** [2/2]

```
const ChildType* const* OGRMultiCurve::begin ( ) const [inline]
```

Return begin of iterator.

Since

GDAL 2.3

**11.79.4.5 CastToMultiLineString()**

```
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

<i>poMC</i>	the input geometry - ownership is passed to the method.
-------------	---

Returns

new geometry.

References `OGRCurve::CastToLineString()`.

Referenced by `OGRGeometryFactory::forceToMultiLineString()`.

11.79.4.6 `end()` [1/2]

```
ChildType** OGRMultiCurve::end ( ) [inline]
```

Return end of iterator

11.79.4.7 `end()` [2/2]

```
const ChildType* const* OGRMultiCurve::end ( ) const [inline]
```

Return end of iterator

11.79.4.8 `exportToWkt()`

```
OGRERR OGRMultiCurve::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>• <b>wkbVariantOgc</b> for old-style 99-402 extended dimension (Z) WKB types</li> <li>• <b>wkbVariantIso</b> for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

## Returns

Currently OGRERR\_NONE is always returned.

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

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

References **wkbVariantIso**.

#### 11.79.4.9 `getDimension()`

```
int OGRMultiCurve::getDimension ( ) const [override], [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. ??).

#### 11.79.4.10 `getGeometryName()`

```
const char * OGRMultiCurve::getGeometryName ( ) const [override], [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. ??).

#### 11.79.4.11 `getGeometryType()`

```
OGRwkbGeometryType OGRMultiCurve::getGeometryType ( ) const [override], [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 `wkbMultiCurve`, `wkbMultiCurveM`, `wkbMultiCurveZ`, and `wkbMultiCurveZM`.

Referenced by `importFromWkt()`.

## 11.79.4.12 hasCurveGeometry()

```
OGRBoolean OGRMultiCurve::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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

<i>bLookForNonLinear</i>	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()**.

Referenced by **OGRGeometryFactory::forceToMultiLineString()**.

## 11.79.4.13 importFromWkt() [1/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

## 11.79.4.14 importFromWkt() [2/3]

```
OGRERR OGRGeometry::importFromWkt
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

## 11.79.4.15 importFromWkt() [3/3]

```
OGRERR OGRMultiCurve::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated

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

References getGeometryType(), wkbFlatten, and wkbMultiCurve.

## 11.79.4.16 isCompatibleSubType()

```
OGRBoolean OGRMultiCurve::isCompatibleSubType (
    OGRwkbGeometryType ) const [override], [protected], [virtual]
```

Returns whether a geometry of the specified geometry type can be a member of this collection.

## Parameters

<i>eSubType</i>	type of the potential member
-----------------	------------------------------

## Returns

TRUE or FALSE

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

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

References OGR\_GT\_IsCurve().



## 11.79.4.17 operator=()

```
OGRMultiCurve & OGRMultiCurve::operator= (
    const OGRMultiCurve & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

Since

GDAL 2.1

References OGRGeometryCollection::operator=().

Referenced by OGRMultiLineString::operator=().

## 11.79.4.18 toUpperClass() [1/2]

```
OGRGeometryCollection* OGRMultiCurve::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

## 11.79.4.19 toUpperClass() [2/2]

```
const OGRGeometryCollection* OGRMultiCurve::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

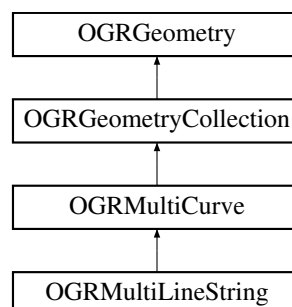
The documentation for this class was generated from the following files:

- ogr\_geometry.h
- ogramulticurve.cpp

## 11.80 OGRMultiLineString Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRMultiLineString:



## Public Types

- typedef **OGRLineString** **ChildType**

## Public Member Functions

- **OGRMultiLineString** ()  
*Create an empty multi line string collection.*
- **OGRMultiLineString** (const **OGRMultiLineString** &other)  
*Copy constructor.*
- **OGRMultiLineString** & **operator=** (const **OGRMultiLineString** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Fetch geometry type.*
- virtual **OGRERR** **exportToWkt** (char \*\*, **OGRwkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known text format.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override  
*Returns if this geometry is or has curve geometry.*
- **OGRGeometryCollection** \* **toUpperClass** ()
- const **OGRGeometryCollection** \* **toUpperClass** () const
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override

## Static Public Member Functions

- static **OGRMultiCurve** \* **CastToMultiCurve** ( **OGRMultiLineString** \*poMLS)  
*Cast to multicurve.*

## Protected Member Functions

- virtual **OGRBoolean** **isCompatibleSubType** ( **OGRwkbGeometryType**) const override

### 11.80.1 Detailed Description

A collection of **OGRLineString** (p. ??).

### 11.80.2 Member Typedef Documentation

### 11.80.2.1 ChildType

```
typedef OGRLineString OGRMultiLineString::ChildType
```

Type of child elements.

## 11.80.3 Constructor & Destructor Documentation

### 11.80.3.1 OGRMultiLineString()

```
OGRMultiLineString::OGRMultiLineString (
    const OGRMultiLineString & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

## 11.80.4 Member Function Documentation

### 11.80.4.1 accept() [1/2]

```
virtual void OGRMultiLineString::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

References **IOGRGeometryVisitor::visit()**.

### 11.80.4.2 accept() [2/2]

```
virtual void OGRMultiLineString::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

References **IOGRConstGeometryVisitor::visit()**.

**11.80.4.3 begin()** [1/2]

```
ChildType** OGRMultiLineString::begin ( ) [inline]
```

Return begin of iterator.

Since

GDAL 2.3

**11.80.4.4 begin()** [2/2]

```
const ChildType* const* OGRMultiLineString::begin ( ) const [inline]
```

Return begin of iterator.

Since

GDAL 2.3

**11.80.4.5 CastToMultiCurve()**

```
OGRMultiCurve * OGRMultiLineString::CastToMultiCurve (
    OGRMultiLineString * poMLS ) [static]
```

Cast to multcurve.

The passed in geometry is consumed and a new one returned.

Parameters

<i>poMLS</i>	the input geometry - ownership is passed to the method.
--------------	---

Returns

new geometry.

References OGRMultiCurve::OGRMultiCurve().

**11.80.4.6 end()** [1/2]

```
ChildType** OGRMultiLineString::end ( ) [inline]
```

Return end of iterator

11.80.4.7 `end()` [2/2]

```
const ChildType* const* OGRMultiLineString::end ( ) const [inline]
```

Return end of iterator

11.80.4.8 `exportToWkt()`

```
OGRERR OGRMultiLineString::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>• wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>• wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

## Returns

Currently OGRERR\_NONE is always returned.

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

11.80.4.9 `getGeometryName()`

```
const char * OGRMultiLineString::getGeometryName ( ) const [override], [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. ??).

11.80.4.10 `getGeometryType()`

```
OGRwkbGeometryType OGRMultiLineString::getGeometryType ( ) const [override], [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 `wkbMultiLineString`, `wkbMultiLineString25D`, `wkbMultiLineStringM`, and `wkbMultiLineStringZM`.

11.80.4.11 `hasCurveGeometry()`

```
OGRBoolean OGRMultiLineString::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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**

<i>bLookForNonLinear</i>	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. ??).

## 11.80.4.12 isCompatibleSubType()

```
OGRBoolean OGRMultiLineString::isCompatibleSubType (
    OGRwkbGeometryType ) const [override], [protected], [virtual]
```

Returns whether a geometry of the specified geometry type can be a member of this collection.

## Parameters

<i>eSubType</i>	type of the potential member
-----------------	------------------------------

## Returns

TRUE or FALSE

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

References wkbFlatten, and wkbLineString.

## 11.80.4.13 operator=()

```
OGRMultiLineString & OGRMultiLineString::operator= (
    const OGRMultiLineString & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

## Since

GDAL 2.1

References OGRMultiCurve::operator=().

## 11.80.4.14 toUpperClass() [1/2]

```
OGRGeometryCollection* OGRMultiLineString::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

## 11.80.4.15 toUpperClass() [2/2]

```
const OGRGeometryCollection* OGRMultiLineString::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

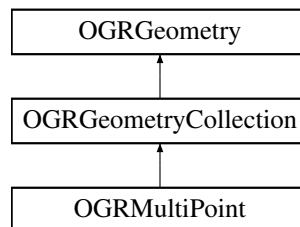
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrmultilinestring.cpp**

## 11.81 OGRMultiPoint Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRMultiPoint:



### Public Types

- typedef **OGRPoint** **ChildType**

### Public Member Functions

- **OGRMultiPoint** ()  
*Create an empty multi point collection.*
- **OGRMultiPoint** (const **OGRMultiPoint** &other)  
*Copy constructor.*
- **OGRMultiPoint** & **operator=** (const **OGRMultiPoint** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Fetch geometry type.*
- **OGRErr** **importFromWkt** (const char \*\*) override
- virtual **OGRErr** **exportToWkt** (char \*\*, **OGRwkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known text format.*
- virtual int **getDimension** () const override



*Get the dimension of this object.*

- **OGRGeometryCollection \* toUpperClass ()**
- **const OGRGeometryCollection \* toUpperClass () const**
- **virtual void accept ( IOGRGeometryVisitor \*visitor) override**
- **virtual void accept ( IOGRConstGeometryVisitor \*visitor) const override**
- **virtual OGRBoolean hasCurveGeometry (int bLookForNonLinear=FALSE) const override**

*Returns if this geometry is or has curve geometry.*

- **virtual OGRErr importFromWkt (const char \*\*ppszInput)=0**

*Assign geometry from well known text data.*

- **OGRErr importFromWkt (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")**

## Protected Member Functions

- **virtual OGRBoolean isCompatibleSubType ( OGRwkbGeometryType) const override**

## Additional Inherited Members

### 11.81.1 Detailed Description

A collection of **OGRPoint** (p. ??).

### 11.81.2 Member Typedef Documentation

#### 11.81.2.1 ChildType

```
typedef OGRPoint OGRMultiPoint::ChildType
```

Type of child elements.

### 11.81.3 Constructor & Destructor Documentation

#### 11.81.3.1 OGRMultiPoint()

```
OGRMultiPoint::OGRMultiPoint (
    const OGRMultiPoint & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

## 11.81.4 Member Function Documentation

### 11.81.4.1 `accept()` [1/2]

```
virtual void OGRMultiPoint::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRGeometryVisitor::visit().

### 11.81.4.2 `accept()` [2/2]

```
virtual void OGRMultiPoint::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRConstGeometryVisitor::visit().

### 11.81.4.3 `begin()` [1/2]

```
ChildType** OGRMultiPoint::begin ( ) [inline]
```

Return begin of iterator.

Since

GDAL 2.3

### 11.81.4.4 `begin()` [2/2]

```
const ChildType* const* OGRMultiPoint::begin ( ) const [inline]
```

Return begin of iterator.

Since

GDAL 2.3

11.81.4.5 `end()` [1/2]

```
ChildType** OGRMultiPoint::end ( ) [inline]
```

Return end of iterator

11.81.4.6 `end()` [2/2]

```
const ChildType* const* OGRMultiPoint::end ( ) const [inline]
```

Return end of iterator

11.81.4.7 `exportToWkt()`

```
OGRERR OGRMultiPoint::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

## Returns

Currently OGRERR\_NONE is always returned.

< Success

< Not enough memory

< Success

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

References **CPLDebug()**, **CPLRealloc()**, **CPLStrdup()**, **getGeometryName()**, **OGRGeometryCollection::getNumGeometries()**, **OGRGeometryCollection::IsEmpty()**, **OGRERR\_NONE**, **OGRERR\_NOT\_ENOUGH\_MEMORY**, **VSIMALLOC\_VERBOSE**, and **wkbVariantIso**.

#### 11.81.4.8 `getDimension()`

```
int OGRMultiPoint::getDimension ( ) const [override], [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. ??).

#### 11.81.4.9 `getGeometryName()`

```
const char * OGRMultiPoint::getGeometryName ( ) const [override], [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()`.

#### 11.81.4.10 `getGeometryType()`

```
OGRwkbGeometryType OGRMultiPoint::getGeometryType ( ) const [override], [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 `wkbMultiPoint`, `wkbMultiPoint25D`, `wkbMultiPointM`, and `wkbMultiPointZM`.

## 11.81.4.11 hasCurveGeometry()

```
OGRBoolean OGRMultiPoint::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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

<i>bLookForNonLinear</i>	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. ??).

## 11.81.4.12 importFromWkt() [1/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

## 11.81.4.13 importFromWkt() [2/3]

```
OGRERR OGRGeometry::importFromWkt
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

OGRErr\_NONE if all goes well, otherwise any of OGRErr\_NOT\_ENOUGH\_DATA, OGRErr\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRErr\_CORRUPT\_DATA may be returned.

## 11.81.4.14 importFromWkt() [3/3]

```
OGRErr OGRMultiPoint::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated < Success

< Success

< Success

< Corrupt data

< Corrupt data

< Success

< Success

< Success

< Success

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

References OGRGeometryCollection::addGeometryDirectly(), CPLFree, EQUAL, OGRErr\_CORRUPT\_DATA, OGRErr\_NONE, OGRPoint::setM(), and OGRPoint::setZ().

## 11.81.4.15 isCompatibleSubType()

```
OGRBoolean OGRMultiPoint::isCompatibleSubType (
    OGRwkbGeometryType ) const [override], [protected], [virtual]
```

Returns whether a geometry of the specified geometry type can be a member of this collection.

## Parameters

<i>eSubType</i>	type of the potential member
-----------------	------------------------------

## Returns

TRUE or FALSE

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

References wkbFlatten, and wkbPoint.

## 11.81.4.16 operator=()

```
OGRMultiPoint & OGRMultiPoint::operator= (
    const OGRMultiPoint & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

## Since

GDAL 2.1

References OGRGeometryCollection::operator=().

## 11.81.4.17 toUpperClass() [1/2]

```
OGRGeometryCollection* OGRMultiPoint::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

## 11.81.4.18 toUpperClass() [2/2]

```
const OGRGeometryCollection* OGRMultiPoint::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

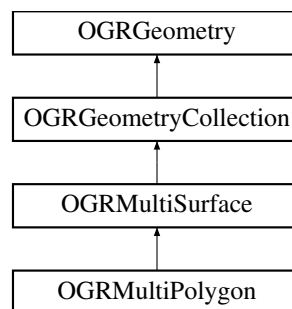
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrmultipoint.cpp**

## 11.82 OGRMultiPolygon Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRMultiPolygon:



## Public Types

- typedef **OGRPolygon** **ChildType**

## Public Member Functions

- **OGRMultiPolygon** ()  
*Create an empty multi polygon collection.*
- **OGRMultiPolygon** (const **OGRMultiPolygon** &other)  
*Copy constructor.*
- **OGRMultiPolygon** & **operator=** (const **OGRMultiPolygon** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Fetch geometry type.*
- virtual **OGRErr** **exportToWkt** (char \*\*, **OGRwkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known text format.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override  
*Returns if this geometry is or has curve geometry.*
- **OGRGeometryCollection** \* **toUpperClass** ()
- const **OGRGeometryCollection** \* **toUpperClass** () const
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override



## Static Public Member Functions

- static **OGRMultiSurface** \* **CastToMultiSurface** ( **OGRMultiPolygon** \*poMP)  
*Cast to multisurface.*

## Protected Member Functions

- virtual **OGRBoolean** **isCompatibleSubType** ( **OGRwkbGeometryType**) const override

## Friends

- class **OGRPolyhedralSurface**
- class **OGRTriangulatedSurface**

### 11.82.1 Detailed Description

A collection of non-overlapping **OGRPolygon** (p. ??).

### 11.82.2 Member Typedef Documentation

#### 11.82.2.1 ChildType

```
typedef OGRPolygon OGRMultiPolygon::ChildType
```

Type of child elements.

### 11.82.3 Constructor & Destructor Documentation

#### 11.82.3.1 OGRMultiPolygon()

```
OGRMultiPolygon::OGRMultiPolygon (  
    const OGRMultiPolygon & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

## 11.82.4 Member Function Documentation

### 11.82.4.1 `accept()` [1/2]

```
virtual void OGRMultiPolygon::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRGeometryVisitor::visit().

### 11.82.4.2 `accept()` [2/2]

```
virtual void OGRMultiPolygon::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRConstGeometryVisitor::visit().

### 11.82.4.3 `begin()` [1/2]

```
ChildType** OGRMultiPolygon::begin ( ) [inline]
```

Return begin of iterator.

Since

GDAL 2.3

Referenced by OGRPolyhedralSurface::begin().

### 11.82.4.4 `begin()` [2/2]

```
const ChildType* const* OGRMultiPolygon::begin ( ) const [inline]
```

Return begin of iterator.

Since

GDAL 2.3

### 11.82.4.5 `CastToMultiSurface()`

```
OGRMultiSurface * OGRMultiPolygon::CastToMultiSurface (
    OGRMultiPolygon * poMP ) [static]
```

Cast to multisurface.

The passed in geometry is consumed and a new one returned .

## Parameters

<i>poMP</i>	the input geometry - ownership is passed to the method.
-------------	---

## Returns

new geometry.

References OGRMultiSurface::OGRMultiSurface().

## 11.82.4.6 end() [1/2]

```
ChildType** OGRMultiPolygon::end ( ) [inline]
```

Return end of iterator

Referenced by OGRPolyhedralSurface::end().

## 11.82.4.7 end() [2/2]

```
const ChildType* const* OGRMultiPolygon::end ( ) const [inline]
```

Return end of iterator

## 11.82.4.8 exportToWkt()

```
OGRERR OGRMultiPolygon::exportToWkt (
    char ** ppszDstText,
    OGRWkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

**Returns**

Currently OGRERR\_NONE is always returned.

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

**11.82.4.9 getGeometryName()**

```
const char * OGRMultiPolygon::getGeometryName ( ) const [override], [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. ??).

**11.82.4.10 getGeometryType()**

```
OGRwkbGeometryType OGRMultiPolygon::getGeometryType ( ) const [override], [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 wkbMultiPolygon, wkbMultiPolygon25D, wkbMultiPolygonM, and wkbMultiPolygonZM.

**11.82.4.11 hasCurveGeometry()**

```
OGRBoolean OGRMultiPolygon::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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

<i>bLookForNonLinear</i>	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. ??).

## 11.82.4.12 isCompatibleSubType()

```
OGRBoolean OGRMultiPolygon::isCompatibleSubType (
    OGRwkbGeometryType ) const [override], [protected], [virtual]
```

Returns whether a geometry of the specified geometry type can be a member of this collection.

## Parameters

<i>eSubType</i>	type of the potential member
-----------------	------------------------------

## Returns

TRUE or FALSE

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

References wkbFlatten, and wkbPolygon.

## 11.82.4.13 operator=()

```
OGRMultiPolygon & OGRMultiPolygon::operator= (
    const OGRMultiPolygon & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

## Since

GDAL 2.1

References OGRMultiSurface::operator=().

#### 11.82.4.14 toUpperClass() [1/2]

```
OGRGeometryCollection* OGRMultiPolygon::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

#### 11.82.4.15 toUpperClass() [2/2]

```
const OGRGeometryCollection* OGRMultiPolygon::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

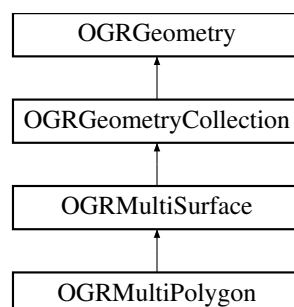
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrmultipolygon.cpp

## 11.83 OGRMultiSurface Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRMultiSurface:



### Public Types

- typedef **OGRSurface** **ChildType**

## Public Member Functions

- **OGRMultiSurface ()**  
*Create an empty multi surface collection.*
- **OGRMultiSurface (const OGRMultiSurface &other)**  
*Copy constructor.*
- **OGRMultiSurface & operator= (const OGRMultiSurface &other)**  
*Assignment operator.*
- **ChildType \*\* begin ()**
- **ChildType \*\* end ()**
- const **ChildType \*const \* begin () const**
- const **ChildType \*const \* end () const**
- virtual const char \* **getGeometryName ()** const override  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType getGeometryType ()** const override  
*Fetch geometry type.*
- **OGRErr importFromWkt (const char \*\*)** override
- virtual **OGRErr exportToWkt (char \*\*, OGRwkbVariant= wkbVariantOldOgc)** const override  
*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 override  
*Get the dimension of this object.*
- virtual **OGRBoolean hasCurveGeometry (int bLookForNonLinear=FALSE)** const override  
*Returns if this geometry is or has curve geometry.*
- **OGRGeometryCollection \* toUpperClass ()**
- const **OGRGeometryCollection \* toUpperClass () const**
- virtual void **accept ( IOGRGeometryVisitor \*visitor)** override
- virtual void **accept ( IOGRConstGeometryVisitor \*visitor)** const override
- virtual **OGRErr importFromWkt (const char \*\*ppszInput)=0**  
*Assign geometry from well known text data.*
- **OGRErr importFromWkt (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")**

## Static Public Member Functions

- static **OGRMultiPolygon \* CastToMultiPolygon ( OGRMultiSurface \*poMS)**  
*Cast to multipolygon.*

## Protected Member Functions

- virtual **OGRBoolean isCompatibleSubType ( OGRwkbGeometryType)** const override

### 11.83.1 Detailed Description

A collection of non-overlapping **OGRSurface** (p. ??).

Since

GDAL 2.0

## 11.83.2 Member Typedef Documentation

### 11.83.2.1 ChildType

```
typedef OGRSurface OGRMultiSurface::ChildType
```

Type of child elements.

## 11.83.3 Constructor & Destructor Documentation

### 11.83.3.1 OGRMultiSurface()

```
OGRMultiSurface::OGRMultiSurface (
    const OGRMultiSurface & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

## 11.83.4 Member Function Documentation

### 11.83.4.1 accept() [1/2]

```
virtual void OGRMultiSurface::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

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

References IOGRGeometryVisitor::visit().



11.83.4.2 `accept()` [2/2]

```
virtual void OGRMultiSurface::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

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

References IOGRConstGeometryVisitor::visit().

11.83.4.3 `begin()` [1/2]

```
ChildType** OGRMultiSurface::begin ( ) [inline]
```

Return begin of iterator.

Since

GDAL 2.3

11.83.4.4 `begin()` [2/2]

```
const ChildType* const* OGRMultiSurface::begin ( ) const [inline]
```

Return begin of iterator.

Since

GDAL 2.3

11.83.4.5 `CastToMultiPolygon()`

```
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

<i>poMS</i>	the input geometry - ownership is passed to the method.
-------------	---

## Returns

new geometry.

Referenced by OGRGeometryFactory::forceToMultiPolygon().

11.83.4.6 `end()` [1/2]

```
ChildType** OGRMultiSurface::end ( ) [inline]
```

Return end of iterator

11.83.4.7 `end()` [2/2]

```
const ChildType* const* OGRMultiSurface::end ( ) const [inline]
```

Return end of iterator

11.83.4.8 `exportToWkt()`

```
OGRERR OGRMultiSurface::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

**Returns**

Currently OGRERR\_NONE is always returned.

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

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

References wkbVariantIso.

**11.83.4.9 getDimension()**

```
int OGRMultiSurface::getDimension ( ) const [override], [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. ??).

**11.83.4.10 getGeometryName()**

```
const char * OGRMultiSurface::getGeometryName ( ) const [override], [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. ??).

11.83.4.11 `getGeometryType()`

```
OGRwkbGeometryType OGRMultiSurface::getGeometryType ( ) const [override], [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 `wkbMultiSurface`, `wkbMultiSurfaceM`, `wkbMultiSurfaceZ`, and `wkbMultiSurfaceZM`.

11.83.4.12 `hasCurveGeometry()`

```
OGRBoolean OGRMultiSurface::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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**

<i>bLookForNonLinear</i>	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().

Referenced by OGRGeometryFactory::forceToMultiPolygon().

#### 11.83.4.13 importFromWkt() [1/3]

**OGRERR** OGRGeometry::importFromWkt [inline]

Deprecated.

**Deprecated** in GDAL 2.3

#### 11.83.4.14 importFromWkt() [2/3]

**OGRERR** OGRGeometry::importFromWkt

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

#### Parameters

<i>ppsInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
-----------------	---

#### Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

#### 11.83.4.15 importFromWkt() [3/3]

```
OGRERR OGRMultiSurface::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated < Success

< Success

< Success

< Corrupt data

< Corrupt data

< Success

< Success

< Success

< Success

< Corrupt data  
< Success

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

References CPLError(), OGRGeometryFactory::createFromWkt(), EQUAL, OGRERR\_CORRUPT\_DATA, OGRERR\_NONE, STARTS\_WITH\_CI, and OGRGeometry::toSurface().

#### 11.83.4.16 isCompatibleSubType()

```
OGRBoolean OGRMultiSurface::isCompatibleSubType (
    OGRwkbGeometryType ) const [override], [protected], [virtual]
```

Returns whether a geometry of the specified geometry type can be a member of this collection.

##### Parameters

<i>eSubType</i>	type of the potential member
-----------------	------------------------------

##### Returns

TRUE or FALSE

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

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

References wkbCurvePolygon, wkbFlatten, and wkbPolygon.

#### 11.83.4.17 operator=()

```
OGRMultiSurface & OGRMultiSurface::operator= (
    const OGRMultiSurface & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

##### Since

GDAL 2.1

References OGRGeometryCollection::operator=().

Referenced by OGRMultiPolygon::operator=().

#### 11.83.4.18 PointOnSurface()

```
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

<i>poPoint</i>	point to be set with an internal point.
----------------	---

## Returns

OGRERR\_NONE if it succeeds or OGRERR\_FAILURE otherwise.

**11.83.4.19 toUpperClass()** [1/2]

```
OGRGeometryCollection* OGRMultiSurface::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

**11.83.4.20 toUpperClass()** [2/2]

```
const OGRGeometryCollection* OGRMultiSurface::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

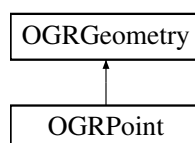
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrmultisurface.cpp

## 11.84 OGRPoint Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRPoint:





## Public Member Functions

- **OGRPoint** ()  
*Create an empty point.*
- **OGRPoint** (double x, double y)  
*Create a point.*
- **OGRPoint** (double x, double y, double z)  
*Create a point.*
- **OGRPoint** (double x, double y, double z, double m)  
*Create a point.*
- **OGRPoint** (const **OGRPoint** &other)  
*Copy constructor.*
- **OGRPoint** & **operator=** (const **OGRPoint** &other)  
*Assignment operator.*
- int **WkbSize** () const override  
*Returns size of related binary representation.*
- **OGRErr importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumedOut) override  
*Assign geometry from well known binary data.*
- **OGRErr exportToWkb** ( **OGRwkbByteOrder**, unsigned char \*, **OGRwkbVariant= wkbVariantOldOgc**) const override  
*Convert a geometry into well known binary format.*
- **OGRErr importFromWkt** (const char \*\*) override
- **OGRErr exportToWkt** (char \*\*pszDstText, **OGRwkbVariant= wkbVariantOldOgc**) const override  
*Convert a geometry into well known text format.*
- virtual int **getDimension** () const override  
*Get the dimension of this object.*
- virtual **OGRGeometry** \* **clone** () const override  
*Make a copy of this object.*
- virtual void **empty** () override  
*Clear geometry information. This restores the geometry to its initial state after construction, and before assignment of actual geometry.*
- virtual void **getEnvelope** (OGREnvelope \*psEnvelope) const override  
*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (OGREnvelope3D \*psEnvelope) const override  
*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- virtual **OGRBoolean isEmpty** () const override  
*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.*
- double **getM** () const
- virtual void **setCoordinateDimension** (int nDimension) override  
*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).*
- void **setM** (double mIn)
- virtual **OGRBoolean Equals** (const **OGRGeometry** \*) const override  
*Returns TRUE if two geometries are equivalent.*
- virtual **OGRBoolean Intersects** (const **OGRGeometry** \*) const override  
*Do these features intersect?*
- virtual **OGRBoolean Within** (const **OGRGeometry** \*) const override  
*Test for containment.*
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType getGeometryType** () const override  
*Fetch geometry type.*
- virtual **OGRerr transform** ( **OGRCoordinateTransformation** \*poCT) override  
*Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** () override  
*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override
- virtual void **swapXY** () override  
*Swap x and y coordinates.*
- virtual **OGRerr importFromWkt** (const char \*\*ppszInput)=0  
*Assign geometry from well known text data.*
- **OGRerr importFromWkt** (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

## Additional Inherited Members

### 11.84.1 Detailed Description

Point class.

Implements SFCOM IPoint methods.

### 11.84.2 Constructor & Destructor Documentation

#### 11.84.2.1 OGRPoint() [1/4]

```
OGRPoint::OGRPoint (
    double xIn,
    double yIn )
```

Create a point.

#### Parameters

$x \leftrightarrow$ In	x
$y \leftrightarrow$ In	y

## 11.84.2.2 OGRPoint() [2/4]

```
OGRPoint::OGRPoint (
    double xIn,
    double yIn,
    double zIn )
```

Create a point.

## Parameters

$x \leftrightarrow$ <i>In</i>	x
$y \leftrightarrow$ <i>In</i>	y
$z \leftrightarrow$ <i>In</i>	z

## 11.84.2.3 OGRPoint() [3/4]

```
OGRPoint::OGRPoint (
    double xIn,
    double yIn,
    double zIn,
    double mIn )
```

Create a point.

## Parameters

<i>xIn</i>	x
<i>yIn</i>	y
<i>zIn</i>	z
$m \leftrightarrow$ <i>In</i>	m

## 11.84.2.4 OGRPoint() [4/4]

```
OGRPoint::OGRPoint (
    const OGRPoint & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

### 11.84.3 Member Function Documentation

#### 11.84.3.1 `accept()` [1/2]

```
virtual void OGRPoint::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRGeometryVisitor::visit().

#### 11.84.3.2 `accept()` [2/2]

```
virtual void OGRPoint::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

References IOGRConstGeometryVisitor::visit().

#### 11.84.3.3 `clone()`

```
OGRGeometry * OGRPoint::clone ( ) const [override], [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(), and OGRPoint().

## 11.84.3.4 empty()

```
void OGRPoint::empty ( ) [override], [virtual]
```

Clear geometry information. This restores the geometry to its 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. ??).

## 11.84.3.5 Equals()

```
OGRBoolean OGRPoint::Equals (
    const OGRGeometry * ) const [override], [virtual]
```

Returns TRUE if two geometries are equivalent.

This operation implements the SQL/MM ST\_OrderingEquals() operation.

The comparison is done in a structural way, that is to say that the geometry types must be identical, as well as the number and ordering of sub-geometries and vertices. Or equivalently, two geometries are considered equal by this method if their WKT/WKB representation is equal. Note: this must be distinguished for equality in a spatial way (which is the purpose of the ST\_Equals() operation).

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(), IsEmpty(), and OGRGeometry::toPoint().

Referenced by OGRGeometryFactory::forceToLineString().

## 11.84.3.6 exportToWkb()

```
OGRERR OGRPoint::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
<i>eWkbVariant</i>	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.

< Success

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

References CPL\_LSBPTR32, CPL\_MSBPTR32, CPL\_SWAPDOUBLE, getGeometryType(), OGRGeometry::getIsoGeometryType(), OGRGeometry::Is3D(), IsEmpty(), OGRGeometry::IsMeasured(), OGRERR\_NONE, wkbFlatten, wkbNDR, wkbVariantIso, and wkbVariantPostGIS1.

## 11.84.3.7 exportToWkt()

```
OGRERR OGRPoint::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

## Returns

Currently OGRERR\_NONE is always returned.

< Success

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

References CPLStrdup(), OGRGeometry::Is3D(), IsEmpty(), OGRGeometry::IsMeasured(), OGRERR\_NONE, and wkbVariantIso.

#### 11.84.3.8 flattenTo2D()

```
void OGRPoint::flattenTo2D ( ) [override], [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. ??).

References OGRGeometry::setMeasured().

Referenced by setCoordinateDimension().

#### 11.84.3.9 getDimension()

```
int OGRPoint::getDimension ( ) const [override], [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. ??).

#### 11.84.3.10 getEnvelope() [1/2]

```
void OGRPoint::getEnvelope (
    OGREnvelope * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

**Parameters**

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

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

References getX(), and getY().

**11.84.3.11 getEnvelope()** [2/2]

```
void OGRPoint::getEnvelope (
    OGREnvelope3D * psEnvelope ) const  [override], [virtual]
```

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

**Parameters**

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

**Since**

OGR 1.9.0

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

References getX(), getY(), and getZ().

**11.84.3.12 getGeometryName()**

```
const char * OGRPoint::getGeometryName ( ) const  [override], [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. ??).



## 11.84.3.13 getGeometryType()

```
OGRwkbGeometryType OGRPoint::getGeometryType ( ) const [override], [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 **wkbPoint**, **wkbPoint25D**, **wkbPointM**, and **wkbPointZM**.

Referenced by **Equals()**, and **exportToWkb()**.

## 11.84.3.14 getM()

```
double OGRPoint::getM ( ) const [inline]
```

Return m

Referenced by **OGRSimpleCurve::addPoint()**, **OGR\_G\_GetPointsZM()**, **OGR\_G\_GetPointZM()**, and **OGRSimpleCurve::setPoint()**.

## 11.84.3.15 getX()

```
double OGRPoint::getX ( ) const [inline]
```

Fetch X coordinate.

Return x

Relates to the **SFCOM IPoint::get\_X()** method.

**Returns**

the X coordinate of this point.

Referenced by **OGRSimpleCurve::addPoint()**, **OGRGeometryFactory::curveFromLineString()**, **Equals()**, **OGRCompoundCurve::get\_Area()**, **OGRCurve::get\_IsClosed()**, **getEnvelope()**, **OGRCurve::IsConvex()**, **OGRLinearRing::isPointInRing()**, **OGRLinearRing::isPointOnRingBoundary()**, **OGR\_G\_GetPoint()**, **OGR\_G\_GetPoints()**, **OGR\_G\_GetPointsZM()**, **OGR\_G\_GetPointZM()**, **OGRSimpleCurve::setPoint()**, and **OGRGeometryFactory::transformWithOptions()**.

### 11.84.3.16 getY()

```
double OGRPoint::getY ( ) const [inline]
```

Fetch Y coordinate.

Return y

Relates to the SFCOM IPoint::get\_Y() method.

#### Returns

the Y coordinate of this point.

Referenced by OGRSimpleCurve::addPoint(), OGRGeometryFactory::curveFromLineString(), Equals(), OGRCompoundCurve::get\_Area(), OGRCurve::get\_IsClosed(), getEnvelope(), OGRCurve::IsConvex(), OGRLinearRing::isPointInRing(), OGRLinearRing::isPointOnRingBoundary(), OGR\_G\_GetPoint(), OGR\_G\_GetPoints(), OGR\_G\_GetPointsZM(), OGR\_G\_GetPointZM(), and OGRSimpleCurve::setPoint().

### 11.84.3.17 getZ()

```
double OGRPoint::getZ ( ) const [inline]
```

Fetch Z coordinate.

Return z

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(), OGRCurve::get\_IsClosed(), getEnvelope(), OGR\_G\_GetPoint(), OGR\_G\_GetPoints(), OGR\_G\_GetPointsZM(), OGR\_G\_GetPointZM(), and OGRSimpleCurve::setPoint().

### 11.84.3.18 importFromWkb()

```
OGRERR OGRPoint::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	if wkbVariantPostGIS1, special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

## Since

GDAL 2.3

< Success

< Not enough data to deserialize

< Not enough data to deserialize

< Not enough data to deserialize

< Success

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

References CPL\_SWAPDOUBLE, OGRERR\_NONE, OGRERR\_NOT\_ENOUGH\_DATA, and wkbNDR.

## 11.84.3.19 importFromWkt() [1/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

## 11.84.3.20 importFromWkt() [2/3]

```
OGRERR OGRGeometry::importFromWkt
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

## 11.84.3.21 importFromWkt() [3/3]

```
OGRERR OGRPoint::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated < Success

< Success

< Corrupt data

< Success

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

References CPLFree, OGRERR\_CORRUPT\_DATA, OGRERR\_NONE, OGRRawPoint::x, and OGRRawPoint::y.

## 11.84.3.22 Intersects()

```
OGRBoolean OGRPoint::Intersects (
    const OGRGeometry * poOtherGeom ) const [override], [virtual]
```

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigorous fashion otherwise TRUE is returned if the envelopes (bounding boxes) of the two geometries overlap.

The poOtherGeom argument may be safely NULL, but in this case the method will always return TRUE. That is, a NULL geometry is treated as being everywhere.

This method is the same as the C function **OGR\_G\_Intersects()** (p. ??).

## Parameters

<i>poOtherGeom</i>	the other geometry to test against.
--------------------	-------------------------------------

**Returns**

TRUE if the geometries intersect, otherwise FALSE.

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

References **OGRGeometry::getGeometryType()**, **OGRGeometry::Intersects()**, **OGRCurvePolygon::Intersects()**, **IsEmpty()**, **OGRGeometry::toCurvePolygon()**, **wkbCurvePolygon**, and **wkbFlatten**.

**11.84.3.23 IsEmpty()**

```
virtual OGRBoolean OGRPoint::IsEmpty ( ) const [inline], [override], [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()**, **exportToWkb()**, **exportToWkt()**, **Intersects()**, **OGRLinearRing::isPointInRing()**, and **Within()**.

**11.84.3.24 operator=()**

```
OGRPoint & OGRPoint::operator= (
    const OGRPoint & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

**Since**

GDAL 2.1

References **OGRGeometry::operator=()**.

**11.84.3.25 setCoordinateDimension()**

```
void OGRPoint::setCoordinateDimension (
    int nNewDimension ) [override], [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, a compound curve, a polygon, etc. will affect the children geometries. This will also remove the M dimension if present before this call.

**Deprecated** use **set3D()** (p. ??) or **setMeasured()** (p. ??).

**Parameters**

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

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

References `flattenTo2D()`, and `OGRGeometry::setMeasured()`.

**11.84.3.26 setM()**

```
void OGRPoint::setM (
    double mIn ) [inline]
```

Set m

**Parameters**

<i>m</i> ↔ <i>In</i>	m
-------------------------	---

Referenced by `OGRSimpleCurve::getPoint()`, `OGRMultiPoint::importFromWkt()`, `OGR_G_AddPointM()`, `OGR_G_↔_AddPointZM()`, `OGR_G_SetPointM()`, and `OGR_G_SetPointZM()`.

**11.84.3.27 setX()**

```
void OGRPoint::setX (
    double xIn ) [inline]
```

Assign point X coordinate.

Set x

**Parameters**

<i>x</i> ↔ <i>In</i>	x
-------------------------	---

There is no corresponding SFCOM method.

Referenced by `OGRSimpleCurve::getPoint()`, `OGRCurve::isConvex()`, `OGR_G_AddPoint()`, `OGR_G_AddPoint_↔_2D()`, `OGR_G_AddPointM()`, `OGR_G_AddPointZM()`, `OGR_G_SetPoint()`, `OGR_G_SetPoint_2D()`, `OGR_G_Set_↔_PointM()`, `OGR_G_SetPointZM()`, `OGRGeometryFactory::transformWithOptions()`, `OGRSimpleCurve::Value()`, and `OGRCircularString::Value()`.

**11.84.3.28 setY()**

```
void OGRPoint::setY (
    double yIn ) [inline]
```

Assign point Y coordinate.

Set y

**Parameters**

$y \leftrightarrow$ <i>In</i>	y
----------------------------------	---

There is no corresponding SFCOM method.

Referenced by OGRSimpleCurve::getPoint(), OGRCurve::IsConvex(), OGR\_G\_AddPoint(), OGR\_G\_AddPoint\_↔2D(), OGR\_G\_AddPointM(), OGR\_G\_AddPointZM(), OGR\_G\_SetPoint(), OGR\_G\_SetPoint\_2D(), OGR\_G\_Set\_↔PointM(), OGR\_G\_SetPointZM(), OGRSimpleCurve::Value(), and OGRCircularString::Value().

**11.84.3.29 setZ()**

```
void OGRPoint::setZ (
    double zIn ) [inline]
```

Assign point Z coordinate. Calling this method will force the geometry coordinate dimension to 3D (wkbPoint|wkbZ).

Set z

**Parameters**

$z \leftrightarrow$ <i>In</i>	z
----------------------------------	---

There is no corresponding SFCOM method.

Referenced by OGRSimpleCurve::getPoint(), OGRMultiPoint::importFromWkt(), OGR\_G\_AddPoint(), OGR\_G\_↔AddPointZM(), OGR\_G\_SetPoint(), OGR\_G\_SetPointZM(), OGRSimpleCurve::Value(), and OGRCircularString::↔Value().

**11.84.3.30 swapXY()**

```
void OGRPoint::swapXY ( ) [override], [virtual]
```

Swap x and y coordinates.

Since

OGR 1.8.0

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

## 11.84.3.31 transform()

```
OGRERR OGRPoint::transform (
    OGRCoordinateTransformation * poCT ) [override], [virtual]
```

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGR↵SpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

## Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

## Returns

OGRERR\_NONE on success or an error code.

< Success

< Failure

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

References **OGRGeometry::assignSpatialReference()**, **OGRCoordinateTransformation::GetTargetCS()**, **OGRERR↵R\_FAILURE**, **OGRERR\_NONE**, and **OGRCoordinateTransformation::Transform()**.

## 11.84.3.32 Within()

```
OGRBoolean OGRPoint::Within (
    const OGRGeometry * ) const [override], [virtual]
```

Test for containment.

Tests if actual geometry object is within the passed geometry.

This method is the same as the C function **OGR\_G\_Within()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a **CPLE\_NotSupported** error.

## Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---



**Returns**

TRUE if poOtherGeom is within this geometry, otherwise FALSE.

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

References OGRCurvePolygon::Contains(), OGRGeometry::getGeometryType(), IsEmpty(), OGRGeometry::toCurvePolygon(), OGRGeometry::Within(), wkbCurvePolygon, and wkbFlatten.

**11.84.3.33 WkbSize()**

```
int OGRPoint::WkbSize ( ) const [override], [virtual]
```

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR\_G\_WkbSize()** (p. ??).

**Returns**

size of binary representation in bytes.

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

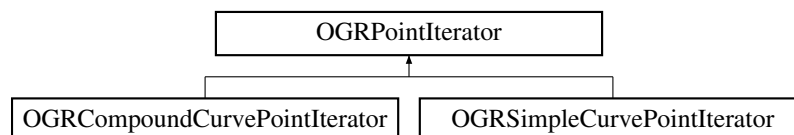
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrpoint.cpp

**11.85 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.*

### 11.85.1 Detailed Description

Interface for a point iterator.

Since

GDAL 2.0

### 11.85.2 Member Function Documentation

#### 11.85.2.1 destroy()

```
void OGRPointIterator::destroy (
    OGRPointIterator * poIter ) [static]
```

Destroys a point iterator.

Since

GDAL 2.0

#### 11.85.2.2 getNextPoint()

```
OGRBoolean OGRPointIterator::getNextPoint (
    OGRPoint * p ) [pure virtual]
```

Returns the next point followed by the iterator.

Parameters

<i>p</i>	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 OGRCompoundCurve::IsConvex().

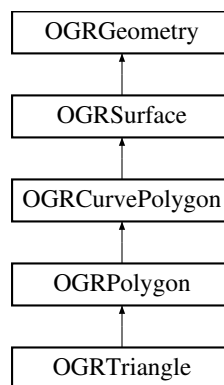
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrcurve.cpp**

## 11.86 OGRPolygon Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRPolygon:



### Public Types

- typedef **OGRLinearRing** **ChildType**

### Public Member Functions

- **OGRPolygon** ()  
*Create an empty polygon.*
- **OGRPolygon** (const **OGRPolygon** &other)  
*Copy constructor.*
- **OGRPolygon** & **operator=** (const **OGRPolygon** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual const char \* **getGeometryName** () const override

- Fetch WKT name for geometry type.*

  - virtual **OGRwkbGeometryType** **getGeometryType** () const override

*Fetch geometry type.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override

*Returns if this geometry is or has curve geometry.*
- virtual **OGRGeometry** \* **getCurveGeometry** (const char \*const \*papszOptions=nullptr) const override

*Return curve version of this geometry.*
- virtual **OGRGeometry** \* **getLinearGeometry** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*papszOptions=nullptr) const override

*Return, possibly approximate, non-curve version of this geometry.*
- virtual int **WkbSize** () const override

*Returns size of related binary representation.*
- virtual **OGRERR** **importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumedOut) override

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

*Convert a geometry into well known binary format.*
- **OGRERR** **importFromWkt** (const char \*\*) override
- virtual **OGRERR** **exportToWkt** (char \*\*papszDstText, **OGRwkbVariant**= **wkbVariantOldOgc**) const override

*Convert a geometry into well known text format.*
- virtual **OGRPolygon** \* **CurvePolyToPoly** (double dfMaxAngleStepSizeDegrees=0, const char \*const \*papszOptions=nullptr) const override

*Return a polygon from a curve polygon.*
- **OGRLinearRing** \* **getExteriorRing** ()

*Fetch reference to external polygon ring.*
- const **OGRLinearRing** \* **getExteriorRing** () const

*Fetch reference to external polygon ring.*
- virtual **OGRLinearRing** \* **getInteriorRing** (int)

*Fetch reference to indicated internal ring.*
- virtual const **OGRLinearRing** \* **getInteriorRing** (int) const

*Fetch reference to indicated internal ring.*
- **OGRLinearRing** \* **stealExteriorRing** ()

*"Steal" reference to external polygon ring.*
- virtual **OGRLinearRing** \* **stealInteriorRing** (int)

*"Steal" reference to indicated interior ring.*
- **OGRBoolean** **IsPointOnSurface** (const **OGRPoint** \*) const
- **OGRCurvePolygon** \* **toUpperClass** ()
- const **OGRCurvePolygon** \* **toUpperClass** () const
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override
- virtual void **closeRings** () override

*Force rings to be closed.*
- virtual **OGRERR** **importFromWkt** (const char \*\*papszInput)=0

*Assign geometry from well known text data.*
- **OGRERR** **importFromWkt** (char \*\*papszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

## Additional Inherited Members

### 11.86.1 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.

### 11.86.2 Member Typedef Documentation

#### 11.86.2.1 ChildType

```
typedef OGRLinearRing OGRPolygon::ChildType
```

Type of child elements.

### 11.86.3 Constructor & Destructor Documentation

#### 11.86.3.1 OGRPolygon()

```
OGRPolygon::OGRPolygon (
    const OGRPolygon & other )
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

### 11.86.4 Member Function Documentation

#### 11.86.4.1 `accept()` [1/2]

```
virtual void OGRPolygon::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

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

References IOGRGeometryVisitor::visit().

#### 11.86.4.2 `accept()` [2/2]

```
virtual void OGRPolygon::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

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

References IOGRConstGeometryVisitor::visit().

#### 11.86.4.3 `begin()` [1/2]

```
ChildType** OGRPolygon::begin ( ) [inline]
```

Return begin of iterator.

Since

GDAL 2.3

#### 11.86.4.4 `begin()` [2/2]

```
const ChildType* const* OGRPolygon::begin ( ) const [inline]
```

Return begin of iterator.

Since

GDAL 2.3

## 11.86.4.5 closeRings()

```
void OGRPolygon::closeRings ( ) [override], [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. ??).

## 11.86.4.6 CurvePolyToPoly()

```
OGRPolygon * OGRPolygon::CurvePolyToPoly (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	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(), and OGRGeometry::toPolygon().

## 11.86.4.7 end() [1/2]

```
ChildType** OGRPolygon::end ( ) [inline]
```

Return end of iterator

#### 11.86.4.8 end() [2/2]

```
const ChildType* const* OGRPolygon::end ( ) const [inline]
```

Return end of iterator

#### 11.86.4.9 exportToWkb()

```
OGRERR OGRPolygon::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
<i>eWkbVariant</i>	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.

< Success

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

References CPL\_SWAP32, getGeometryType(), OGRGeometry::getIsoGeometryType(), OGRGeometry::Is3D(), OGRGeometry::IsMeasured(), OGRERR\_NONE, wkbFlatten, wkbVariantIso, and wkbVariantPostGIS1.

#### 11.86.4.10 exportToWkt()

```
OGRERR OGRPolygon::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).



## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

## Returns

Currently OGRERR\_NONE is always returned.

< Success

< Success

< Not enough memory

< Success

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

References CPLAssert, CPLCalloc(), CPLDebug(), CPLFree, CPLStrdup(), OGRSimpleCurve::exportToWkt(), getExteriorRing(), getGeometryName(), OGRSimpleCurve::getNumPoints(), OGRGeometry::Is3D(), OGRCurvePolygon::IsEmpty(), OGRGeometry::IsMeasured(), OGRERR\_NONE, OGRERR\_NOT\_ENOUGH\_MEMORY, OGRSimpleCurve::set3D(), OGRSimpleCurve::setMeasured(), STARTS\_WITH\_CI, OGRGeometry::toLinearRing(), VSI\_MALLOC\_VERBOSE, VSIFree(), and wkbVariantIso.

## 11.86.4.11 getCurveGeometry()

```
OGRGeometry * OGRPolygon::getCurveGeometry (
    const char *const * ppszOptions = nullptr ) const [override], [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

<i>ppszOptions</i>	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()**, **OGRCurvePolygon::assignSpatialReference()**, **OGRCurvePolygon::clone()**, **OGRGeometry::getSpatialReference()**, **OGRCurvePolygon::OGRCurvePolygon()**, **wkbFlatten**, and **wkbLineString**.

**11.86.4.12 getExteriorRing()** [1/2]

```
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.

References **OGRGeometry::toLinearRing()**.

Referenced by **exportToWkt()**, **OGRGeometryFactory::forceTo()**, **OGRGeometryFactory::forceToMultiLineString()**, and **OGRGeometryFactory::forceToPolygon()**.

**11.86.4.13 getExteriorRing()** [2/2]

```
const OGRLinearRing * OGRPolygon::getExteriorRing ( ) const
```

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.

References **OGRGeometry::toLinearRing()**.

## 11.86.4.14 getGeometryName()

```
const char * OGRPolygon::getGeometryName ( ) const [override], [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. ??).

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

Referenced by exportToWkt().

## 11.86.4.15 getGeometryType()

```
OGRwkbGeometryType OGRPolygon::getGeometryType ( ) const [override], [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. ??).

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

References wkbPolygon, wkbPolygon25D, wkbPolygonM, and wkbPolygonZM.

Referenced by exportToWkb().

## 11.86.4.16 getInteriorRing() [1/2]

```
OGRLinearRing * OGRPolygon::getInteriorRing (
    int iRing ) [virtual]
```

Fetch reference to indicated internal ring.

Note that the returned ring pointer is to an internal data object of the **OGRPolygon** (p. ??). It should not be modified or deleted by the application, and the pointer is only valid till the polygon is next modified. Use the **OGRGeometry::clone()** (p. ??) method to make a separate copy within the application.

Relates to the SFCOM IPolygon::get\_InternalRing() method.

## Parameters

<i>iRing</i>	internal ring index from 0 to <b>getNumInteriorRings()</b> (p. ??) - 1.
--------------	---

## Returns

pointer to interior ring. May be NULL.

References **OGRGeometry::toLinearRing()**.

Referenced by **OGRGeometryFactory::forceToMultiLineString()**.

11.86.4.17 **getInteriorRing()** [2/2]

```
const OGRLinearRing * OGRPolygon::getInteriorRing (
    int iRing ) const [virtual]
```

Fetch reference to indicated internal ring.

Note that the returned ring pointer is to an internal data object of the **OGRPolygon** (p. ??). It should not be modified or deleted by the application, and the pointer is only valid till the polygon is next modified. Use the **OGRGeometry::clone()** (p. ??) method to make a separate copy within the application.

Relates to the SFCOM IPolygon::get\_InternalRing() method.

## Parameters

<i>iRing</i>	internal ring index from 0 to <b>getNumInteriorRings()</b> (p. ??) - 1.
--------------	---

## Returns

pointer to interior ring. May be NULL.

References **OGRGeometry::toLinearRing()**.

11.86.4.18 **getLinearGeometry()**

```
OGRGeometry * OGRPolygon::getLinearGeometry (
    double dfMaxAngleStepSizeDegrees = 0,
    const char *const * papszOptions = nullptr ) const [override], [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

<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings. See <b>OGRGeometryFactory::curveToLineString()</b> (p. ??) for valid options.

## Returns

a new geometry.

## Since

GDAL 2.0

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

References **OGRGeometry::getLinearGeometry()**.

## 11.86.4.19 hasCurveGeometry()

```
OGRBoolean OGRPolygon::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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

<i>bLookForNonLinear</i>	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. ??).

## 11.86.4.20 importFromWkb()

```

OGRERR OGRPolygon::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]

```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	if wkbVariantPostGIS1, special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

## Since

GDAL 2.3

< Success

< Success

< Success

< Success

< Success

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

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

References CPLAssert, OGRERR\_NONE, and wkbNDR.

Referenced by OGRTriangle::importFromWkb().

11.86.4.21 `importFromWkt()` [1/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

11.86.4.22 `importFromWkt()` [2/3]

```
OGRERR OGRGeometry::importFromWkt
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

11.86.4.23 `importFromWkt()` [3/3]

```
OGRERR OGRPolygon::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated < Success  
< Success

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

References CPLFree, and OGRERR\_NONE.

#### 11.86.4.24 IsPointOnSurface()

```
OGRBoolean OGRPolygon::IsPointOnSurface (
    const OGRPoint * pt ) const
```

Return whether the point is on the surface.

##### Returns

TRUE or FALSE

#### 11.86.4.25 operator=()

```
OGRPolygon & OGRPolygon::operator= (
    const OGRPolygon & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

##### Since

GDAL 2.1

References OGRCurvePolygon::operator=().

Referenced by OGRTriangle::operator=().

#### 11.86.4.26 stealExteriorRing()

```
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(), and OGRGeometry::toLinearRing().

Referenced by OGRGeometryFactory::forceToPolygon().

#### 11.86.4.27 stealInteriorRing()

```
OGRLinearRing * OGRPolygon::stealInteriorRing (
    int iRing ) [virtual]
```

"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

<i>iRing</i>	internal ring index from 0 to <b>getNumInteriorRings()</b> (p. ??) - 1.
--------------	---

## Returns

pointer to interior ring. May be NULL.

References OGRGeometry::toLinearRing().

Referenced by OGRGeometryFactory::forceToPolygon().

## 11.86.4.28 toUpperClass() [1/2]

```
OGRCurvePolygon* OGRPolygon::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

## 11.86.4.29 toUpperClass() [2/2]

```
const OGRCurvePolygon* OGRPolygon::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

## 11.86.4.30 WkbSize()

```
int OGRPolygon::WkbSize ( ) const [override], [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. ??).

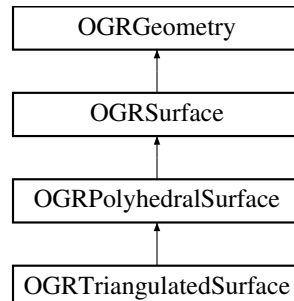
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrpolygon.cpp**

## 11.87 OGRPolyhedralSurface Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRPolyhedralSurface:



### Public Types

- typedef **OGRPolygon** **ChildType**

### Public Member Functions

- **OGRPolyhedralSurface** ()  
*Create an empty PolyhedralSurface.*
- **OGRPolyhedralSurface** (const **OGRPolyhedralSurface** &poGeom)  
*Copy constructor.*
- ~**OGRPolyhedralSurface** () override  
*Destructor.*
- **OGRPolyhedralSurface** & **operator=** (const **OGRPolyhedralSurface** &other)  
*Assignment operator.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- virtual int **WkbSize** () const override  
*Returns size of related binary representation.*
- virtual const char \* **getGeometryName** () const override  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Returns the WKB Type of PolyhedralSurface.*
- virtual **OGRwkbVariant** **importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumedOut) override  
*Assign geometry from well known binary data.*
- virtual **OGRwkbVariant** **exportToWkb** (**OGRwkbByteOrder**, unsigned char \*, **OGRwkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known binary format.*
- **OGRwkbVariant** **importFromWkt** (const char \*\*) override
- virtual **OGRwkbVariant** **exportToWkt** (char \*\*ppszDstText, **OGRwkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known text format.*
- virtual int **getDimension** () const override

- Get the dimension of this object.*

  - virtual void **empty** () override

*Clear geometry information. This restores the geometry to its initial state after construction, and before assignment of actual geometry.*
- virtual **OGRGeometry** \* **clone** () const override

*Make a copy of this object.*
- virtual void **getEnvelope** (OGREnvelope \*psEnvelope) const override

*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (OGREnvelope3D \*psEnvelope) const override

*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- virtual void **flattenTo2D** () override

*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual **OGRERR** **transform** ( **OGRCoordinateTransformation** \*) override

*Apply arbitrary coordinate transformation to geometry.*
- virtual **OGRBoolean** **Equals** (const **OGRGeometry** \*) const override

*Returns TRUE if two geometries are equivalent.*
- virtual double **get\_Area** () const override

*Returns the area enclosed.*
- virtual **OGRERR** **PointOnSurface** ( **OGRPoint** \*) const override

*This method relates to the SFCOM ISurface::get\_PointOnSurface() method.*
- virtual **OGRBoolean** **hasCurveGeometry** (int bLookForNonLinear=FALSE) const override

*Returns if this geometry is or has curve geometry.*
- virtual **OGRERR** **addGeometry** (const **OGRGeometry** \*)

*Add a new geometry to a collection.*
- **OGRERR** **addGeometryDirectly** ( **OGRGeometry** \*poNewGeom)

*Add a geometry directly to the container.*
- int **getNumGeometries** () const

*Fetch number of geometries in PolyhedralSurface.*
- **OGRGeometry** \* **getGeometryRef** (int i)

*Fetch geometry from container.*
- const **OGRGeometry** \* **getGeometryRef** (int i) const

*Fetch geometry from container.*
- virtual **OGRBoolean** **IsEmpty** () const override

*Checks if the PolyhedralSurface is empty.*
- virtual void **setCoordinateDimension** (int nDimension) override

*Set the coordinate dimension.*
- virtual void **set3D** ( **OGRBoolean** bls3D) override

*Set the type as 3D geometry.*
- virtual void **setMeasured** ( **OGRBoolean** blsMeasured) override

*Set the type as Measured.*
- virtual void **swapXY** () override

*Swap x and y coordinates.*
- **OGRERR** **removeGeometry** (int iIndex, int bDelete=TRUE)

*Remove a geometry from the container.*
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override
- virtual void **assignSpatialReference** ( **OGRSpatialReference** \*poSR) override

*Assign spatial reference to this object.*
- virtual **OGRERR** **importFromWkt** (const char \*\*ppszInput)=0

*Assign geometry from well known text data.*
- **OGRERR** **importFromWkt** (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

## Static Public Member Functions

- static **OGRMultiPolygon** \* **CastToMultiPolygon** ( **OGRPolyhedralSurface** \*poPS)  
*Casts the **OGRPolyhedralSurface** (p. ??) to an **OGRMultiPolygon** (p. ??).*

### 11.87.1 Detailed Description

PolyhedralSurface class.

Since

GDAL 2.2

### 11.87.2 Member Typedef Documentation

#### 11.87.2.1 ChildType

```
typedef OGRPolygon OGRPolyhedralSurface::ChildType
```

Type of child elements.

### 11.87.3 Member Function Documentation

#### 11.87.3.1 **accept()** [1/2]

```
virtual void OGRPolyhedralSurface::accept (  
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

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

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

References **IOGRGeometryVisitor::visit()**.

11.87.3.2 `accept()` [2/2]

```
virtual void OGRPolyhedralSurface::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

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

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

References IOGRConstGeometryVisitor::visit().

11.87.3.3 `addGeometry()`

```
OGRERR OGRPolyhedralSurface::addGeometry (
    const OGRGeometry * poNewGeom ) [virtual]
```

Add a new geometry to a collection.

Only a POLYGON can be added to a POLYHEDRALSURFACE.

## Returns

OGRERR\_OGRERR\_NONE if the polygon is successfully added

< Unsupported geometry type

< Failure

< Success

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

References addGeometryDirectly(), OGRGeometry::clone(), OGRGeometry::getGeometryType(), OGRERR\_FAILURE, OGRERR\_NONE, and OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE.

Referenced by OGRTriangulatedSurface::addGeometry(), and OGRGeometryFactory::forceTo().

11.87.3.4 `addGeometryDirectly()`

```
OGRERR OGRPolyhedralSurface::addGeometryDirectly (
    OGRGeometry * poNewGeom )
```

Add a geometry directly to the container.

This method is the same as the C function **OGR\_G\_AddGeometryDirectly()** (p. ??).

There is no SFCOM analog to this method.

## Parameters

<i>poNewGeom</i>	geometry to add to the container.
------------------	-----------------------------------

## Returns

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

< Unsupported geometry type

< Failure

< Success

References OGRGeometry::getGeometryType(), OGRERR\_FAILURE, OGRERR\_NONE, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, and VSI\_REALLOC\_VERBOSE.

Referenced by addGeometry(), OGRTriangulatedSurface::addGeometry(), OGRTriangulatedSurface::CastToPolyhedralSurface(), and OGRGeometryFactory::forceTo().

## 11.87.3.5 assignSpatialReference()

```
void OGRPolyhedralSurface::assignSpatialReference (
    OGRSpatialReference * poSR ) [override], [virtual]
```

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.

Starting with GDAL 2.3, this will also assign the spatial reference to potential sub-geometries of the geometry (**OGRGeometryCollection** (p. ??), **OGRCurvePolygon**/**OGRPolygon**, **OGRCompoundCurve** (p. ??), **OGRPolyhedralSurface** (p. ??) and their derived classes).

This is similar to the SFCOM IGeometry::put\_SpatialReference() method.

This method is the same as the C function **OGR\_G\_AssignSpatialReference()** (p. ??).

## Parameters

<i>poSR</i>	new spatial reference system to apply.
-------------	--

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

References OGRGeometry::assignSpatialReference().

Referenced by OGRTriangulatedSurface::CastToPolyhedralSurface(), clone(), OGRGeometryFactory::forceTo(), and OGRTriangulatedSurface::operator=().

11.87.3.6 `begin()` [1/2]

```
ChildType** OGRPolyhedralSurface::begin ( ) [inline]
```

Return begin of iterator.

Since

GDAL 2.3

References OGRMultiPolygon::begin().

11.87.3.7 `begin()` [2/2]

```
const ChildType* const* OGRPolyhedralSurface::begin ( ) const [inline]
```

Return begin of iterator.

Since

GDAL 2.3

References OGRMultiPolygon::begin().

11.87.3.8 `CastToMultiPolygon()`

```
OGRMultiPolygon * OGRPolyhedralSurface::CastToMultiPolygon (
    OGRPolyhedralSurface * poPS ) [static]
```

Casts the **OGRPolyhedralSurface** (p. ??) to an **OGRMultiPolygon** (p. ??).

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

Parameters

<i>poPS</i>	the input geometry - ownership is passed to the method.
-------------	---

Returns

new geometry.

Referenced by OGRGeometryFactory::forceToMultiPolygon().

## 11.87.3.9 clone()

```
OGRGeometry * OGRPolyhedralSurface::clone ( ) const [override], [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.

< Success

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

References assignSpatialReference(), OGRGeometryFactory::createGeometry(), getGeometryType(), OGRGeometry::getSpatialReference(), OGRERR\_NONE, and OGRGeometry::toPolyhedralSurface().

## 11.87.3.10 empty()

```
void OGRPolyhedralSurface::empty ( ) [override], [virtual]
```

Clear geometry information. This restores the geometry to its 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 CPLFree.

Referenced by OGRTriangulatedSurface::operator=().

## 11.87.3.11 end() [1/2]

```
ChildType** OGRPolyhedralSurface::end ( ) [inline]
```

Return end of iterator

References OGRMultiPolygon::end().



11.87.3.12 `end()` [2/2]

```
const ChildType* const* OGRPolyhedralSurface::end ( ) const [inline]
```

Return end of iterator

References `OGRMultiPolygon::end()`.

11.87.3.13 `Equals()`

```
OGRBoolean OGRPolyhedralSurface::Equals (
    const OGRGeometry * ) const [override], [virtual]
```

Returns TRUE if two geometries are equivalent.

This operation implements the SQL/MM ST\_OrderingEquals() operation.

The comparison is done in a structural way, that is to say that the geometry types must be identical, as well as the number and ordering of sub-geometries and vertices. Or equivalently, two geometries are considered equal by this method if their WKT/WKB representation is equal. Note: this must be distinguished for equality in a spatial way (which is the purpose of the ST\_Equals() operation).

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()`, `IsEmpty()`, and `OGRGeometry::toPolyhedralSurface()`.

11.87.3.14 `exportToWkb()`

```
OGRERR OGRPolyhedralSurface::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
<i>eWkbVariant</i>	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.

< Success

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

References CPL\_SWAP32, OGRGeometry::getIsoGeometryType(), OGRERR\_NONE, and wkbVariantIso.

## 11.87.3.15 exportToWkt()

```
OGRErr OGRPolyhedralSurface::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li>wkbVariantOgc for old-style 99-402 extended dimension (Z) WKB types</li> <li>wkbVariantIso for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

## Returns

Currently OGRERR\_NONE is always returned.

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

References wkbVariantIso.

11.87.3.16 `flattenTo2D()`

```
void OGRPolyhedralSurface::flattenTo2D ( ) [override], [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. ??).

11.87.3.17 `get_Area()`

```
double OGRPolyhedralSurface::get_Area ( ) const [override], [virtual]
```

Returns the area enclosed.

This method is built on the SFCGAL library, check it for the definition of the geometry operation. If OGR is built without the SFCGAL library, this method will always return -1.0

#### Returns

area enclosed by the PolyhedralSurface

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

References `CPL::Error()`.

11.87.3.18 `getDimension()`

```
int OGRPolyhedralSurface::getDimension ( ) const [override], [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. ??).

11.87.3.19 `getEnvelope()` [1/2]

```
void OGRPolyhedralSurface::getEnvelope (
    OGREnvelope * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope for this geometry in the passed `psEnvelope` structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

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

### 11.87.3.20 getEnvelope() [2/2]

```
void OGRPolyhedralSurface::getEnvelope (
    OGREnvelope3D * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

## Since

OGR 1.9.0

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

### 11.87.3.21 getGeometryName()

```
const char * OGRPolyhedralSurface::getGeometryName ( ) const [override], [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 **OGRTriangulatedSurface** (p. ??).

### 11.87.3.22 getGeometryRef() [1/2]

```
OGRGeometry * OGRPolyhedralSurface::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 until the next change to the geometry container. Use IGeometry::clone() to make a copy.

**Parameters**

<i>i</i>	the index of the geometry to fetch, between 0 and <b>getNumGeometries()</b> (p. ??) - 1.
----------	--

**Returns**

pointer to requested geometry.

Referenced by `OGRGeometryFactory::forceTo()`, `OGRGeometryFactory::forceToPolygon()`, and `OGR↳  
TriangulatedSurface::operator=()`.

**11.87.3.23 getGeometryRef()** [2/2]

```
const  OGRGeometry * OGRPolyhedralSurface::getGeometryRef (
        int i ) const
```

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 until the next change to the geometry container. Use `IGeometry::clone()` to make a copy.

**Parameters**

<i>i</i>	the index of the geometry to fetch, between 0 and <b>getNumGeometries()</b> (p. ??) - 1.
----------	--

**Returns**

pointer to requested geometry.

**11.87.3.24 getNumGeometries()**

```
int OGRPolyhedralSurface::getNumGeometries ( ) const
```

Fetch number of geometries in PolyhedralSurface.

**Returns**

count of children geometries. May be zero.

Referenced by `OGRGeometry::dumpReadable()`, `OGRGeometryFactory::forceTo()`, and `OGRGeometryFactory↳  
::forceToPolygon()`.

## 11.87.3.25 hasCurveGeometry()

```
OGRBoolean OGRPolyhedralSurface::hasCurveGeometry (
    int bLookForNonLinear = FALSE ) const [override], [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

<i>bLookForNonLinear</i>	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. ??).

## 11.87.3.26 importFromWkb()

```
OGRERR OGRPolyhedralSurface::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	if wkbVariantPostGIS1, special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

## Since

GDAL 2.3

< Success

< Not enough memory

< Not enough data to deserialize

< Success

< Corrupt data

< Success

< Success

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

References CPLAssert, CPLDebug(), OGRGeometryFactory::createFromWkb(), getGeometryType(), OGRERR\_CORRUPT\_DATA, OGRERR\_NONE, OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_NOT\_ENOUGH\_MEMORY, VSI\_CALLOC\_VERBOSE, and wkbXDR.

### 11.87.3.27 importFromWkt() [1/3]

**OGRERR** OGRGeometry::importFromWkt

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).



## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

## 11.87.3.28 importFromWkt() [2/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

## 11.87.3.29 importFromWkt() [3/3]

```
OGRERR OGRPolyhedralSurface::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated < Success

< Success

< Success

< Corrupt data

< Success

< Success

< Success

< Success

< Corrupt data  
< Success

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

References [CPLError\(\)](#), [OGRGeometryFactory::createGeometry\(\)](#), [EQUAL](#), [OGRERR\\_NONE](#), and [OGRGeometry::toPolygon\(\)](#).

### 11.87.3.30 isEmpty()

```
OGRBoolean OGRPolyhedralSurface::IsEmpty ( ) const [override], [virtual]
```

Checks if the PolyhedralSurface is empty.

#### Returns

TRUE if the PolyhedralSurface is empty, FALSE otherwise

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

Referenced by [Equals\(\)](#).

### 11.87.3.31 PointOnSurface()

```
OGRErr OGRPolyhedralSurface::PointOnSurface (
    OGRPoint * poPoint ) const [override], [virtual]
```

This method relates to the [SFCOM ISurface::get\\_PointOnSurface\(\)](#) method.

NOTE: Only implemented when GEOS included in build.

## Parameters

<i>poPoint</i>	point to be set with an internal point.
----------------	---

## Returns

OGRERR\_NONE if it succeeds or OGRERR\_FAILURE otherwise.

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

## 11.87.3.32 removeGeometry()

```
OGRERR OGRPolyhedralSurface::removeGeometry (
    int iGeom,
    int bDelete = TRUE )
```

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.

## Parameters

<i>iGeom</i>	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
<i>bDelete</i>	if TRUE the geometry will be deallocated, otherwise it will not. The default is TRUE as the container is considered to own the geometries in it.

## Returns

OGRERR\_NONE if successful, or OGRERR\_FAILURE if the index is out of range.

## 11.87.3.33 setCoordinateDimension()

```
void OGRPolyhedralSurface::setCoordinateDimension (
    int nNewDimension ) [override], [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. This will also remove the M dimension if present before this call.

## Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

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

References **OGRGeometry::setCoordinateDimension()**.

### 11.87.3.34 transform()

```
OGRERR OGRPolyhedralSurface::transform (
    OGRCoordinateTransformation * poCT ) [override], [virtual]
```

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGR↔SpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

#### Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

#### Returns

**OGRERR\_NONE** on success or an error code.

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

### 11.87.3.35 WkbSize()

```
int OGRPolyhedralSurface::WkbSize ( ) const [override], [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.

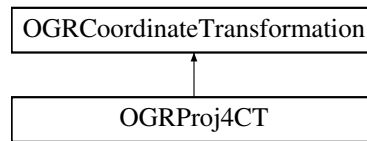
Implements **OGRGeometry** (p. ??).

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

- **ogr\_geometry.h**
- **ogrpolyhedralsurface.cpp**

## 11.88 OGRProj4CT Class Reference

Inheritance diagram for OGRProj4CT:



### Public Member Functions

- virtual **OGRSpatialReference** \* **GetSourceCS** () override
- virtual **OGRSpatialReference** \* **GetTargetCS** () override
- virtual int **Transform** (int nCount, double \*x, double \*y, double \*z=nullptr) override
- virtual int **TransformEx** (int nCount, double \*x, double \*y, double \*z=nullptr, int \*panSuccess=nullptr) override
- virtual bool **GetEmitErrors** () override
- virtual void **SetEmitErrors** (bool bEmitErrors) override

### Additional Inherited Members

#### 11.88.1 Member Function Documentation

##### 11.88.1.1 GetEmitErrors()

```
virtual bool OGRProj4CT::GetEmitErrors ( ) [inline], [override], [virtual]
```

Whether the transformer will emit CPLError

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

##### 11.88.1.2 GetSourceCS()

```
OGRSpatialReference * OGRProj4CT::GetSourceCS ( ) [override], [virtual]
```

Fetch internal source coordinate system.

Implements **OGRCoordinateTransformation** (p. ??).

### 11.88.1.3 GetTargetCS()

```
OGRSpatialReference * OGRProj4CT::GetTargetCS ( ) [override], [virtual]
```

Fetch internal target coordinate system.

Implements **OGRCoordinateTransformation** (p. ??).

### 11.88.1.4 SetEmitErrors()

```
virtual void OGRProj4CT::SetEmitErrors (
    bool ) [inline], [override], [virtual]
```

Set if the transformer must emit CPLError

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

### 11.88.1.5 Transform()

```
int OGRProj4CT::Transform (
    int nCount,
    double * x,
    double * y,
    double * z = nullptr ) [override], [virtual]
```

Transform points from source to destination space.

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

The method **TransformEx()** (p. ??) allows extended success information to be captured indicating which points failed to transform.

#### Parameters

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.

#### Returns

TRUE on success, or FALSE if some or all points fail to transform.

Implements **OGRCoordinateTransformation** (p. ??).

References CPLMalloc().

## 11.88.1.6 TransformEx()

```
int OGRProj4CT::TransformEx (
    int nCount,
    double * x,
    double * y,
    double * z = nullptr,
    int * pabSuccess = nullptr ) [override], [virtual]
```

Transform an array of points

## Parameters

<i>nCount</i>	Number of points
<i>x</i>	Array of nCount x values.
<i>y</i>	Array of nCount y values.
<i>z</i>	Array of nCount z values.
<i>pabSuccess</i>	Output array of nCount value that will be set to TRUE/FALSE

## Returns

TRUE or FALSE

Implements **OGRCoordinateTransformation** (p. ??).

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

- ogrct.cpp

## 11.89 OGRProj4Datum Struct Reference

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

- ogr\_srs\_proj4.cpp

## 11.90 OGRProj4PM Struct Reference

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

- ogr\_srs\_proj4.cpp

## 11.91 OGRRawPoint Class Reference

```
#include <ogr_geometry.h>
```

## Public Member Functions

- **OGRRawPoint** ()
- **OGRRawPoint** (double xIn, double yIn)

## Public Attributes

- double **x**
- double **y**

### 11.91.1 Detailed Description

Simple container for a position.

### 11.91.2 Constructor & Destructor Documentation

#### 11.91.2.1 OGRRawPoint() [1/2]

```
OGRRawPoint::OGRRawPoint ( ) [inline]
```

Constructor

#### 11.91.2.2 OGRRawPoint() [2/2]

```
OGRRawPoint::OGRRawPoint (
    double xIn,
    double yIn ) [inline]
```

Constructor

### 11.91.3 Member Data Documentation

#### 11.91.3.1 x

```
double OGRRawPoint::x
```

x

Referenced by OGRSimpleCurve::getX(), and OGRPoint::importFromWkt().



## 11.91.3.2 y

```
double OGRRawPoint::y
```

y

Referenced by OGRSimpleCurve::getY(), and OGRPoint::importFromWkt().

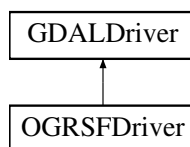
The documentation for this class was generated from the following file:

- **ogr\_geometry.h**

## 11.92 OGRSFDriver Class Reference

```
#include <ogr_ssf_frmts.h>
```

Inheritance diagram for OGRSFDriver:



### 11.92.1 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 file:

- **ogr\_ssf\_frmts.h**

## 11.93 OGRSFDriverRegistrar Class Reference

```
#include <ogr_ssf_frmts.h>
```

### 11.93.1 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

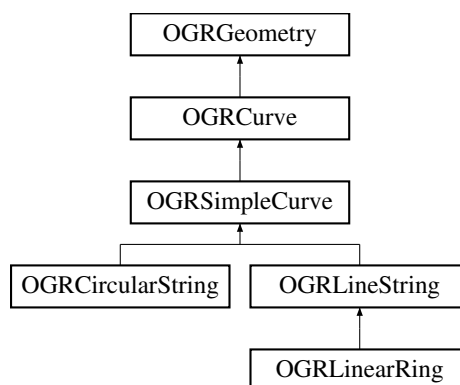
The documentation for this class was generated from the following files:

- **ogrsf\_frmts.h**
- **ogrsfdriverregistrar.cpp**

## 11.94 OGRSimpleCurve Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRSimpleCurve:



#### Public Types

- typedef **OGRPoint** **ChildType**

## Public Member Functions

- **OGRSimpleCurve & operator=** (const **OGRSimpleCurve** &other)  
*Assignment operator.*
- Iterator **begin** ()
- Iterator **end** ()
- ConstIterator **begin** () const
- ConstIterator **end** () const
- virtual int **WkbSize** () const override  
*Returns size of related binary representation.*
- virtual **OGRERR** **importFromWkb** (const unsigned char \*, int, **OGRWkbVariant**, int &nBytesConsumedOut) override  
*Assign geometry from well known binary data.*
- virtual **OGRERR** **exportToWkb** ( **OGRWkbByteOrder**, unsigned char \*, **OGRWkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known binary format.*
- **OGRERR** **importFromWkt** (const char \*\*) override
- virtual **OGRERR** **exportToWkt** (char \*\*ppszDstText, **OGRWkbVariant**= **wkbVariantOldOgc**) const override  
*Convert a geometry into well known text format.*
- virtual **OGRGeometry** \* **clone** () const override  
*Make a copy of this object.*
- virtual void **empty** () override  
*Clear geometry information. This restores the geometry to its initial state after construction, and before assignment of actual geometry.*
- virtual void **getEnvelope** (OGREnvelope \*psEnvelope) const override  
*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (OGREnvelope3D \*psEnvelope) const override  
*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- virtual **OGRBoolean** **IsEmpty** () const override  
*Returns TRUE (non-zero) if the object has no points.*
- virtual double **get\_Length** () const override  
*Returns the length of the curve.*
- virtual void **StartPoint** ( **OGRPoint** \*) const override  
*Return the curve start point.*
- virtual void **EndPoint** ( **OGRPoint** \*) const override  
*Return the curve end point.*
- virtual void **Value** (double, **OGRPoint** \*) const override  
*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 override  
*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.*

- double **getM** (int i) const  
*Get measure at vertex.*
- virtual **OGRBoolean Equals** (const **OGRGeometry** \*) const override  
*Returns TRUE if two geometries are equivalent.*
- virtual void **setCoordinateDimension** (int nDimension) override  
*Set the coordinate dimension.*
- virtual void **set3D** ( **OGRBoolean** bls3D) override  
*Add or remove the Z coordinate dimension.*
- virtual void **setMeasured** ( **OGRBoolean** blsMeasured) override  
*Add or remove the M 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)  
*Set the location of a vertex in line string.*
- void **setZ** (int, double)  
*Set the Z of a vertex in line string.*
- void **setM** (int, double)  
*Set the M of a vertex in line string.*
- void **setPoint** (int, double, double, double)  
*Set the location of a vertex in line string.*
- void **setPointM** (int, double, double, double)  
*Set the location of a vertex in line string.*
- void **setPoint** (int, double, double, double, double)  
*Set the location of a vertex in line string.*
- void **setPoints** (int, const **OGRRawPoint** \*, const double \*==nullptr)  
*Assign all points in a line string.*
- void **setPointsM** (int, const **OGRRawPoint** \*, const double \*)  
*Assign all points in a line string.*
- void **setPoints** (int, const double \*padfX, const double \*padfY, const double \*padfZIn=nullptr)  
*Assign all points in a line string.*
- void **setPointsM** (int, const double \*padfX, const double \*padfY, const double \*padfMIn=nullptr)  
*Assign all points in a line string.*
- void **setPoints** (int, const double \*padfX, const double \*padfY, const double \*padfZIn, const double \*padfMIn)  
*Assign all points in a line string.*
- void **addPoint** (const **OGRPoint** \*)  
*Add a point to a line string.*
- void **addPoint** (double, double)  
*Add a point to a line string.*
- void **addPoint** (double, double, double)  
*Add a point to a line string.*
- void **addPointM** (double, double, double)  
*Add a point to a line string.*
- void **addPoint** (double, double, double, double)  
*Add a point to a line string.*
- void **getPoints** ( **OGRRawPoint** \*, double \*==nullptr) const  
*Returns all points of line string.*

- void **getPoints** (void \*pabyX, int nXStride, void \*pabyY, int nYStride, void \*pabyZ=nullptr, int nZStride=0) const  
*Returns all points of line string.*
- void **getPoints** (void \*pabyX, int nXStride, void \*pabyY, int nYStride, void \*pabyZ, int nZStride, void \*pabyM, int nMStride) 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 override  
*Returns a point iterator over the curve.*
- virtual **OGRERR** **transform** ( **OGRCoordinateTransformation** \*poCT) override  
*Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** () override  
*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual void **segmentize** (double dfMaxLength) override  
*Modify the geometry such it has no segment longer then the given distance.*
- virtual void **swapXY** () override  
*Swap x and y coordinates.*
- virtual **OGRERR** **importFromWkt** (const char \*\*ppszInput)=0  
*Assign geometry from well known text data.*
- **OGRERR** **importFromWkt** (char \*\*ppszInput) CPL\_WARN\_DEPRECATED("Use importFromWkt( const char\*\*) instead")

### Protected Member Functions

- virtual double **get\_LinearArea** () const  
*Compute area of ring / closed linestring.*
- **OGRSimpleCurve** ()
- **OGRSimpleCurve** (const **OGRSimpleCurve** &other)  
*Copy constructor.*

### Additional Inherited Members

#### 11.94.1 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

#### 11.94.2 Member Typedef Documentation

### 11.94.2.1 ChildType

```
typedef OGRPoint OGRSimpleCurve::ChildType
```

Type of child elements.

## 11.94.3 Constructor & Destructor Documentation

### 11.94.3.1 OGRSimpleCurve() [1/2]

```
OGRSimpleCurve::OGRSimpleCurve ( ) [protected]
```

Constructor

### 11.94.3.2 OGRSimpleCurve() [2/2]

```
OGRSimpleCurve::OGRSimpleCurve (
    const OGRSimpleCurve & other ) [protected]
```

Copy constructor.

Note: before GDAL 2.1, only the default implementation of the constructor existed, which could be unsafe to use.

Since

GDAL 2.1

References `setPoints()`.

## 11.94.4 Member Function Documentation

### 11.94.4.1 addPoint() [1/4]

```
void OGRSimpleCurve::addPoint (
    const OGRPoint * poPoint )
```

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

## Parameters

<i>poPoint</i>	the point to assign to the new vertex.
----------------	--

References OGRPoint::getM(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), OGRGeometry::Is3D(), OGRGeometry::IsMeasured(), setPoint(), and setPointM().

Referenced by OGRLinearRing::closeRings(), OGRGeometryFactory::curveFromLineString(), OGRGeometryFactory::curveToLineString(), OGR\_G\_AddPoint(), OGR\_G\_AddPoint\_2D(), OGR\_G\_AddPointZM(), OGRTriangle::OGRTriangle(), and OGRLayer::SetSpatialFilterRect().

## 11.94.4.2 addPoint() [2/4]

```
void OGRSimpleCurve::addPoint (
    double x,
    double y )
```

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

## Parameters

<i>x</i>	the X coordinate to assign to the new point.
<i>y</i>	the Y coordinate to assign to the new point.

References setPoint().

## 11.94.4.3 addPoint() [3/4]

```
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

<i>x</i>	the X coordinate to assign to the new point.
<i>y</i>	the Y coordinate to assign to the new point.
<i>z</i>	the Z coordinate to assign to the new point (defaults to zero).

References `setPoint()`.

#### 11.94.4.4 `addPoint()` [ 4 / 4 ]

```
void OGRSimpleCurve::addPoint (
    double x,
    double y,
    double z,
    double m )
```

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

##### Parameters

<i>x</i>	the X coordinate to assign to the new point.
<i>y</i>	the Y coordinate to assign to the new point.
<i>z</i>	the Z coordinate to assign to the new point (defaults to zero).
<i>m</i>	the M coordinate to assign to the new point (defaults to zero).

References `setPoint()`.

#### 11.94.4.5 `addPointM()`

```
void OGRSimpleCurve::addPointM (
    double x,
    double y,
    double m )
```

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

##### Parameters

<i>x</i>	the X coordinate to assign to the new point.
<i>y</i>	the Y coordinate to assign to the new point.
<i>m</i>	the M coordinate to assign to the new point.

References `setPointM()`.

Referenced by `OGR_G_AddPointM()`.



## 11.94.4.6 addSubLineString()

```
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

<i>poOtherLine</i>	the other <b>OGRLineString</b> (p. ??).
<i>nStartVertex</i>	the first vertex to copy, defaults to 0 to start with the first vertex in the other linestring.
<i>nEndVertex</i>	the last vertex to copy, defaults to -1 indicating the last vertex of the other line string.

References CPLAssert, getNumPoints(), and setNumPoints().

Referenced by OGRCircularString::CurveToLine(), OGRGeometryFactory::forceToLineString(), and OGRGeometryFactory::forceToMultiLineString().

## 11.94.4.7 begin() [1/2]

```
OGRSimpleCurve::Iterator OGRSimpleCurve::begin ( )
```

Return begin of point iterator.

Using this iterator for standard range-based loops is safe, but due to implementation limitations, you shouldn't try to access (dereference) more than one iterator step at a time, since you will get a reference to the same **OGRPoint** (p. ??)& object.

Since

GDAL 2.3

## 11.94.4.8 begin() [2/2]

```
OGRSimpleCurve::ConstIterator OGRSimpleCurve::begin ( ) const
```

Return begin of point iterator.

Using this iterator for standard range-based loops is safe, but due to implementation limitations, you shouldn't try to access (dereference) more than one iterator step at a time, since you will get a reference to the same **OGRPoint** (p. ??)& object.

Since

GDAL 2.3

## 11.94.4.9 clone()

```
OGRGeometry * OGRSimpleCurve::clone ( ) const [override], [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()**, **OGRGeometry::getGeometryType()**, **getNumPoints()**, **OGRGeometry::getSpatialReference()**, **setPoints()**, and **OGRGeometry::toSimpleCurve()**.

Referenced by **OGRGeometryFactory::curveFromLineString()**, and **OGRLineString::CurveToLine()**.

## 11.94.4.10 empty()

```
void OGRSimpleCurve::empty ( ) [override], [virtual]
```

Clear geometry information. This restores the geometry to its 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()**.

## 11.94.4.11 end() [1/2]

```
OGRSimpleCurve::Iterator OGRSimpleCurve::end ( )
```

Return end of point iterator.

## 11.94.4.12 end() [2/2]

```
OGRSimpleCurve::ConstIterator OGRSimpleCurve::end ( ) const
```

Return end of point iterator.

## 11.94.4.13 EndPoint()

```
void OGRSimpleCurve::EndPoint (
    OGRPoint * poPoint ) const [override], [virtual]
```

Return the curve end point.

This method relates to the SF COM ICurve::get\_EndPoint() method.

## Parameters

<i>poPoint</i>	the point to be assigned the end location.
----------------	--

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

References `getPoint()`.

Referenced by `OGRGeometryFactory::forceToLineString()`, `Value()`, and `OGRCircularString::Value()`.

## 11.94.4.14 Equals()

```
OGRBoolean OGRSimpleCurve::Equals (
    const OGRGeometry * ) const [override], [virtual]
```

Returns TRUE if two geometries are equivalent.

This operation implements the SQL/MM ST\_OrderingEquals() operation.

The comparison is done in a structural way, that is to say that the geometry types must be identical, as well as the number and ordering of sub-geometries and vertices. Or equivalently, two geometries are considered equal by this method if their WKT/WKB representation is equal. Note: this must be distinguished for equality in a spatial way (which is the purpose of the ST\_Equals() operation).

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()`, `IsEmpty()`, and `OGRGeometry::toSimpleCurve()`.

## 11.94.4.15 exportToWkb()

```
OGRERR OGRSimpleCurve::exportToWkb (
    OGRwkbByteOrder eByteOrder,
    unsigned char * pabyData,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [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

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

## Returns

Currently `OGRERR_NONE` is always returned.

< Success

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

Reimplemented in **OGRCircularString** (p. ??), and **OGRLinearRing** (p. ??).

References `OGRGeometry::CoordinateDimension()`, `CPL_LSBPTR32`, `CPL_MSBPTR32`, `CPL_SWAP32`, `CPL_SWAP64PTR`, `OGRGeometry::getGeometryType()`, `OGRGeometry::getIsoGeometryType()`, `OGRGeometry::Is3D()`, `OGRGeometry::IsMeasured()`, `OGRERR_NONE`, `wkbFlatten`, `wkbNDR`, `wkbVariantIso`, and `wkbVariantPostGIS1`.

Referenced by `OGRCircularString::exportToWkb()`.

## 11.94.4.16 exportToWkt()

```
OGRERR OGRSimpleCurve::exportToWkt (
    char ** ppszDstText,
    OGRwkbVariant eWkbVariant = wkbVariantOldOgc ) const [override], [virtual]
```

Convert a geometry into well known text format.

This method relates to the `SFCOM IWks::ExportToWKT()` method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

## Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, <code>*ppszDstText</code> should be freed with <b>CPLFree()</b> (p. ??).
<i>eWkbVariant</i>	the specification that must be conformed too : <ul style="list-style-type: none"> <li><code>wkbVariantOgc</code> for old-style 99-402 extended dimension (Z) WKB types</li> <li><code>wkbVariantIso</code> for SFSQL 1.2 and ISO SQL/MM Part 3</li> </ul>

**Returns**

Currently OGRERR\_NONE is always returned.

< Success

< Not enough memory

< Not enough memory

< Success

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

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

References CPLDebug(), CPLStrdup(), OGRGeometry::getGeometryName(), OGRGeometry::Is3D(), IsEmpty(), OGRGeometry::IsMeasured(), OGRERR\_NONE, OGRERR\_NOT\_ENOUGH\_MEMORY, CPLString::Printf(), VS↵ I\_MALLOC\_VERBOSE, VSIFree(), and wkbVariantIso.

Referenced by OGRCircularString::exportToWkt(), and OGRPolygon::exportToWkt().

**11.94.4.17 flattenTo2D()**

```
void OGRSimpleCurve::flattenTo2D ( ) [override], [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. ??).

References setMeasured().

**11.94.4.18 get\_Length()**

```
double OGRSimpleCurve::get_Length ( ) const [override], [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().

**11.94.4.19 get\_LinearArea()**

```
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.

References WkbSize().

Referenced by OGRCircularString::get\_Area().

**11.94.4.20 getEnvelope()** [1/2]

```
void OGRSimpleCurve::getEnvelope (
    OGREnvelope * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

**Parameters**

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

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

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

References IsEmpty().

Referenced by getEnvelope(), OGRCircularString::getEnvelope(), OGRLinearRing::isPointInRing(), and OGRLinearRing::isPointOnRingBoundary().

**11.94.4.21 getEnvelope()** [2/2]

```
void OGRSimpleCurve::getEnvelope (
    OGREnvelope3D * psEnvelope ) const [override], [virtual]
```

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

## Since

OGR 1.9.0

Implements **OGRGeometry** (p. ??).Reimplemented in **OGRCircularString** (p. ??).References `getEnvelope()`, and `IsEmpty()`.11.94.4.22 `getM()`

```
double OGRSimpleCurve::getM (
    int iVertex ) const
```

Get measure at vertex.

Returns the M (measure) value at the indicated vertex. If no M value is available, 0.0 is returned. If `iVertex` is out of range a crash may occur, no internal range checking is performed.

## Parameters

<i>iVertex</i>	the vertex to return, between 0 and <b><code>getNumPoints()</code></b> (p. ??)-1.
----------------	---

## Returns

M value.

11.94.4.23 `getNumPoints()`

```
int OGRSimpleCurve::getNumPoints ( ) const [inline], [override], [virtual]
```

Fetch vertex count.

Returns the number of vertices in the line string.

## Returns

vertex count.

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

Referenced by `addSubLineString()`, `clone()`, `OGRGeometryFactory::curveFromLineString()`, `OGRGeometry::dumpReadable()`, `Equals()`, `OGRPolygon::exportToWkt()`, `OGRGeometryFactory::forceTo()`, `OGRGeometryFactory::forceToLineString()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRSimpleCurvePointIterator::getNextPoint()`, `OGRLinearRing::isPointInRing()`, `OGRLinearRing::isPointOnRingBoundary()`, `OGR_G_GetPoints()`, `OGR_G_GetPointsZM()`, `OGRLinearRing::OGRLinearRing()`, and `OGRLinearRing::transform()`.

#### 11.94.4.24 `getPoint()`

```
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>i</i>	the vertex to fetch, from 0 to <b>getNumPoints()</b> (p. ??)-1.
<i>poPoint</i>	a point to initialize with the fetched point.

References `CPLAssert`, `OGRPoint::setM()`, `OGRPoint::setX()`, `OGRPoint::setY()`, and `OGRPoint::setZ()`.

Referenced by `OGRGeometryFactory::approximateArcAngles()`, `OGRLinearRing::closeRings()`, `OGRGeometryFactory::curveFromLineString()`, `EndPoint()`, `OGRSimpleCurvePointIterator::getNextPoint()`, `OGRLinearRing::reverseWindingOrder()`, and `StartPoint()`.

#### 11.94.4.25 `getPointIterator()`

```
OGRPointIterator * OGRSimpleCurve::getPointIterator ( ) const [override], [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. ??).

#### 11.94.4.26 `getPoints()` [1/3]

```
void OGRSimpleCurve::getPoints (
    OGRRawPoint * paoPointsOut,
    double * padfZOut = nullptr ) const
```

Returns all points of line string.

This method copies all points into user list. This list must be at least `sizeof(OGRRawPoint) * OGRGeometry::getNumPoints()` byte in size. It also copies all Z coordinates.

There is no SFCOM analog to this method.



## Parameters

<i>paoPointsOut</i>	a buffer into which the points is written.
<i>padfZOut</i>	the Z values that go with the points (optional, may be NULL).

Referenced by `getPoints()`, `OGR_G_GetPoints()`, and `OGR_G_GetPointsZM()`.

11.94.4.27 `getPoints()` [2/3]

```
void OGRSimpleCurve::getPoints (
    void * pabyX,
    int nXStride,
    void * pabyY,
    int nYStride,
    void * pabyZ = nullptr,
    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 consecutive elements of the array.

On some CPU architectures, care must be taken so that the arrays are properly aligned.

There is no SFCOM analog to this method.

## Parameters

<i>pabyX</i>	a buffer of at least $(\text{sizeof}(\text{double}) * nXStride * nPointCount)$ bytes, may be NULL.
<i>nXStride</i>	the number of bytes between 2 elements of <i>pabyX</i> .
<i>pabyY</i>	a buffer of at least $(\text{sizeof}(\text{double}) * nYStride * nPointCount)$ bytes, may be NULL.
<i>nYStride</i>	the number of bytes between 2 elements of <i>pabyY</i> .
<i>pabyZ</i>	a buffer of at last size $(\text{sizeof}(\text{double}) * nZStride * nPointCount)$ bytes, may be NULL.
<i>nZStride</i>	the number of bytes between 2 elements of <i>pabyZ</i> .

Since

OGR 1.9.0

References `getPoints()`.

11.94.4.28 `getPoints()` [3/3]

```
void OGRSimpleCurve::getPoints (
    void * pabyX,
```

```

    int nXStride,
    void * pabyY,
    int nYStride,
    void * pabyZ,
    int nZStride,
    void * pabyM,
    int nMStride ) const

```

Returns all points of line string.

This method copies all points into user arrays. The user provides the stride between 2 consecutive elements of the array.

On some CPU architectures, care must be taken so that the arrays are properly aligned.

There is no SFCOM analog to this method.

#### Parameters

<i>pabyX</i>	a buffer of at least (sizeof(double) * <i>nXStride</i> * <i>nPointCount</i> ) bytes, may be NULL.
<i>nXStride</i>	the number of bytes between 2 elements of <i>pabyX</i> .
<i>pabyY</i>	a buffer of at least (sizeof(double) * <i>nYStride</i> * <i>nPointCount</i> ) bytes, may be NULL.
<i>nYStride</i>	the number of bytes between 2 elements of <i>pabyY</i> .
<i>pabyZ</i>	a buffer of at last size (sizeof(double) * <i>nZStride</i> * <i>nPointCount</i> ) bytes, may be NULL.
<i>nZStride</i>	the number of bytes between 2 elements of <i>pabyZ</i> .
<i>pabyM</i>	a buffer of at last size (sizeof(double) * <i>nMStride</i> * <i>nPointCount</i> ) bytes, may be NULL.
<i>nMStride</i>	the number of bytes between 2 elements of <i>pabyM</i> .

Since

OGR 2.1.0

#### 11.94.4.29 getSubLine()

```

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

<i>dfDistanceFrom</i>	The distance from the origin of linestring, where the subline should begins
<i>dfDistanceTo</i>	The distance from the origin of linestring, where the subline should ends
<i>bAsRatio</i>	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 OGRGeometry::assignSpatialReference(), CPL::Error(), get\_Length(), OGRGeometry::getCoordinateDimension(), OGRGeometry::getSpatialReference(), and setCoordinateDimension().

**11.94.4.30 getX()**

```
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**

<i>iVertex</i>	the vertex to return, between 0 and <b>getNumPoints()</b> (p. ??)-1.
----------------	--

**Returns**

X value.

References OGRRawPoint::x.

Referenced by OGRLinearRing::closeRings(), OGRGeometryFactory::curveFromLineString(), Equals(), OGRLinearRing::isPointInRing(), and OGRLinearRing::isPointOnRingBoundary().

**11.94.4.31 getY()**

```
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**

<i>iVertex</i>	the vertex to return, between 0 and <b>getNumPoints()</b> (p. ??)-1.
----------------	--

**Returns**

X value.

References `OGRRawPoint::y`.

Referenced by `OGRLinearRing::closeRings()`, `OGRGeometryFactory::curveFromLineString()`, `Equals()`, `OGRLinearRing::isPointInRing()`, and `OGRLinearRing::isPointOnRingBoundary()`.

**11.94.4.32 getZ()**

```
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**

<i>iVertex</i>	the vertex to return, between 0 and <code>getNumPoints()</code> (p. ??)-1.
----------------	--

**Returns**

Z value.

Referenced by `OGRLinearRing::closeRings()`, and `Equals()`.

**11.94.4.33 importFromWkb()**

```
OGRERR OGRSimpleCurve::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the `SFCOM IWks::ImportFromWKB()` method.

This method is the same as the C function `OGR_G_ImportFromWkb()` (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	if wkbVariantPostGIS1, special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

## Since

GDAL 2.3

< Success

< Corrupt data

< Not enough data to deserialize

< Failure

< Success

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

Reimplemented in **OGRCircularString** (p. ??), and **OGRLinearRing** (p. ??).

References OGRGeometry::CoordinateDimension(), CPLError(), OGRERR\_CORRUPT\_DATA, and OGRERR\_NONE.

Referenced by OGRCircularString::importFromWkb().

## 11.94.4.34 importFromWkt() [1/3]

```
OGRERR OGRGeometry::importFromWkt [inline]
```

Deprecated.

**Deprecated** in GDAL 2.3

## 11.94.4.35 importFromWkt() [2/3]

```
OGRERR OGRGeometry::importFromWkt
```

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

## 11.94.4.36 importFromWkt() [3/3]

```
OGRERR OGRSimpleCurve::importFromWkt (
    const char ** ppszInput ) [override], [virtual]
```

deprecated < Success

< Success

< Corrupt data

< Success

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

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

References OGRERR\_CORRUPT\_DATA, OGRERR\_NONE, set3D(), and setMeasured().

Referenced by OGRCircularString::importFromWkt().

## 11.94.4.37 isEmpty()

```
OGRBoolean OGRSimpleCurve::IsEmpty( ) const [override], [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().

## 11.94.4.38 operator=()

```
OGRSimpleCurve & OGRSimpleCurve::operator= (
    const OGRSimpleCurve & other )
```

Assignment operator.

Note: before GDAL 2.1, only the default implementation of the operator existed, which could be unsafe to use.

Since

GDAL 2.1

References OGRGeometry::operator=(), and setPoints().

Referenced by OGRLineString::operator=(), and OGRCircularString::operator=().

## 11.94.4.39 Project()

```
double OGRSimpleCurve::Project (
    const OGRPoint * ) const [virtual]
```

Project point on linestring.

The input point projected 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().

## 11.94.4.40 reversePoints()

```
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(), and OGRCircularString::segmentize().

#### 11.94.4.41 segmentize()

```
void OGRSimpleCurve::segmentize (
    double dfMaxLength ) [override], [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

<i>dfMaxLength</i>	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 consecutive vertices is lower than *dfMaxLength*.

## Parameters

<i>dfMaxLength</i>	maximum length between 2 consecutive vertices.
--------------------	--

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

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

References CPLError().

## 11.94.4.42 set3D()

```
void OGRSimpleCurve::set3D (
    OGRBoolean bIs3D ) [override], [virtual]
```

Add or remove the Z coordinate dimension.

This method adds or removes the explicit Z coordinate dimension. Removing the Z coordinate dimension of a geometry will remove any existing Z values. Adding the Z dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

## Parameters

<i>bIs3D</i>	Should the geometry have a Z dimension, either TRUE or FALSE.
--------------	---

Since

GDAL 2.1

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

Referenced by OGRPolygon::exportToWkt(), and importFromWkt().

## 11.94.4.43 setCoordinateDimension()

```
void OGRSimpleCurve::setCoordinateDimension (
    int nNewDimension ) [override], [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, a compound curve, a polygon, etc. will affect the children geometries. This will also remove the M dimension if present before this call.

**Deprecated** use **set3D()** (p. ??) or **setMeasured()** (p. ??).

#### Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

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

References **setMeasured()**.

Referenced by **getSubLine()**.

#### 11.94.4.44 **setM()**

```
void OGRSimpleCurve::setM (
    int iPoint,
    double mIn )
```

Set the M 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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>mIn</i>	input M coordinate to assign.

References **setNumPoints()**.

#### 11.94.4.45 **setMeasured()**

```
void OGRSimpleCurve::setMeasured (
    OGRBoolean bIsMeasured ) [override], [virtual]
```

Add or remove the M coordinate dimension.

This method adds or removes the explicit M coordinate dimension. Removing the M coordinate dimension of a geometry will remove any existing M values. Adding the M dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

#### Parameters

<i>bIsMeasured</i>	Should the geometry have a M dimension, either TRUE or FALSE.
--------------------	---

Since

GDAL 2.1

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

Referenced by OGRPolygon::exportToWkt(), flattenTo2D(), importFromWkt(), and setCoordinateDimension().

#### 11.94.4.46 setNumPoints()

```
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

<i>nNewPointCount</i>	the new number of points for geometry.
<i>bZeroizeNewContent</i>	whether to set to zero the new elements of arrays that are extended.

References CPLAssert, CPLFree, and VSI\_REALLOC\_VERBOSE.

Referenced by addSubLineString(), empty(), OGRCompoundCurve::get\_Area(), OGR\_G\_SetPointCount(), OGRLinearRing::OGRLinearRing(), setM(), setPoint(), setPointM(), setPoints(), setPointsM(), and setZ().

#### 11.94.4.47 setPoint() [1/4]

```
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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>poPoint</i>	the value to assign to the vertex.

References OGRPoint::getM(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), and setPointM().

Referenced by addPoint(), OGRGeometryFactory::approximateArcAngles(), OGRCompoundCurve::get\_Area(), and OGRLinearRing::reverseWindingOrder().

#### 11.94.4.48 setPoint() [2/4]

```
void OGRSimpleCurve::setPoint (
    int iPoint,
    double xIn,
    double yIn )
```

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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>xIn</i>	input X coordinate to assign.
<i>yIn</i>	input Y coordinate to assign.

References setNumPoints().

#### 11.94.4.49 setPoint() [3/4]

```
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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>xIn</i>	input X coordinate to assign.
<i>yIn</i>	input Y coordinate to assign.
<i>zIn</i>	input Z coordinate to assign (defaults to zero).

References `setNumPoints()`.

#### 11.94.4.50 `setPoint()` [4/4]

```
void OGRSimpleCurve::setPoint (
    int iPoint,
    double xIn,
    double yIn,
    double zIn,
    double mIn )
```

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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>xIn</i>	input X coordinate to assign.
<i>yIn</i>	input Y coordinate to assign.
<i>zIn</i>	input Z coordinate to assign (defaults to zero).
<i>mIn</i>	input M coordinate to assign (defaults to zero).

References `setNumPoints()`.

#### 11.94.4.51 `setPointM()`

```
void OGRSimpleCurve::setPointM (
    int iPoint,
    double xIn,
    double yIn,
    double mIn )
```

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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>xIn</i>	input X coordinate to assign.
<i>yIn</i>	input Y coordinate to assign.
<i>mIn</i>	input M coordinate to assign (defaults to zero).

References `setNumPoints()`.

Referenced by `addPoint()`, `addPointM()`, and `setPoint()`.

#### 11.94.4.52 `setPoints()` [1/4]

```
void OGRSimpleCurve::setPoints (
    int nPointsIn,
    const OGRRawPoint * paoPointsIn,
    const double * padfZIn = nullptr )
```

Assign all points in a line string.

This method clears any existing points assigned to this line string, and assigns a whole new set. It is the most efficient way of assigning the value of a line string.

There is no SFCOM analog to this method.

##### Parameters

<i>nPointsIn</i>	number of points being passed in <i>paoPointsIn</i>
<i>paoPointsIn</i>	list of points being assigned.
<i>padfZIn</i>	the Z values that go with the points (optional, may be NULL).

References `OGRGeometry::getCoordinateDimension()`, and `setNumPoints()`.

Referenced by `clone()`, `OGRLinearRing::clone()`, `OGRSimpleCurve()`, and `operator=()`.

#### 11.94.4.53 `setPoints()` [2/4]

```
void OGRSimpleCurve::setPoints (
    int nPointsIn,
    const OGRRawPoint * paoPointsIn,
    const double * padfZIn,
    const double * padfMIn )
```

Assign all points in a line string.

This method clears any existing points assigned to this line string, and assigns a whole new set. It is the most efficient way of assigning the value of a line string.

There is no SFCOM analog to this method.

##### Parameters

<i>nPointsIn</i>	number of points being passed in <i>paoPointsIn</i>
<i>paoPointsIn</i>	list of points being assigned.
<i>padfZIn</i>	the Z values that go with the points.
<i>padfMIn</i>	the M values that go with the points.

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

#### 11.94.4.54 setPoints() [ 3 / 4 ]

```
void OGRSimpleCurve::setPoints (
    int nPointsIn,
    const double * padfX,
    const double * padfY,
    const double * padfZIn = nullptr )
```

Assign all points in a line string.

This method clear any existing points assigned to this line string, and assigns a whole new set.

There is no SFCOM analog to this method.

##### Parameters

<i>nPointsIn</i>	number of points being passed in <i>padfX</i> and <i>padfY</i> .
<i>padfX</i>	list of X coordinates of points being assigned.
<i>padfY</i>	list of Y coordinates of points being assigned.
<i>padfZIn</i>	list of Z coordinates of points being assigned (defaults to NULL for 2D objects).

References setNumPoints().

#### 11.94.4.55 setPoints() [ 4 / 4 ]

```
void OGRSimpleCurve::setPoints (
    int nPointsIn,
    const double * padfX,
    const double * padfY,
    const double * padfZIn,
    const double * padfMIn )
```

Assign all points in a line string.

This method clear any existing points assigned to this line string, and assigns a whole new set.

There is no SFCOM analog to this method.

##### Parameters

<i>nPointsIn</i>	number of points being passed in <i>padfX</i> and <i>padfY</i> .
<i>padfX</i>	list of X coordinates of points being assigned.
<i>padfY</i>	list of Y coordinates of points being assigned.
<i>padfZIn</i>	list of Z coordinates of points being assigned.
<i>padfMIn</i>	list of M coordinates of points being assigned.

References `setNumPoints()`.

#### 11.94.4.56 `setPointsM()` [1/2]

```
void OGRSimpleCurve::setPointsM (
    int nPointsIn,
    const OGRRawPoint * paoPointsIn,
    const double * padfMIn )
```

Assign all points in a line string.

This method clears any existing points assigned to this line string, and assigns a whole new set. It is the most efficient way of assigning the value of a line string.

There is no SFCOM analog to this method.

##### Parameters

<i>nPointsIn</i>	number of points being passed in <i>paoPointsIn</i>
<i>paoPointsIn</i>	list of points being assigned.
<i>padfMIn</i>	the M values that go with the points.

References `setNumPoints()`.

#### 11.94.4.57 `setPointsM()` [2/2]

```
void OGRSimpleCurve::setPointsM (
    int nPointsIn,
    const double * padfX,
    const double * padfY,
    const double * padfMIn = nullptr )
```

Assign all points in a line string.

This method clear any existing points assigned to this line string, and assigns a whole new set.

There is no SFCOM analog to this method.

##### Parameters

<i>nPointsIn</i>	number of points being passed in <i>padfX</i> and <i>padfY</i> .
<i>padfX</i>	list of X coordinates of points being assigned.
<i>padfY</i>	list of Y coordinates of points being assigned.
<i>padfMIn</i>	list of M coordinates of points being assigned.



References `setNumPoints()`.

#### 11.94.4.58 `setZ()`

```
void OGRSimpleCurve::setZ (
    int iPoint,
    double zIn )
```

Set the Z 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

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>zIn</i>	input Z coordinate to assign.

References `OGRGeometry::getCoordinateDimension()`, and `setNumPoints()`.

#### 11.94.4.59 `StartPoint()`

```
void OGRSimpleCurve::StartPoint (
    OGRPoint * poPoint ) const [override], [virtual]
```

Return the curve start point.

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

##### Parameters

<i>poPoint</i>	the point to be assigned the start location.
----------------	--

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

References `getPoint()`.

Referenced by `OGRGeometryFactory::forceToLineString()`, `Value()`, and `OGRCircularString::Value()`.

#### 11.94.4.60 `swapXY()`

```
void OGRSimpleCurve::swapXY ( ) [override], [virtual]
```

Swap x and y coordinates.

Since

OGR 1.8.0

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

#### 11.94.4.61 transform()

```
OGRERR OGRSimpleCurve::transform (
    OGRCoordinateTransformation * poCT ) [override], [virtual]
```

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGR↵SpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

#### Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

#### Returns

OGRERR\_NONE on success or an error code.

< Not enough memory

< Failure

< Failure

< Failure

< Success

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

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

References CPLError(), CPLGetConfigOption(), OGRERR\_NOT\_ENOUGH\_MEMORY, OGRCoordinate↵Transformation::TransformEx(), VSI\_CALLOC\_VERBOSE, VSI\_MALLOC\_VERBOSE, and VSIFree().

## 11.94.4.62 Value()

```
void OGRSimpleCurve::Value (
    double dfDistance,
    OGRPoint * poPoint ) const [override], [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

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

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

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

References **EndPoint()**, **OGRGeometry::getCoordinateDimension()**, **OGRPoint::setX()**, **OGRPoint::setY()**, **OGRPoint::setZ()**, and **StartPoint()**.

## 11.94.4.63 WkbSize()

```
int OGRSimpleCurve::WkbSize ( ) const [override], [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::CoordinateDimension()**.

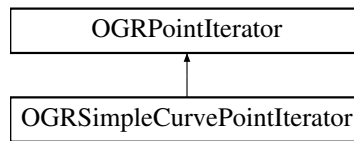
Referenced by **get\_LinearArea()**.

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

- **ogr\_geometry.h**
- **ogrcurve.cpp**
- **ogrlinestring.cpp**

## 11.95 OGRSimpleCurvePointIterator Class Reference

Inheritance diagram for OGRSimpleCurvePointIterator:



### Public Member Functions

- virtual **OGRBoolean** **getNextPoint** ( **OGRPoint** \*p) override  
*Returns the next point followed by the iterator.*

### Additional Inherited Members

#### 11.95.1 Member Function Documentation

##### 11.95.1.1 getNextPoint()

```
OGRBoolean OGRSimpleCurvePointIterator::getNextPoint (
    OGRPoint * p ) [override], [virtual]
```

Returns the next point followed by the iterator.

#### Parameters

<i>p</i>	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

## 11.96 OGRSpatialReference Class Reference

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

### Public Member Functions

- **OGRSpatialReference** (const **OGRSpatialReference** &)
- **OGRSpatialReference** (const char \*=*nullptr*)  
*Constructor.*
- virtual **~OGRSpatialReference** ()  
*OGRSpatialReference* (p. ??) *destructor.*
- **OGRSpatialReference** & **operator=** (const **OGRSpatialReference** &)
- 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.*
- void **dumpReadable** ()
- **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 \*=*nullptr*) const  
*Export coordinate system in XML format.*
- **OGRERR** **exportToPanorama** (long \*, long \*, long \*, long \*, double \*) const
- **OGRERR** **exportToERM** (char \*pszProj, char \*pszDatum, char \*pszUnits)
- **OGRERR** **exportToMlCoordSys** (char \*\*) const  
*Export coordinate system in Mapinfo style CoordSys format.*
- **OGRERR** **importFromWkt** (char \*\*)  
*Import from WKT string.*
- **OGRERR** **importFromWkt** (const char \*\*)  
*Import from WKT string.*
- **OGRERR** **importFromWkt** (const 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 \*==nullptr, double \*==nullptr)  
*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 importFromMlCoordSys** (const char \*)  
*Import Mapinfo style CoordSys definition.*
- **OGRERR morphToESRI** ()  
*Convert in place to ESRI WKT format.*
- **OGRERR morphFromESRI** ()  
*Convert in place from ESRI WKT format.*
- **OGRSpatialReference \* convertToOtherProjection** (const char \*pszTargetProjection, const char \*const  
\*papszOptions=nullptr) const  
*Convert to another equivalent projection.*
- **OGRERR Validate** () const  
*Validate SRS tokens.*
- **OGRERR StripCTParms** ( **OGR\_SRSNode** \*==nullptr)  
*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** () const  
*This method returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.*
- int **EPSGTreatsAsNorthingEasting** () const  
*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.*

  - **OGR\_SRSNode \* GetRoot ()**

*Return root node.*
- const **OGR\_SRSNode \* GetRoot ()** const

*Return root node.*
- void **SetRoot ( OGR\_SRSNode \*)**

*Set the root SRS node.*
- **OGR\_SRSNode \* GetAttrNode (const char \*)**

*Find named node in tree.*
- const **OGR\_SRSNode \* GetAttrNode (const char \*)** const

*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 SetNode (const char \*, double)**

*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 \*\*)** const CPL\_WARN\_DEPRECATED("Use GetLinearUnits( const char\*\*) instead")

*Fetch linear projection units.*
- double **GetLinearUnits (const char \*\*=nullptr)** const

*Fetch linear projection units.*
- double **GetTargetLinearUnits (const char \*pszTargetKey, char \*\*ppszRetName)** const CPL\_WARN\_DEPRECATED("Use GetTargetLinearUnits( const char\* Name=nullptr) const")

*Fetch linear units for target.*
- double const char \*\*double **GetTargetLinearUnits (const char \*pszTargetKey, const char \*\*ppszRetName=nullptr)** 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 \*\*)** const CPL\_WARN\_DEPRECATED("Use GetAngularUnits( const char\*\*) instead")

*Fetch angular geographic coordinate system units.*
- double **GetAngularUnits (const char \*\*=nullptr)** const

*Fetch angular geographic coordinate system units.*
- double **GetPrimeMeridian (char \*\*)** const CPL\_WARN\_DEPRECATED("Use GetPrimeMeridian( const char\*\*) instead")

*Fetch prime meridian info.*
- double **GetPrimeMeridian (const char \*\*=nullptr)** 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 **IsSameGeogCS** (const **OGRSpatialReference** \*, const char \*const \*papszOptions) 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 ?*
- int **IsSame** (const **OGRSpatialReference** \*, const char \*const \*papszOptions) const

*Do these two spatial references describe the same system ?*
- void **Clear** ()

*Wipe current definition.*
- **OGRERR SetLocalCS** (const char \*)

*Set the user visible LOCAL\_CS name.*
- **OGRERR SetProjCS** (const char \*)

*Set the user visible PROJCS name.*
- **OGRERR SetProjection** (const char \*)

*Set a projection name.*
- **OGRERR SetGeocCS** (const char \*pszGeocName)

*Set the user visible GEOCCS name.*
- **OGRERR SetGeogCS** (const char \*pszGeogName, const char \*pszDatumName, const char \*pszEllipsoidName, double dfSemiMajor, double dfInvFlattening, const char \*pszPMName=NULLPTR, double dfPMOffset=0.0, const char \*pszUnits=NULLPTR, double dfConvertToRadians=0.0)

*Set geographic coordinate system.*
- **OGRERR SetWellKnownGeogCS** (const char \*)

*Set a GeogCS based on well known name.*
- **OGRERR CopyGeogCSFrom** (const **OGRSpatialReference** \*poSrcSRS)

*Copy GEOGCS from another **OGRSpatialReference** (p. ??).*
- **OGRERR SetVertCS** (const char \*pszVertCSName, const char \*pszVertDatumName, int nVertDatumClass=2005)

*Set the user visible VERT\_CS name.*
- **OGRERR SetCompoundCS** (const char \*pszName, const **OGRSpatialReference** \*poHorizSRS, const **OGRSpatialReference** \*poVertSRS)

*Setup a compound coordinate system.*
- **OGRERR SetFromUserInput** (const char \*)

*Set spatial reference from various text formats.*
- **OGRERR SetTOWGS84** (double, double, double, double=0.0, double=0.0, double=0.0, double=0.0)

*Set the Bursa-Wolf conversion to WGS84.*
- **OGRERR GetTOWGS84** (double \*padfCoef, int nCoeff=7) const

*Fetch TOWGS84 parameters, if available.*
- double **GetSemiMajor** (**OGRERR** \*p=NULLPTR) const

*Get spheroid semi major axis.*
- double **GetSemiMinor** (**OGRERR** \*p=NULLPTR) const

*Get spheroid semi minor axis.*



- double **GetInvFlattening** ( **OGRerr** \*=**nullptr**) const  
*Get spheroid inverse flattening.*
- double **GetEccentricity** () const  
*Get spheroid eccentricity.*
- double **GetSquaredEccentricity** () const  
*Get spheroid squared eccentricity.*
- **OGRerr** **SetAuthority** (const char \*pszTargetKey, const char \*pszAuthority, int nCode)  
*Set the authority for a node.*
- **OGRerr** **AutoidentifyEPSG** ()  
*Set EPSG authority info if possible.*
- **OGRSpatialReferenceH** \* **FindMatches** (char \*\*papszOptions, int \*pnEntries, int \*\*ppanMatchConfidence) const  
*Try to identify a match between the passed SRS and a related SRS in a catalog (currently EPSG only)*
- int **GetEPSGGeogCS** () const
- 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=**nullptr**) 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=**nullptr**) 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** \*=**nullptr**) 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** \*=**nullptr**) 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** **SetEckertIV** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- **OGRerr** **SetEckertVI** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- **OGRerr** **SetEquirectangular** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRerr** **SetEquirectangular2** (double dfCenterLat, double dfCenterLong, double dfPseudoStdParallel1, double dfFalseEasting, double dfFalseNorthing)
- **OGRerr** **SetGEOS** (double dfCentralMeridian, double dfSatelliteHeight, double dfFalseEasting, double dfFalseNorthing)

- **OGRErr SetGH** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetIGH** ()
- **OGRErr SetGS** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetGaussSchreiberTMercator** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetGnomonic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetHOM** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)  
*Set a Hotine Oblique Mercator projection using azimuth angle.*
- **OGRErr SetHOM2PNO** (double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing)  
*Set a Hotine Oblique Mercator projection using two points on projection centerline.*
- **OGRErr SetHOMAC** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)  
*Set an Hotine Oblique Mercator Azimuth Center projection using azimuth angle.*
- **OGRErr SetIWMPolyconic** (double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetKrovak** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetLAEA** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetLCC** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetLCC1SP** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetLCCB** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetMC** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetMercator** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetMercator2SP** (double dfStdP1, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetMollweide** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetNZMG** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetOS** (double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetOrthographic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetPolyconic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetPS** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetRobinson** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetSinusoidal** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetStereographic** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetSOC** (double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetTM** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetTMVariant** (const char \*pszVariantName, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetTMG** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)

- **OGRErr SetTMSO** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetTPED** (double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetVDG** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetUTM** (int nZone, int bNorth=TRUE)  
*Set UTM projection definition.*
- int **GetUTMZone** (int \*pbNorth=nullptr) const  
*Get utm zone information.*
- **OGRErr SetWagner** (int nVariation, double dfCenterLat, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr SetQSC** (double dfCenterLat, double dfCenterLong)
- **OGRErr SetSCH** (double dfPegLat, double dfPegLong, double dfPegHeading, double dfPegHgt)
- **OGRErr SetStatePlane** (int nZone, int bNAD83=TRUE, const char \*pszOverrideUnitName=nullptr, double dfOverrideUnit=0.0)  
*Set State Plane projection definition.*
- **OGRErr ImportFromESRISStatePlaneWKT** (int nCode, const char \*pszDatumName, const char \*pszUnitsName, int nPCSCode, const char \*pszCSName=nullptr)
- **OGRErr ImportFromESRIWisconsinWKT** (const char \*pszPrjName, double dfCentralMeridian, double dfLatOfOrigin, const char \*pszUnitsName, const char \*pszCSName=nullptr)

### Static Public Member Functions

- static void **DestroySpatialReference** ( OGRSpatialReference \*poSRS)  
*OGRSpatialReference (p. ??) destructor.*
- static int **IsAngularParameter** (const char \*)
- static int **IsLongitudeParameter** (const char \*)
- static int **IsLinearParameter** (const char \*)
- static OGRSpatialReference \* **GetWGS84SRS** ()  
*Returns an instance of a SRS object with WGS84 WKT.*
- static OGRSpatialReferenceH **ToHandle** ( OGRSpatialReference \*poSRS)
- static OGRSpatialReference \* **FromHandle** ( OGRSpatialReferenceH hSRS)

### 11.96.1 Detailed Description

This class represents an 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.

Consult also the `OGC WKT Coordinate System Issues` page for implementation details of WKT in OGR.

### 11.96.2 Constructor & Destructor Documentation

#### 11.96.2.1 OGRSpatialReference() [1/2]

```
OGRSpatialReference::OGRSpatialReference (
    const OGRSpatialReference & oOther )
```

Simple copy constructor. See also **Clone()** (p. ??).

## Parameters

<i>oOther</i>	other spatial reference
---------------	-------------------------

References `OGR_SRSNode::Clone()`.

Referenced by `Clone()`, `CloneGeogCS()`, `convertToOtherProjection()`, and `GetWGS84SRS()`.

### 11.96.2.2 OGRSpatialReference() [2/2]

```
OGRSpatialReference::OGRSpatialReference (
    const char * pszWKT = nullptr ) [explicit]
```

Constructor.

This constructor takes an optional string argument which if passed should be a WKT representation of an SRS. Passing this is equivalent to not passing it, and then calling `importFromWkt()` (p. ??) with the WKT string.

Note that newly created objects are given a reference count of one.

The C function `OSRNewSpatialReference()` (p. ??) does the same thing as this constructor.

## Parameters

<i>pszWKT</i>	well known text definition to which the object should be initialized, or NULL (the default).
---------------	--

References `importFromWkt()`.

### 11.96.2.3 ~OGRSpatialReference()

```
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

## 11.96.3 Member Function Documentation

## 11.96.3.1 AutoidentifyEPSG()

```
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 (i.e. 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. ??).

Since GDAL 2.3, the **FindMatches()** (p. ??) method can also be used for improved matching by researching the EPSG catalog.

## Returns

OGRERR\_NONE or OGRERR\_UNSUPPORTED\_SRS.

< Success

< Success

< Unsupported SRS

References EQUAL, GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), GetEPSGGeogCS(), GetLinear↵ Units(), GetNormProjParm(), GetProjParm(), GetUTMZone(), IsGeographic(), IsProjected(), OGRERR\_NONE, OGRERR\_UNSUPPORTED\_SRS, SetAuthority(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_↵ POLAR\_STEREOGRAPHIC.

## 11.96.3.2 Clear()

```
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(), importFromERM(), importFromOzi(), importFromPanorama(), importFrom↵ PCI(), importFromProj4(), importFromWkt(), importFromXML(), morphToESRI(), operator=(), SetFromUserInput(), SetGeogCS(), and SetVertCS().

### 11.96.3.3 Clone()

```
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(), and OGRSpatialReference().

Referenced by convertToOtherProjection(), and exportToPrettyWkt().

### 11.96.3.4 CloneGeogCS()

```
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(), OGRSpatialReference(), SetAngularUnits(), SetRoot(), and SRS\_UA\_DEGREE\_CONV.

Referenced by morphFromESRI().

### 11.96.3.5 convertToOtherProjection()

```
OGRSpatialReference * OGRSpatialReference::convertToOtherProjection (
    const char * pszTargetProjection,
    const char *const * papszOptions = nullptr ) const
```

Convert to another equivalent projection.

Currently implemented:

- SRS\_PT\_MERCATOR\_1SP to SRS\_PT\_MERCATOR\_2SP
- SRS\_PT\_MERCATOR\_2SP to SRS\_PT\_MERCATOR\_1SP
- SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP to SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP
- SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP to SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP

## Parameters

<i>pszTargetProjection</i>	target projection.
<i>papszOptions</i>	lists of options. None supported currently.

## Returns

a new SRS, or NULL in case of error.

## Since

GDAL 2.3

References Clone(), CopyGeogCSFrom(), EQUAL, GetAttrValue(), GetEccentricity(), GetNormProjParm(), GetSemiMajor(), GetSquaredEccentricity(), M\_PI, OGRSpatialReference(), SetLCC(), SetMercator(), SetMercator2SP(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP, SRS\_PT\_MERCATOR\_1SP, and SRS\_PT\_MERCATOR\_2SP.

Referenced by IsSame(), and morphToESRI().

## 11.96.3.6 CopyGeogCSFrom()

```
OGRERR OGRSpatialReference::CopyGeogCSFrom (
    const OGRSpatialReference * poSrcSRS )
```

Copy GEOGCS from another **OGRSpatialReference** (p. ??).

The GEOGCS information is copied into this **OGRSpatialReference** (p. ??) from another. If this object has a PROJCS root already, the GEOGCS is installed within it, otherwise it is installed as the root.

## Parameters

<i>poSrcSRS</i>	the spatial reference to copy the GEOGCS information from.
-----------------	--

## Returns

OGRERR\_NONE on success or an error code.

< Failure

< Success

< Failure

< Failure  
< Success

References Clear(), OGR\_SRSNode::Clone(), OGR\_SRSNode::DestroyChild(), EQUAL, OGR\_SRSNode::FindChild(), GetAttrNode(), GetRoot(), OGR\_SRSNode::InsertChild(), IsGeocentric(), OGRERR\_FAILURE, OGRERR\_NONE, and SetRoot().

Referenced by convertToOtherProjection(), importFromERM(), importFromPanorama(), importFromPCI(), SetGeogCS(), and SetWellKnownGeogCS().

### 11.96.3.7 Dereference()

```
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().

### 11.96.3.8 DestroySpatialReference()

```
void OGRSpatialReference::DestroySpatialReference (
    OGRSpatialReference * poSRS ) [static]
```

**OGRSpatialReference** (p. ??) destructor.

This static method will destroy a **OGRSpatialReference** (p. ??). It is equivalent to calling delete on the object, but it ensures that the deallocation is properly executed within the OGR libraries heap on platforms where this can matter (win32).

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

#### Parameters

<i>poSRS</i>	the object to delete
--------------	----------------------

#### Since

GDAL 1.7.0



## 11.96.3.9 dumpReadable()

```
void OGRSpatialReference::dumpReadable ( )
```

Dump pretty wkt to stdout, mostly for debugging.

References CPLFree, and exportToPrettyWkt().

## 11.96.3.10 EPSGTreatsAsLatLong()

```
int OGRSpatialReference::EPSGTreatsAsLatLong ( ) const
```

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 EQUAL, GetAttrNode(), GetAuthorityName(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::Get↵ChildCount(), OGR\_SRSNode::GetValue(), and IsGeographic().

## 11.96.3.11 EPSGTreatsAsNorthingEasting()

```
int OGRSpatialReference::EPSGTreatsAsNorthingEasting ( ) const
```

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 EQUAL, GetAttrNode(), GetAuthorityName(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::Get↵ChildCount(), OGR\_SRSNode::GetValue(), and IsProjected().

Referenced by importFromEPSG().

## 11.96.3.12 exportToERM()

```

OGRERR OGRSpatialReference::exportToERM (
    char * pszProj,
    char * pszDatum,
    char * pszUnits )

```

Convert coordinate system to ERMapper format.

## Parameters

<i>pszProj</i>	32 character buffer to receive projection name.
<i>pszDatum</i>	32 character buffer to receive datum name.
<i>pszUnits</i>	32 character buffer to receive units name.

## Returns

OGRERR\_NONE on success, OGRERR\_SRS\_UNSUPPORTED if not translation is found, or OGRERR\_FAILURE on other failures.

< Unsupported SRS

< Success

< Unsupported SRS

< Success

< Success

< Unsupported SRS

< Success

References EQUAL, GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), GetEPSGGeogCS(), GetLinearUnits(), GetUTMZone(), importFromDict(), IsGeographic(), IsProjected(), OGRERR\_NONE, and OGRERR\_UNSUPPORTED\_SRS.

## 11.96.3.13 exportToMICoordSys()

```

OGRERR OGRSpatialReference::exportToMICoordSys (
    char ** ppszResult ) const

```

Export coordinate system in Mapinfo style CoordSys format.

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

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

## Parameters

<i>ppsResult</i>	pointer to which dynamically allocated Mapinfo CoordSys definition will be assigned.
------------------	--

## Returns

OGRERR\_NONE on success, OGRERR\_FAILURE on failure, OGRERR\_UNSUPPORTED\_OPERATION if MITAB library was not linked in.

< Unsupported operation

References CPLError(), OGRERR\_FAILURE, and OGRERR\_NONE.

## 11.96.3.14 exportToPanorama()

```
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()** (p. ??).

## Parameters

<i>piProjSys</i>	Pointer to variable, where the projection system code will be returned.
<i>piDatum</i>	Pointer to variable, where the coordinate system code will be returned.
<i>piEllips</i>	Pointer to variable, where the spheroid code will be returned.
<i>piZone</i>	Pointer to variable, where the zone for UTM projection system will be returned.
<i>padfPrjParams</i>	an existing 7 double buffer into which the projection parameters will be placed. See <b>importFromPanorama()</b> (p. ??) for the list of parameters.

## Returns

OGRERR\_NONE on success or an error code on failure.

< Success

< Success

References CPLAssert, CPLDebug(), EQUAL, GetAttrValue(), GetInvFlattening(), GetNormProjParm(), GetSemiMajor(), GetUTMZone(), IsLocal(), OGRERR\_NONE, SRS\_DN\_WGS84, SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_1ST\_POINT, SRS\_PP\_LATITUDE\_OF\_2ND\_POINT, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_LONGITUDE\_OF\_CENTER, SRS\_PP\_SCALE\_FACTOR, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, SRS\_PT\_AZIMUTHAL\_EQUIDISTANT, SRS\_PT\_CYLINDRICAL\_EQUAL\_AREA, SRS\_PT\_EQUIDISTANT\_CONIC, SRS\_PT\_EQUIRECTANGULAR, SRS\_PT\_GNOMONIC, SRS\_PT\_IMW\_POLYCONIC, SRS\_PT\_LAMBERT\_AZIMUTHAL\_EQUAL\_AREA, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP, SRS\_PT\_MERCATOR\_1SP, SRS\_PT\_MOLLWEIDE, SRS\_PT\_POLAR\_STEREOGRAPHIC, SRS\_PT\_POLYCONIC, SRS\_PT\_STEREOGRAPHIC, SRS\_PT\_TRANSVERSE\_MERCATOR, and SRS\_PT\_WAGNER\_I.

## 11.96.3.15 exportToPCI()

```

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()** (p. ??) when no longer needed.

LOCAL\_CS coordinate systems are not translatable. An empty string will be returned along with OGRERR\_NONE.

This method is the equivalent of the C function **OSRExportToPCI()** (p. ??).

## Parameters

<i>ppszProj</i>	pointer to which dynamically allocated PCI projection definition will be assigned.
<i>ppszUnits</i>	pointer to which dynamically allocated units definition will be assigned.
<i>ppadfPrjParams</i>	pointer to which dynamically allocated array of 17 projection parameters will be assigned. See <b>importFromPCI()</b> (p. ??) for the list of parameters.

## Returns

OGRERR\_NONE on success or an error code on failure.

< Success

< Success

< Success

References CPLAtof(), CPLDebug(), CPLMalloc(), CPLPrintInt32(), CPLPrintStringFill(), CPLStrdup(), CSLCount(), CSLDestroy(), EQUAL, GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetInvFlattening(), GetLinearUnits(), GetNormProjParm(), GetSemiMajor(), GetTOWGS84(), GetUTMZone(), OGR\_SRSNode::GetValue(), IsLocal(), OGRERR\_NONE, OGR\_SRSNode::SRSCalcSemiMinorFromInvFlattening(), SRS\_DN\_NAD27, SRS\_DN\_NAD83, SRS\_DN\_WGS84, SRS\_PP\_AZIMUTH, SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_LATITUDE\_OF\_POINT\_1, SRS\_PP\_LATITUDE\_OF\_POINT\_2, SRS\_PP\_LONGITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_POINT\_1, SRS\_PP\_LONGITUDE\_OF\_POINT\_2, SRS\_PP\_SCALE\_FACTOR, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, SRS\_PT\_ALBERS\_CONIC\_EQUAL\_AREA, SRS\_PT\_AZIMUTHAL\_EQUIDISTANT, SRS\_PT\_CASSINI\_SOLDNER, SRS\_PT\_EQUIDISTANT\_CONIC, SRS\_PT\_EQUIRECTANGULAR, SRS\_PT\_GNOMONIC, SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR, SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR\_TWO\_POINT\_NATURAL\_ORIGIN, SRS\_PT\_LAMBERT\_AZIMUTHAL\_EQUAL\_AREA, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP, SRS\_PT\_MERCATOR\_1SP, SRS\_PT\_MILLER\_CYLINDRICAL, SRS\_PT\_OBLIQUE\_STEREOGRAPHIC, SRS\_PT\_ORTHOGRAPHIC, SRS\_PT\_POLAR\_STEREOGRAPHIC, SRS\_PT\_POLYCONIC, SRS\_PT\_ROBINSON, SRS\_PT\_SINUSOIDAL, SRS\_PT\_STEREOGRAPHIC, SRS\_PT\_TRANSVERSE\_MERCATOR, SRS\_PT\_VANDERGRINTEN, STARTS\_WITH\_CI, VSIFCloseL(), and VSIFOpenL().

Referenced by OSRExportToPCI().

### 11.96.3.16 exportToPrettyWkt()

```
OGRERR OGRSpatialReference::exportToPrettyWkt (
    char ** ppszResult,
    int bSimplify = FALSE ) const
```

Convert this SRS into a nicely formatted WKT string for display to a person.

Consult also the [OGC WKT Coordinate System Issues](#) page for implementation details of WKT in OGR.

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

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

#### Parameters

<i>ppszResult</i>	the resulting string is returned in this pointer.
<i>bSimplify</i>	TRUE if the AXIS, AUTHORITY and EXTENSION nodes should be stripped off.

#### Returns

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

< Success

References Clone(), CPLStrdup(), OGR\_SRSNode::exportToPrettyWkt(), GetRoot(), OGRERR\_NONE, and OGR\_SRSNode::StripNodes().

Referenced by dumpReadable().

### 11.96.3.17 exportToProj4()

```
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()** (p. ??) 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. Starting with GDAL >= 2.2, setting OSR\_USE\_ETMERC to NO will expand to the 'tmerc' projection method (useful with PROJ >= 4.9.3, where utm uses etmerc)

This method is the equivalent of the C function **OSRExportToProj4()** (p. ??).

#### Parameters

<i>ppsProj4</i>	pointer to which dynamically allocated PROJ.4 definition will be assigned.
-----------------	--

#### Returns

OGRErrorNone on success or an error code on failure.

< Unsupported SRS

< Success

< Success



< Unsupported SRS

< Unsupported SRS

< Success

< Failure

< Unsupported SRS

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure



< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure



< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Failure

< Success

References CPLError(), CPLStrdup(), and GetRoot().

### 11.96.3.18 exportToUSGS()

```
OGRERR OGRSpatialReference::exportToUSGS (
    long * piProjSys,
    long * piZone,
    double ** ppadfPrjParams,
    long * piDatum ) const
```

Export coordinate system in USGS GCTP projection definition.

This method is the equivalent of the C function **OSRExportToUSGS()** (p. ??).

#### Parameters

<i>piProjSys</i>	Pointer to variable, where the projection system code will be returned.
<i>piZone</i>	Pointer to variable, where the zone for UTM and State Plane projection systems will be returned.
<i>ppadfPrjParams</i>	Pointer to which dynamically allocated array of 15 projection parameters will be assigned. See <b>importFromUSGS()</b> (p. ??) for the list of parameters. Caller responsible to free this array.
<i>piDatum</i>	Pointer to variable, where the datum code will be returned.

**Returns**

OGRERR\_NONE on success or an error code on failure.

< Success

< Success

References CPLDebug(), CPLDecToPackedDMS(), CPLMalloc(), EQUAL, GetAttrValue(), GetInvFlattening(), GetNormProjParm(), GetSemiMajor(), GetUTMZone(), IsLocal(), OGRERR\_NONE, SRS\_DN\_NAD27, SRS\_DN\_NAD83, SRS\_DN\_WGS84, SRS\_PP\_AZIMUTH, SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_LATITUDE\_OF\_POINT\_1, SRS\_PP\_LATITUDE\_OF\_POINT\_2, SRS\_PP\_LONGITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_POINT\_1, SRS\_PP\_LONGITUDE\_OF\_POINT\_2, SRS\_PP\_SCALE\_FACTOR, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, SRS\_PT\_ALBERS\_CONIC\_EQUAL\_AREA, SRS\_PT\_AZIMUTHAL\_EQUIDISTANT, SRS\_PT\_EQUIDISTANT\_CONIC, SRS\_PT\_EQUIRECTANGULAR, SRS\_PT\_GNOMONIC, SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR, SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR\_TWO\_POINT\_NATURAL\_ORIGIN, SRS\_PT\_LAMBERT\_AZIMUTHAL\_EQUAL\_AREA, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP, SRS\_PT\_MERCATOR\_1SP, SRS\_PT\_MILLER\_CYLINDRICAL, SRS\_PT\_MOLLWEIDE, SRS\_PT\_ORTHOGRAPHIC, SRS\_PT\_POLAR\_STEREOGRAPHIC, SRS\_PT\_POLYCONIC, SRS\_PT\_ROBINSON, SRS\_PT\_SINUSOIDAL, SRS\_PT\_STEREOGRAPHIC, SRS\_PT\_TRANSVERSE\_MERCATOR, SRS\_PT\_VANDERGRINTEN, SRS\_PT\_WAGNER\_IV, and SRS\_PT\_WAGNER\_VII.

Referenced by OSRExportToUSGS().

**11.96.3.19 exportToWkt()**

```
OGRERR OGRSpatialReference::exportToWkt (
    char ** ppszResult ) const
```

Convert this SRS into WKT format.

Consult also the [OGC WKT Coordinate System Issues](#) page for implementation details of WKT in OGR.

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

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

**Parameters**

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

**Returns**

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

< Success

References CPLStrdup(), OGR\_SRSNode::exportToWkt(), and OGRERR\_NONE.

Referenced by Validate().

## 11.96.3.20 exportToXML()

```
OGRERR OGRSpatialReference::exportToXML (
    char ** ,
    const char * = nullptr ) 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()** (p. ??) when no longer needed.

LOCAL\_CS coordinate systems are not translatable. An empty string will be returned along with `OGRERR_NONE`.

This method is the equivalent of the C function **OSRExportToXML()** (p. ??).

## Parameters

<i>ppszRawXML</i>	pointer to which dynamically allocated XML definition will be assigned.
<i>pszDialect</i>	currently ignored. The dialect used is GML based.

## Returns

`OGRERR_NONE` on success or an error code on failure.

< Unsupported SRS

< Success

References `CPLDestroyXMLNode()`, `CPLSerializeXMLTree()`, `IsGeographic()`, `IsProjected()`, `OGRERR_NONE`, and `OGRERR_UNSUPPORTED_SRS`.

Referenced by `OSRExportToXML()`.

## 11.96.3.21 FindMatches()

```
OGRSpatialReferenceH * OGRSpatialReference::FindMatches (
    char ** ppszOptions,
    int * pnEntries,
    int ** ppanMatchConfidence ) const
```

Try to identify a match between the passed SRS and a related SRS in a catalog (currently EPSG only)

Matching may be partial, or may fail. Returned entries will be sorted by decreasing match confidence (first entry has the highest match confidence).

The exact way matching is done may change in future versions.

The current algorithm is:

- try first **AutoidentifyEPSG()** (p. ??). If it succeeds, return the corresponding SRS

- otherwise iterate over all SRS from the EPSG catalog (as found in GDAL pcs.csv and gcs.csv files+esri\_↵extra.wkt), and find those that match the input SRS using the **IsSame()** (p. ??) function (ignoring TOWGS84 clauses)
- if there is a single match using **IsSame()** (p. ??) or one of the matches has the same SRS name, return it with 100% confidence
- if a SRS has the same SRS name, but does not pass the **IsSame()** (p. ??) criteria, return it with 50% confidence.
- otherwise return all candidate SRS that pass the **IsSame()** (p. ??) criteria with a 90% confidence.

A pre-built SRS cache in `~/.gdal/X.Y/srs_cache` will be used if existing, otherwise it will be built at the first run of this function.

This method is the same as **OSRFindMatches()** (p. ??).

#### Parameters

<i>papszOptions</i>	NULL terminated list of options or NULL
<i>pnEntries</i>	Output parameter. Number of values in the returned array.
<i>ppanMatchConfidence</i>	Output parameter (or NULL). *ppanMatchConfidence will be allocated to an array of *pnEntries whose values between 0 and 100 indicate the confidence in the match. 100 is the highest confidence level. The array must be freed with <b>CPLFree()</b> (p. ??).

#### Returns

an array of SRS that match the passed SRS, or NULL. Must be freed with **OSRFreeSRSArray()** (p. ??)

#### Since

GDAL 2.3

< Success

< Success

Referenced by OSRFindMatches().

#### 11.96.3.22 FindProjParm()

```
int OGRSpatialReference::FindProjParm (
    const char * pszParameter,
    const OGR_SRSNode * poPROJCS = nullptr ) const
```

Return the child index of the named projection parameter on its parent PROJCS node.

#### Parameters

<i>pszParameter</i>	projection parameter to look for
<i>poPROJCS</i>	projection CS node to look in. If NULL is passed, the PROJCS node of the SpatialReference object will be searched.

**Returns**

the child index of the named projection parameter. -1 on failure

References EQUAL, GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_LONGITUDE\_OF\_CENTER, and SRS\_PP\_LONGITUDE\_OF\_ORIGIN.

Referenced by GetProjParm(), morphFromESRI(), and morphToESRI().

**11.96.3.23 Fixup()**

```
OGRERR OGRSpatialReference::Fixup ( )
```

Fixup as needed.

Some mechanisms to create WKT using **OGRSpatialReference** (p. ??), and some imported WKT, are not valid according to the OGC CT specification. This method attempts to fill in any missing defaults that are required, and fixup ordering problems (using **OSRFixupOrdering()** (p. ??)) so that the resulting WKT is valid.

This method should be expected to evolve over time to as problems are discovered. The following are among the fixup actions this method will take:

- Fixup the ordering of nodes to match the BNF WKT ordering, using the **FixupOrdering()** (p. ??) method.
- Add missing linear or angular units nodes.

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

**Returns**

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

References CPLAtof(), OGR\_SRSNode::FindChild(), FixupOrdering(), GetAttrNode(), SetAngularUnits(), SetLinearUnits(), SRS\_UA\_DEGREE, SRS\_UA\_DEGREE\_CONV, and SRS\_UL\_METER.

Referenced by morphToESRI().

**11.96.3.24 FixupOrdering()**

```
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.

< Success

References OGR\_SRSNode::FixupOrdering(), GetRoot(), and OGRERR\_NONE.

Referenced by Fixup(), and morphFromESRI().

## 11.96.3.25 FromHandle()

```
static OGRSpatialReference* OGRSpatialReference::FromHandle (
    OGRSpatialReferenceH hSRS ) [inline], [static]
```

Convert a OGRSpatialReferenceH to a OGRSpatialReference\*.

Since

GDAL 2.3

Referenced by OGR\_G\_AssignSpatialReference(), OGR\_G\_CreateFromFgf(), OGR\_G\_CreateFromWkb(), OGR\_G\_TransformTo(), OSRExportToPCI(), OSRExportToUSGS(), OSRExportToXML(), OSRImportFromOzi(), OSRImportFromPCI(), OSRImportFromUSGS(), OSRImportFromXML(), and OSRValidate().

## 11.96.3.26 GetAngularUnits() [1/2]

```
double OGRSpatialReference::GetAngularUnits (
    char ** ppszName ) const
```

Fetch angular geographic coordinate system units.

If no units are available, a value of "degree" and SRS\_UA\_DEGREE\_CONV will be assumed. This method only checks directly under the GEOGCS node for units.

This method does the same thing as the C function **OSRGetAngularUnits()** (p. ??).

## Parameters

<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and should not be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call.
-----------------	--

## Returns

the value to multiply by angular distances to transform them to radians.

Since

GDAL 2.3.0

Referenced by morphToESRI().



## 11.96.3.27 GetAngularUnits() [2/2]

```
double OGRSpatialReference::GetAngularUnits (
    const char ** ppszName = nullptr ) const
```

Fetch angular geographic coordinate system units.

If no units are available, a value of "degree" and SRS\_UA\_DEGREE\_CONV will be assumed. This method only checks directly under the GEOGCS node for units.

This method does the same thing as the C function **OSRGetAngularUnits()** (p. ??).

## Parameters

<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and should not be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call.
-----------------	--

## Returns

the value to multiply by angular distances to transform them to radians.

**Deprecated** GDAL 2.3.0. Use **GetAngularUnits(const char\*\*) const** (p. ??).

References CPLAtof(), EQUAL, GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), and SRS\_UA\_DEGREE\_CONV.

## 11.96.3.28 GetAttrNode() [1/2]

```
OGR_SRSNode * OGRSpatialReference::GetAttrNode (
    const char * pszNodePath )
```

Find named node in tree.

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

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

## Parameters

<i>pszNodePath</i>	the name of the node to search for. May contain multiple components such as "GEOGCS UNIT".
--------------------	--

## Returns

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

References CSLCount(), CSLDestroy(), CSLTokenizeStringComplex(), OGR\_SRSNode::GetNode(), and GetRoot().

Referenced by CloneGeogCS(), CopyGeogCSFrom(), EPSGTreatsAsLatLong(), EPSGTreatsAsNorthingEasting(), exportToPCI(), FindProjParm(), Fixup(), GetAngularUnits(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), GetAxis(), GetExtension(), GetInvFlattening(), GetPrimeMeridian(), GetProjParm(), GetSemiMajor(), GetTargetLinearUnits(), GetTOWGS84(), importFromEPSG(), IsGeographic(), IsProjected(), IsSame(), IsVertical(), morphFromESRI(), morphToESRI(), SetAngularUnits(), SetAuthority(), SetAxes(), SetExtension(), SetGeocCS(), SetGeogCS(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetProjCS(), SetProjection(), SetProjParm(), SetTargetLinearUnits(), SetTOWGS84(), and SetVertCS().

### 11.96.3.29 GetAttrNode() [2/2]

```
const OGR_SRSNode * OGRSpatialReference::GetAttrNode (
    const char * pszNodePath ) const
```

Find named node in tree.

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

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

#### Parameters

<i>pszNodePath</i>	the name of the node to search for. May contain multiple components such as "GEOGCS UNIT".
--------------------	--

#### Returns

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

References GetAttrNode().

### 11.96.3.30 GetAttrValue()

```
const char * OGRSpatialReference::GetAttrValue (
    const char * pszNodeName,
    int iAttr = 0 ) const
```

Fetch indicated attribute of named node.

This method uses **GetAttrNode()** (p. ??) to find the named node, and then extracts the value of the indicated child. Thus a call to GetAttrValue("UNIT",1) would return the second child of the UNIT node, which is normally the length of the linear unit in meters.

This method does the same thing as the C function **OSRGetAttrValue()** (p. ??).

#### Parameters

<i>pszNodeName</i>	the tree node to look for (case insensitive).
<i>iAttr</i>	the child of the node to fetch (zero based).

**Returns**

the requested value, or NULL if it fails for any reason.

References `GetAttrNode()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetChildCount()`, and `OGR_SRSNode::GetValue()`.

Referenced by `AutoidentifyEPSG()`, `convertToOtherProjection()`, `exportToERM()`, `exportToPanorama()`, `exportToPCI()`, `exportToUSGS()`, `GetEPSGGeogCS()`, `GetUTMZone()`, `IsSame()`, `IsSameGeogCS()`, `IsSameVertCS()`, `morphFromESRI()`, and `morphToESRI()`.

**11.96.3.31 GetAuthorityCode()**

```
const char * OGRSpatialReference::GetAuthorityCode (
    const char * pszTargetKey ) const
```

Get the authority code for a node.

This method is used to query an AUTHORITY[] node from within the WKT tree, and fetch the code value.

While in theory values may be non-numeric, for the EPSG authority all code values should be integral.

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

**Parameters**

<i>pszTargetKey</i>	the partial or complete path to the node to get an authority from. i.e. "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()`, `GetAttrNode()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetChildCount()`, and `OGR_SRSNode::GetValue()`.

Referenced by `AutoidentifyEPSG()`, `exportToERM()`, `exportToPCI()`, `GetEPSGGeogCS()`, and `morphToESRI()`.

**11.96.3.32 GetAuthorityName()**

```
const char * OGRSpatialReference::GetAuthorityName (
    const char * pszTargetKey ) const
```

Get the authority name for a node.

This method is used to query an AUTHORITY[] node from within the WKT tree, and fetch the authority name value.

The most common authority is "EPSG".

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

**Parameters**

<i>pszTargetKey</i>	the partial or complete path to the node to get an authority from. i.e. "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(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by AutoidentifyEPSG(), EPSGTreatsAsLatLong(), EPSGTreatsAsNorthingEasting(), exportToERM(), exportToPCI(), GetEPSGGeogCS(), and morphToESRI().

**11.96.3.33 GetAxis()**

```
const char * OGRSpatialReference::GetAxis (
    const char * pszTargetKey,
    int iAxis,
    OGRAxisOrientation * peOrientation ) const
```

Fetch the orientation of one axis.

Fetches the request axis (iAxis - zero based) from the indicated portion of the coordinate system (pszTargetKey) which should be either "GEOGCS" or "PROJCS".

No CPLError is issued on routine failures (such as not finding the AXIS).

This method is equivalent to the C function **OSRGetAxis()** (p. ??).

**Parameters**

<i>pszTargetKey</i>	the coordinate system part to query ("PROJCS" or "GEOGCS").
<i>iAxis</i>	the axis to query (0 for first, 1 for second).
<i>peOrientation</i>	location into which to place the fetch orientation, may be NULL.

**Returns**

the name of the axis or NULL on failure.

References CPLDebug(), EQUAL, GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), OAO\_Down, OAO\_East, OAO\_North, OAO\_Other, OAO\_South, OAO\_Up, and OAO\_West.

### 11.96.3.34 GetEccentricity()

```
double OGRSpatialReference::GetEccentricity ( ) const
```

Get spheroid eccentricity.

#### Returns

eccentricity (or -1 in case of error)

#### Since

GDAL 2.3

< Success

< Success

References GetInvFlattening(), and OGRERR\_NONE.

Referenced by convertToOtherProjection().

### 11.96.3.35 GetEPSGGeogCS()

```
int OGRSpatialReference::GetEPSGGeogCS ( ) const
```

Try to establish what the EPSG code for this coordinate systems GEOGCS might be. Returns -1 if no reasonable guess can be made.

#### Returns

EPSG code

References EQUAL, GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), and GetPrimeMeridian().

Referenced by AutoIdentifyEPSG(), and exportToERM().

### 11.96.3.36 GetExtension()

```
const char * OGRSpatialReference::GetExtension (
    const char * pszTargetKey,
    const char * pszName,
    const char * pszDefault = nullptr ) const
```

Fetch extension value.

Fetch the value of the named EXTENSION item for the identified target node.

## Parameters

<i>pszTargetKey</i>	the name or path to the parent node of the EXTENSION.
<i>pszName</i>	the name of the extension being fetched.
<i>pszDefault</i>	the value to return if the extension is not found.

## Returns

node value if successful or pszDefault on failure.

References EQUAL, GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by IsSame().

## 11.96.3.37 GetInvFlattening()

```
double OGRSpatialReference::GetInvFlattening (
    OGRErr * pnErr = nullptr ) const
```

Get spheroid inverse flattening.

This method does the same thing as the C function **OSRGetInvFlattening()** (p. ??).

## Parameters

<i>pnErr</i>	if non-NULL set to OGRERR_FAILURE if no inverse flattening can be found.
--------------	--

## Returns

inverse flattening, or SRS\_WGS84\_INVFLATTENING if it can't be found.

< Success

< Failure

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), OGRERR\_FAILURE, OGRERR\_NONE, and SRS\_WGS84\_INVFLATTENING.

Referenced by exportToPanorama(), exportToPCI(), exportToUSGS(), GetEccentricity(), GetSemiMinor(), and GetSquaredEccentricity().

## 11.96.3.38 GetLinearUnits() [1/2]

```
double OGRSpatialReference::GetLinearUnits (
    char ** ppszName ) const
```

Fetch linear projection units.

If no units are available, a value of "Meters" and 1.0 will be assumed. This method only checks directly under the PROJCS, GEOCCS or LOCAL\_CS node for units.

This method does the same thing as the C function **OSRGetLinearUnits()** (p. ??)

## Parameters

<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and should not be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call.
-----------------	--

## Returns

the value to multiply by linear distances to transform them to meters.

**Deprecated** GDAL 2.3.0. Use **GetLinearUnits(const char\*\*) const** (p. ??).

References [GetTargetLinearUnits\(\)](#).

Referenced by [AutoIdentifyEPSG\(\)](#), [exportToERM\(\)](#), [exportToPCI\(\)](#), [IsSame\(\)](#), [morphToESRI\(\)](#), and [SetLinearUnitsAndUpdateParameters\(\)](#).

## 11.96.3.39 GetLinearUnits() [2/2]

```
double OGRSpatialReference::GetLinearUnits (
    const char ** ppszName = nullptr ) const
```

Fetch linear projection units.

If no units are available, a value of "Meters" and 1.0 will be assumed. This method only checks directly under the PROJCS, GEOCCS or LOCAL\_CS node for units.

This method does the same thing as the C function **OSRGetLinearUnits()** (p. ??)

## Parameters

<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and should not be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call.
-----------------	--

## Returns

the value to multiply by linear distances to transform them to meters.

## Since

GDAL 2.3.0

References [GetTargetLinearUnits\(\)](#).

### 11.96.3.40 GetNormProjParm()

```
double OGRSpatialReference::GetNormProjParm (
    const char * pszName,
    double dfDefaultValue = 0.0,
    OGRErr * pnErr = nullptr ) const
```

Fetch a normalized projection parameter value.

This method is the same as **GetProjParm()** (p. ??) except that the value of the parameter is "normalized" into degrees or meters depending on whether it is linear or angular.

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

#### Parameters

<i>pszName</i>	the name of the parameter to fetch, from the set of SRS_PP codes in <b>ogr_srs_api.h</b> (p. ??).
<i>dfDefaultValue</i>	the value to return if this parameter doesn't exist.
<i>pnErr</i>	place to put error code on failure. Ignored if NULL.

#### Returns

value of parameter.

< Success

< Success

References **GetProjParm()**, **IsAngularParameter()**, **IsLinearParameter()**, **IsLongitudeParameter()**, and **OGRERR\_↔** NONE.

Referenced by **AutoIdentifyEPSG()**, **convertToOtherProjection()**, **exportToPanorama()**, **exportToPCI()**, **exportToU↔** SGS(), and **GetUTMZone()**.

### 11.96.3.41 GetPrimeMeridian() [1/2]

```
double OGRSpatialReference::GetPrimeMeridian (
    char ** ppszName ) const
```

Fetch prime meridian info.

Returns the offset of the prime meridian from greenwich in degrees, and the prime meridian name (if requested). If no PRIMEM value exists in the coordinate system definition a value of "Greenwich" and an offset of 0.0 is assumed.

If the prime meridian name is returned, the pointer is to an internal copy of the name. It should not be freed, altered or depended on after the next OGR call.

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

#### Parameters

<i>ppszName</i>	return location for prime meridian name. If NULL, name is not returned.
-----------------	---



**Returns**

the offset to the GEOGCS prime meridian from greenwich in decimal degrees.

**Since**

GDAL 2.3.0

Referenced by GetEPSGGeogCS().

**11.96.3.42 GetPrimeMeridian()** [2/2]

```
double OGRSpatialReference::GetPrimeMeridian (
    const char ** ppszName = nullptr ) const
```

Fetch prime meridian info.

Returns the offset of the prime meridian from greenwich in degrees, and the prime meridian name (if requested). If no PRIMEM value exists in the coordinate system definition a value of "Greenwich" and an offset of 0.0 is assumed.

If the prime meridian name is returned, the pointer is to an internal copy of the name. It should not be freed, altered or depended on after the next OGR call.

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

**Parameters**

<i>ppszName</i>	return location for prime meridian name. If NULL, name is not returned.
-----------------	---

**Returns**

the offset to the GEOGCS prime meridian from greenwich in decimal degrees.

**Deprecated** GDAL 2.3.0. Use **GetPrimeMeridian(const char\*\*) const** (p. ??).

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), and SRS\_PM\_GREENWICH.

**11.96.3.43 GetProjParm()**

```
double OGRSpatialReference::GetProjParm (
    const char * pszName,
    double dfDefaultValue = 0.0,
    OGRErr * pnErr = nullptr ) const
```

Fetch a projection parameter value.

NOTE: This code should be modified to translate non degree angles into degrees based on the GEOGCS unit. This has not yet been done.

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

## Parameters

<i>pszName</i>	the name of the parameter to fetch, from the set of SRS_PP codes in <b>ogr_srs_api.h</b> (p. ??).
<i>dfDefaultValue</i>	the value to return if this parameter doesn't exist.
<i>pnErr</i>	place to put error code on failure. Ignored if NULL.

## Returns

value of parameter.

< Success

< Failure

< Failure

References CPLAtof(), FindProjParm(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetValue(), OGRERR\_FAILURE, and OGRERR\_NONE.

Referenced by AutoidentifyEPSG(), GetNormProjParm(), GetUTMZone(), IsSame(), morphFromESRI(), morph↔ToESRI(), and SetLinearUnitsAndUpdateParameters().

## 11.96.3.44 GetReferenceCount()

```
int OGRSpatialReference::GetReferenceCount ( ) const [inline]
```

Fetch current reference count.

## Returns

the current reference count.

## 11.96.3.45 GetSemiMajor()

```
double OGRSpatialReference::GetSemiMajor (
    OGRErr * pnErr = nullptr ) const
```

Get spheroid semi major axis.

This method does the same thing as the C function **OSRGetSemiMajor()** (p. ??).

## Parameters

<i>pnErr</i>	if non-NULL set to OGRERR_FAILURE if semi major axis can be found.
--------------	--

**Returns**

semi-major axis, or SRS\_WGS84\_SEMIMAJOR if it can't be found.

< Success

< Failure

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), OGRERR\_FAILURE, OGRERR\_NONE, and SRS\_WGS84\_SEMIMAJOR.

Referenced by convertToOtherProjection(), exportToPanorama(), exportToPCI(), exportToUSGS(), and GetSemiMinor().

**11.96.3.46 GetSemiMinor()**

```
double OGRSpatialReference::GetSemiMinor (
    OGRERR * pnErr = nullptr ) const
```

Get spheroid semi minor axis.

This method does the same thing as the C function **OSRGetSemiMinor()** (p. ??).

**Parameters**

<i>pnErr</i>	if non-NULL set to OGRERR_FAILURE if semi minor axis can be found.
--------------	--

**Returns**

semi-minor axis, or WGS84 semi minor if it can't be found.

References GetInvFlattening(), GetSemiMajor(), and OSRCalcSemiMinorFromInvFlattening().

**11.96.3.47 GetSquaredEccentricity()**

```
double OGRSpatialReference::GetSquaredEccentricity ( ) const
```

Get spheroid squared eccentricity.

**Returns**

squared eccentricity (or -1 in case of error)

**Since**

GDAL 2.3

< Success

< Success

References GetInvFlattening(), and OGRERR\_NONE.

Referenced by convertToOtherProjection().

**11.96.3.48 GetTargetLinearUnits()** [1/2]

```
double OGRSpatialReference::GetTargetLinearUnits (
    const char * pszTargetKey,
    char ** ppszName ) const
```

Fetch linear units for target.

If no units are available, a value of "Meters" and 1.0 will be assumed.

This method does the same thing as the C function **OSRGetTargetLinearUnits()** (p. ??)

**Parameters**

<i>pszTargetKey</i>	the key to look on. i.e. "PROJCS" or "VERT_CS". Might be NULL, in which case PROJCS will be implied (and if not found, LOCAL_CS, GEOCCS and VERT_CS are looked up)
<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and should not be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call. ppszName can be set to NULL.

**Returns**

the value to multiply by linear distances to transform them to meters.

**Since**

GDAL 2.3.0

Referenced by GetLinearUnits().

**11.96.3.49 GetTargetLinearUnits()** [2/2]

```
double OGRSpatialReference::GetTargetLinearUnits (
    const char * pszTargetKey,
    const char ** ppszName = nullptr ) const
```

Fetch linear units for target.

If no units are available, a value of "Meters" and 1.0 will be assumed.

This method does the same thing as the C function **OSRGetTargetLinearUnits()** (p. ??)

**Parameters**

<i>pszTargetKey</i>	the key to look on. i.e. "PROJCS" or "VERT_CS". Might be NULL, in which case PROJCS will be implied (and if not found, LOCAL_CS, GEOCCS and VERT_CS are looked up)
<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and should not be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call. ppszName can be set to NULL.

**Returns**

the value to multiply by linear distances to transform them to meters.

**Since**

OGR 1.9.0

**Deprecated** GDAL 2.3.0. Use **GetTargetLinearUnits(const char\*, const char\*\*) const** (p. ??).

References CPLAtof(), EQUAL, GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), and IsVertical().

**11.96.3.50 GetTOWGS84()**

```
OGRERR OGRSpatialReference::GetTOWGS84 (
    double * padfCoeff,
    int nCoeffCount = 7 ) const
```

Fetch TOWGS84 parameters, if available.

**Parameters**

<i>padfCoeff</i>	array into which up to 7 coefficients are placed.
<i>nCoeffCount</i>	size of padfCoeff - defaults to 7.

**Returns**

OGRERR\_NONE on success, or OGRERR\_FAILURE if there is no TOWGS84 node available.

< Failure

< Success

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), OGRERR\_FAILURE, and OGRERR\_NONE.

Referenced by exportToPCI(), and IsSameGeogCS().

**11.96.3.51 GetUTMZone()**

```
int OGRSpatialReference::GetUTMZone (
    int * pbNorth = nullptr ) const
```

Get utm zone information.

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

In SWIG bindings (Python, Java, etc) the **GetUTMZone()** (p. ??) method returns a zone which is negative in the southern hemisphere instead of having the pbNorth flag used in the C and C++ interface.

## Parameters

<i>pbNorth</i>	pointer to in to set to TRUE if northern hemisphere, or FALSE if southern.
----------------	--

## Returns

UTM zone number or zero if this isn't a UTM definition.

References EQUAL, GetAttrValue(), GetNormProjParm(), GetProjParm(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_TRANSVERSE\_MERCATOR.

Referenced by AutoidentifyEPSG(), exportToERM(), exportToPanorama(), exportToPCI(), exportToUSGS(), and morphToESRI().

## 11.96.3.52 GetWGS84SRS()

```
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

References OGRSpatialReference(), and SRS\_WKT\_WGS84.

## 11.96.3.53 importFromCRSURL()

```
OGRERR OGRSpatialReference::importFromCRSURL (
    const char * pszURL )
```

Initialize from OGC URL.

Initializes this spatial reference from a coordinate system defined by an OGC URL prefixed with "http://opengis.net/def/crs" per best practice paper 11-135. Currently EPSG and OGC authority values are supported, including OGC auto codes, but not including CRS1 or CRS88 (NAVD88).

This method is also supported through **SetFromUserInput()** (p. ??) which can normally be used for URLs.

## Parameters

<i>pszURL</i>	the URL string.
---------------	-----------------

## Returns

OGRERR\_NONE on success or an error code.

< Failure

< Failure

< Failure

< Success

< Success

References CPLError(), and STARTS\_WITH\_CI.

Referenced by SetFromUserInput().

## 11.96.3.54 importFromDict()

```
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()** (p. ??). Normally this results in searching /usr/local/share/gdal or somewhere similar.

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

## Parameters

<i>pszDictFile</i>	the name of the dictionary file to load.
<i>pszCode</i>	the code to lookup in the dictionary.

## Returns

OGRERR\_NONE on success, or OGRERR\_SRS\_UNSUPPORTED if the code isn't found, and OGRERR\_↵ SRS\_FAILURE if something more dramatic goes wrong.

< Unsupported SRS

< Unsupported SRS

< Unsupported SRS

< Unsupported SRS

< Success

References `CPLFindFile()`, `CPLReadLineL()`, `EQUALN`, `OGRERR_NONE`, `OGRERR_UNSUPPORTED_SRS`, `S↔TARTS_WITH_CI`, `VSIFCloseL()`, and `VSIFOpenL()`.

Referenced by `exportToERM()`, `importFromERM()`, `ImportFromESRIStatePlaneWKT()`, `ImportFromESRI↔WisconsinWKT()`, and `SetFromUserInput()`.

### 11.96.3.55 importFromEPSG()

```
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()`** (p. ??) for details.

This method is relatively expensive, and generally involves quite a bit of text file scanning. Reasonable efforts should be made to avoid calling it many times for the same coordinate system.

This method is similar to **`importFromEPSGA()`** (p. ??) except that EPSG preferred axis ordering will *not* be applied for geographic coordinate systems. EPSG normally defines geographic coordinate systems to use lat/long contrary to typical GIS use). Since OGR 1.10.0, EPSG preferred axis ordering will also *not* be applied for projected coordinate systems that use northing/easting order.

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

#### Parameters

<i>nCode</i>	a GCS or PCS code from the horizontal coordinate system table.
--------------	--

#### Returns

`OGRERR_NONE` on success, or an error code on failure.

< Success



References `EPSGTreatsAsNorthingEasting()`, `GetAttrNode()`, `importFromEPSGA()`, `OGRERR_NONE`, and `OGR_SRSNode::StripNodes()`.

Referenced by `importFromERM()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, `morphFromESRI()`, `SetFromUserInput()`, and `SetWellKnownGeogCS()`.

### 11.96.3.56 importFromEPSGA()

```
OGRERR OGRSpatialReference::importFromEPSGA (
    int nCode )
```

Initialize SRS based on EPSG GCS or PCS code.

This method will initialize the spatial reference based on the passed in EPSG GCS or PCS code.

This method is similar to **importFromEPSG()** (p. ??) except that EPSG preferred axis ordering *will* be applied for geographic and projected coordinate systems. EPSG normally defines geographic coordinate systems to use lat/long, and also there are also a few projected coordinate systems that use northing/easting order contrary to typical GIS use). See **OGRSpatialReference::importFromEPSG()** (p. ??) for more details on operation of this method.

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

#### Parameters

<i>nCode</i>	a GCS or PCS code from the horizontal coordinate system table.
--------------	--

#### Returns

OGRERR\_NONE on success, or an error code on failure.

Referenced by `importFromEPSG()`, `SetFromUserInput()`, and `SetWellKnownGeogCS()`.

### 11.96.3.57 importFromERM()

```
OGRERR OGRSpatialReference::importFromERM (
    const char * pszProj,
    const char * pszDatum,
    const char * pszUnits )
```

Create OGR WKT from ERMapper projection definitions.

Generates an **OGRSpatialReference** (p. ??) definition from an ERMapper datum and projection name. Based on the `ecw_cs.wkt` dictionary file from `gdal/data`.

#### Parameters

<i>pszProj</i>	the projection name, such as "NUTM11" or "GEOGRAPHIC".
<i>pszDatum</i>	the datum name, such as "NAD83".
<i>pszUnits</i>	the linear units "FEET" or "METERS".

**Returns**

OGRERR\_NONE on success or OGRERR\_UNSUPPORTED\_SRS if not found.

< Success

< Success

< Success

< Success

References Clear(), CopyGeogCSFrom(), CPLAtof(), EQUAL, importFromDict(), importFromEPSG(), IsLocal(), OGRERR\_NONE, SetLinearUnits(), SRS\_UL\_METER, SRS\_UL\_US\_FOOT, SRS\_UL\_US\_FOOT\_CONV, and STARTS\_WITH\_CI.

**11.96.3.58 importFromESRI()**

```
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 automatic call to **morphFromESRI()** (p. ??).

Currently only GEOGRAPHIC, UTM, STATEPLANE, GREATBRITIAN\_GRID, ALBERS, EQUIDISTANT\_CONIC, TRANSVERSE (mercator), POLAR, MERCATOR and POLYCONIC projections are supported from old style files.

At this time there is no equivalent exportToESRI() method. Writing old style .prj files is not supported by **OGRSpatialReference** (p. ??). However the **morphToESRI()** (p. ??) and **exportToWkt()** (p. ??) methods can be used to generate output suitable to write to new style (Arc 8) .prj files.

This function is the equivalent of the C function **OSRImportFromESRI()** (p. ??).

**Parameters**

<i>papszPrj</i>	NULL terminated list of strings containing the definition.
-----------------	--

**Returns**

OGRERR\_NONE on success or an error code in case of failure.

< Corrupt data

< Success

< Corrupt data

< Corrupt data

< Corrupt data

< Corrupt data

< Success

References CPLDebug(), CPLFree, CPLRealloc(), CPLStrdup(), EQUAL, importFromWkt(), morphFromESRI(), OGRERR\_CORRUPT\_DATA, OGRERR\_NONE, SetUTM(), and STARTS\_WITH\_CI.

#### 11.96.3.59 ImportFromESRIStatePlaneWKT()

```
OGRERR OGRSpatialReference::ImportFromESRIStatePlaneWKT (
    int nCode,
    const char * pszDatumName,
    const char * pszUnitsName,
    int nPCSCode,
    const char * pszCSName = nullptr )
```

ImportFromESRIStatePlaneWKT < Success

< Failure

< Failure

< Failure

< Failure

References CPLError(), EQUAL, importFromDict(), OGRERR\_FAILURE, and OGRERR\_NONE.

#### 11.96.3.60 ImportFromESRIWisconsinWKT()

```
OGRERR OGRSpatialReference::ImportFromESRIWisconsinWKT (
    const char * pszPrjName,
    double dfCentralMeridian,
    double dfLatOfOrigin,
    const char * pszUnitsName,
    const char * pszCSName = nullptr )
```

ImportFromESRIWisconsinWKT < Success

< Failure

< Failure

< Failure

References EQUAL, importFromDict(), OGRERR\_FAILURE, OGRERR\_NONE, SRS\_PT\_TRANSVERSE\_MERCATOR, and STARTS\_WITH\_CI.

### 11.96.3.61 importFromMlCoordSys()

```
OGRERR OGRSpatialReference::importFromMlCoordSys (
    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 **OSRImportFromMlCoordSys()** (p. ??).

#### Parameters

<i>pszCoordSys</i>	Mapinfo style CoordSys definition string.
--------------------	---

#### Returns

OGRERR\_NONE on success, OGRERR\_FAILURE on failure, OGRERR\_UNSUPPORTED\_OPERATION if MITAB library was not linked in.

< Unsupported operation

References CPLError(), OGRERR\_FAILURE, and OGRERR\_NONE.

### 11.96.3.62 importFromOzi()

```
OGRERR OGRSpatialReference::importFromOzi (
    const char *const * papszLines )
```

Import coordinate system from OziExplorer projection definition.

This method will import projection definition in style, used by OziExplorer software.

#### Parameters

<i>papszLines</i>	Map file lines. This is an array of strings containing the whole OziExplorer .MAP file. The array is terminated by a NULL pointer.
-------------------	--

**Returns**

OGRERR\_NONE on success or an error code in case of failure.

**Since**

OGR 1.10

< Not enough data to deserialize

< Not enough data to deserialize

< Success

< Not enough data to deserialize

< Failure

References Clear(), CPLAtof(), CPLAtofM(), CPLDebug(), CPLError(), CSLCount(), CSLDestroy(), CSLT\_A↔LLOWEMPTYTOKENS, CSLT\_STRIPENDSPACES, CSLT\_STRIPLEADSPACES, CSLTokenizeString2(), CSL↔TokenizeStringComplex(), EQUAL, IsLocal(), OGRERR\_NOT\_ENOUGH\_DATA, SetACEA(), SetLCC(), SetLC↔C1SP(), SetLocalCS(), SetMercator(), SetSinusoidal(), SetTM(), SetUTM(), and STARTS\_WITH\_CI.

Referenced by OSRImportFromOzi().

**11.96.3.63 importFromPanorama()**

```
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()** (p. ??).

**Parameters**

<i>iProjSys</i>	
<i>iDatum</i>	
<i>iEllips</i>	
<i>padfPrjParams</i>	Array of 8 coordinate system parameters:

```
[0] Latitude of the first standard parallel (radians)
[1] Latitude of the second standard parallel (radians)
[2] Latitude of center of projection (radians)
```

```

[3] Longitude of center of projection (radians)
[4] Scaling factor
[5] False Easting
[6] False Northing
[7] Zone number

```

Particular projection uses different parameters, unused ones may be set to zero. If NULL supplied instead of array pointer default values will be used (i.e., zeroes).

#### Returns

OGRERR\_NONE on success or an error code in case of failure.

< Not enough memory

< Success

< Success

References Clear(), CopyGeogCSFrom(), CPLDebug(), CPLError(), CPLMalloc(), importFromEPSG(), IsLocal(), OGRERR\_NONE, OGRERR\_NOT\_ENOUGH\_MEMORY, SetAE(), SetAuthority(), SetCEA(), SetEC(), SetEquiangular(), SetGeogCS(), SetGnomonic(), SetWMPolyconic(), SetLAEA(), SetLCC(), SetLocalCS(), SetMC(), SetMercator(), SetMollweide(), SetPolyconic(), SetPS(), SetStereographic(), SetTM(), SetUTM(), and SetWagner().

#### 11.96.3.64 importFromPCI()

```

OGRERR OGRSpatialReference::importFromPCI (
    const char * pszProj,
    const char * pszUnits = nullptr,
    double * padfPrjParams = nullptr )

```

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

<i>pszProj</i>	NULL terminated string containing the definition. Looks like "pppppppppppp Ennn" or "pppppppppppp Dnnn", where "pppppppppppp" is a projection code, "Ennn" is an ellipsoid code, "Dnnn" — a datum code.
<i>pszUnits</i>	Grid units code ("DEGREE" or "METRE"). If NULL "METRE" will be used.
<i>padfPrjParams</i>	Array of 17 coordinate system parameters:

[0] Spheroid semi major axis [1] Spheroid semi minor axis [2] Reference Longitude [3] Reference Latitude [4] First Standard Parallel [5] Second Standard Parallel [6] False Easting [7] False Northing [8] Scale Factor [9] Height above sphere surface [10] Longitude of 1st point on center line [11] Latitude of 1st point on center line [12] Longitude of 2nd point on center line [13] Latitude of 2nd point on center line [14] Azimuth east of north for center line [15] Landsat satellite number [16] Landsat path number

Particular projection uses different parameters, unused ones may be set to zero. If NULL is supplied instead of an array pointer, default values will be used (i.e., zeroes).

#### Returns

OGRERR\_NONE on success or an error code in case of failure.

< Corrupt data

< Not enough memory

< Success

References Clear(), CopyGeogCSFrom(), CPLAtof(), CPLDebug(), CPLMalloc(), CPLScanLong(), CPLStrnlen(), CSLCount(), CSLDestroy(), EQUAL, EQUALN, importFromEPSG(), IsGeographic(), IsProjected(), OGRERR\_↵  
R\_CORRUPT\_DATA, OGRERR\_NOT\_ENOUGH\_MEMORY, SetACEA(), SetAE(), SetCS(), SetEC(), Set↵  
Equiarectangular2(), SetGnomonic(), SetHOM(), SetHOM2PNO(), SetLAEA(), SetLCC(), SetLCC1SP(), Set↵  
LinearUnits(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetMC(), SetMercator(), SetOrthographic(),  
SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStatePlane(), SetStereographic(), Set↵  
TM(), SetUTM(), SetVDG(), SRS\_UL\_FOOT, SRS\_UL\_FOOT\_CONV, SRS\_UL\_METER, SRS\_UL\_US\_FOOT,  
SRS\_UL\_US\_FOOT\_CONV, STARTS\_WITH\_CI, VSIFCloseL(), and VSIFOpenL().

Referenced by OSRImportFromPCI().

#### 11.96.3.65 importFromProj4()

```
OGRERR OGRSpatialReference::importFromProj4 (
    const char * pszProj4 )
```

Import PROJ.4 coordinate string.

The **OGRSpatialReference** (p. ??) is initialized from the passed PROJ.4 style coordinate system string. In addition to many +proj formulations which have OGC equivalents, it is also possible to import "+init=epsg:n" style definitions. These are passed to **importFromEPSG()** (p. ??). Other init strings (such as the state plane zones) are not currently supported.

Example: pszProj4 = "+proj=utm +zone=11 +datum=WGS84"

Some parameters, such as grids, recognized by PROJ.4 may not be well understood and translated into the **OGR↵  
RSpatialReference** (p. ??) model. It is possible to add the +wktext parameter which is a special keyword that OGR recognized as meaning "embed the entire PROJ.4 string in the WKT and use it literally when converting back to PROJ.4 format".

For example: "+proj=nzmg +lat\_0=-41 +lon\_0=173 +x\_0=2510000 +y\_0=6023150 +ellps=intl +units=m  
+nadgrids=nzgd2kgrid0005.gsb +wktext"

will be translated as :

```
PROJCS["unnamed",
  GEOGCS["International 1909 (Hayford)",
    DATUM["unknown",
      SPHEROID["intl", 6378388, 297]],
    PRIMEM["Greenwich", 0],
    UNIT["degree", 0.0174532925199433]],
  PROJECTION["New_Zealand_Map_Grid"],
  PARAMETER["latitude_of_origin", -41],
  PARAMETER["central_meridian", 173],
  PARAMETER["false_easting", 2510000],
  PARAMETER["false_northing", 6023150],
  UNIT["Meter", 1],
  EXTENSION["PROJ4", "+proj=nzmg +lat_0=-41 +lon_0=173 +x_0=2510000  
+y_0=6023150 +ellps=intl +units=m +nadgrids=nzgd2kgrid0005.gsb +wktext"]]
```

Special processing for 'etmerc' (GDAL >= 1.10 ): if +proj=etmerc is found in the passed string, the SRS built will use the WKT representation for a standard Transverse Mercator, but will also include a PROJ4 EXTENSION node to preserve the etmerc projection method.

For example: "+proj=etmerc +lat\_0=0 +lon\_0=9 +k=0.9996 +units=m +x\_0=500000 +datum=WGS84"

will be translated as :

```
PROJCS["unnamed",
  GEOGCS["WGS 84",
    DATUM["WGS_1984",
      SPHEROID["WGS 84",6378137,298.257223563,
        AUTHORITY["EPSG","7030"]],
      TOWGS84[0,0,0,0,0,0,0],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
      AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
      AUTHORITY["EPSG","9108"]],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0],
  PARAMETER["central_meridian",9],
  PARAMETER["scale_factor",0.9996],
  PARAMETER["false_easting",500000],
  PARAMETER["false_northing",0],
  UNIT["Meter",1],
  EXTENSION["PROJ4","+proj=etmerc +lat_0=0 +lon_0=9 +k=0.9996 +units=m +x_0=500000 +datum=WGS84 +nodefs"]
]
```

This method is the equivalent of the C function **OSRImportFromProj4()** (p. ??).

#### Parameters

<i>pszProj4</i>	the PROJ.4 style string.
-----------------	--------------------------

#### Returns

OGRERR\_NONE on success or OGRERR\_CORRUPT\_DATA on failure.

< Success

< Corrupt data

< Corrupt data

< Corrupt data



< Success

< Success

< Unsupported SRS

< Success

References Clear(), CPLDebug(), CPLDMSToDec(), CPLFree, CPLSPrintf(), CPLStrdup(), CSLAddNameValue(), CSLDestroy(), CSLFetchNameValue(), OGR\_SRSNode::DestroyChild(), EQUAL, OGR\_SRSNode::FindChild(), GetRoot(), importFromEPSG(), OGRERR\_CORRUPT\_DATA, OGRERR\_NONE, SetBonne(), SetCEA(), SetCS(), SetGeocCS(), SetNZMG(), SetTM(), and SetTMSO().

Referenced by SetFromUserInput().

### 11.96.3.66 importFromUrl()

```
OGRERR OGRSpatialReference::importFromUrl (
    const char * pszUrl )
```

Set spatial reference from a URL.

This method will download the spatial reference at a given URL and feed it into SetFromUserInput for you.

This method does the same thing as the **OSRImportFromUrl()** (p. ??) function.

#### Parameters

<i>pszUrl</i>	text definition to try to deduce SRS from.
---------------	--

#### Returns

OGRERR\_NONE on success, or an error code with the curl error message if it is unable to download data.

< Failure

< Failure

< Failure

< Failure

< Failure

< Success

< Failure

< Success

References CPLError(), and STARTS\_WITH\_CI.

Referenced by SetFromUserInput().

### 11.96.3.67 importFromURN()

```
OGRERR OGRSpatialReference::importFromURN (
    const char * pszURN )
```

Initialize from OGC URN.

Initializes this spatial reference from a coordinate system defined by an OGC URN prefixed with "urn:ogc:def:crs:" per recommendation paper 06-023r1. Currently EPSG and OGC authority values are supported, including OGC auto codes, but not including CRS1 or CRS88 (NAVD88).

This method is also support through **SetFromUserInput()** (p. ??) which can normally be used for URNs.

#### Parameters

<i>pszURN</i>	the urn string.
---------------	-----------------

#### Returns

OGRERR\_NONE on success or an error code.

< Failure

< Failure

< Success

< Success

References CPLError(), and STARTS\_WITH\_CI.

Referenced by SetFromUserInput().

## 11.96.3.68 importFromUSGS()

```
OGRErr OGRSpatialReference::importFromUSGS (
    long iProjSys,
    long iZone,
    double * padfPrjParams,
    long iDatum,
    int nUSGSAngleFormat = TRUE )
```

Import coordinate system from USGS projection definition.

This method will import projection definition in style, used by USGS GCTP software. GCTP operates on angles in packed DMS format (see **CPLDecToPackedDMS()** (p. ??) function for details), so all angle values (latitudes, longitudes, azimuths, etc.) specified in the padfPrjParams array should be in the packed DMS format, unless bAnglesInPackedDMSFormat is set to FALSE.

This function is the equivalent of the C function **OSRImportFromUSGS()** (p. ??). Note that the bAnglesInPackedDMSFormat parameter is only present in the C++ method. The C function assumes bAnglesInPackedFormat = TRUE.

## Parameters

<i>iProjSys</i>	Input projection system code, used in GCTP.
<i>iZone</i>	Input zone for UTM and State Plane projection systems. For Southern Hemisphere UTM use a negative zone code. iZone ignored for all other projections.
<i>padfPrjParams</i>	

```
----- | Array Element | Code & Projection Id |----- | 8 | 9 | 10 |
11 | 12 | ----- 0 Geographic | | | | | 1 U T M | | | | | 2 State Plane | | | | | 3
Albers Equal Area | | | | | 4 Lambert Conformal C | | | | | 5 Mercator | | | | | 6 Polar Stereographic | | | | | 7
Polyconic | | | | | 8 Equid. Conic A |zero | | | | | Equid. Conic B |one | | | | | 9 Transverse Mercator | | | | | 10
Stereographic | | | | | 11 Lambert Azimuthal | | | | | 12 Azimuthal | | | | | 13 Gnomonic | | | | | 14 Orthographic
| | | | | 15 Gen. Vert. Near Per | | | | | 16 Sinusoidal | | | | | 17 Equirectangular | | | | | 18 Miller Cylindrical |
| | | | | 19 Van der Grinten | | | | | 20 Hotin Oblique Merc A |Long1|Lat1|Long2|Lat2|zero| Hotin Oblique Merc B |
| | | | | one | 21 Robinson | | | | | 22 Space Oblique Merc A |PSRev|LRat|PFlag| |zero| Space Oblique Merc B | |
| | | | | one | 23 Alaska Conformal | | | | | 24 Interrupted Goode | | | | | 25 Mollweide | | | | | 26 Interrupt Mollweide
| | | | | 27 Hammer | | | | | 28 Wagner IV | | | | | 29 Wagner VII | | | | | 30 Oblated Equal Area |Angle| | | | |
-----
```

where

Lon/Z	Longitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
Lat/Z	Latitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
SMajor	Semi-major axis of ellipsoid. If zero, Clarke 1866 in meters is assumed.
SMinor	Eccentricity squared of the ellipsoid if less than zero, if zero, a spherical form is assumed, or if greater than zero, the semi-minor axis of ellipsoid.
Sphere	Radius of reference sphere. If zero, 6370997 meters is used.
STDPAR	Latitude of the standard parallel
STDPR1	Latitude of the first standard parallel
STDPR2	Latitude of the second standard parallel
CentMer	Longitude of the central meridian
OriginLat	Latitude of the projection origin

FE	False easting in the same units as the semi-major axis
FN	False northing in the same units as the semi-major axis
TrueScale	Latitude of true scale
LongPol	Longitude down below pole of map
Factor	Scale factor at central meridian (Transverse Mercator) or center of projection (Hotine Oblique Mercator)
CentLon	Longitude of center of projection
CenterLat	Latitude of center of projection
Height	Height of perspective point
Long1	Longitude of first point on center line (Hotine Oblique Mercator, format A)
Long2	Longitude of second point on center line (Hotine Oblique Mercator, format A)
Lat1	Latitude of first point on center line (Hotine Oblique Mercator, format A)
Lat2	Latitude of second point on center line (Hotine Oblique Mercator, format A)
AziAng	Azimuth angle east of north of center line (Hotine Oblique Mercator, format B)
AzmthPt	Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, format B)
IncAng	Inclination of orbit at ascending node, counter-clockwise from equator (SOM, format A)
AscLong	Longitude of ascending orbit at equator (SOM, format A)
PSRev	Period of satellite revolution in minutes (SOM, format A)
LRat	Landsat ratio to compensate for confusion at northern end of orbit (SOM, format A -- use 0.5201613)
PFlag	End of path flag for Landsat: 0 = start of path, 1 = end of path (SOM, format A)
Satnum	Landsat Satellite Number (SOM, format B)
Path	Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and WRS-2 for Landsat 4, 5 and 6.) (SOM, format B)
Shapem	Oblated Equal Area oval shape parameter m
Shapen	Oblated Equal Area oval shape parameter n
Angle	Oblated Equal Area oval rotation angle

Array elements 13 and 14 are set to zero. All array elements with blank fields are set to zero too.

#### Parameters

<i>iDatum</i>	Input spheroid.
---------------	-----------------

If the datum code is negative, the first two values in the parameter array (parm) are used to define the values as follows:

- If `padfPrjParams[0]` is a non-zero value and `padfPrjParams[1]` is greater than one, the semimajor axis is set to `padfPrjParams[0]` and the semiminor axis is set to `padfPrjParams[1]`.
- If `padfPrjParams[0]` is nonzero and `padfPrjParams[1]` is greater than zero but less than or equal to one, the semimajor axis is set to `padfPrjParams[0]` and the semiminor axis is computed from the eccentricity squared value `padfPrjParams[1]`:  

$$\text{semiminor} = \sqrt{1.0 - \text{ES}} * \text{semimajor}$$
 where  

$$\text{ES} = \text{eccentricity squared}$$
- If `padfPrjParams[0]` is nonzero and `padfPrjParams[1]` is equal to zero, the semimajor axis and semiminor axis are set to `padfPrjParams[0]`.

- If `padfPrjParams[0]` equals zero and `padfPrjParams[1]` is greater than zero, the default Clarke 1866 is used to assign values to the semimajor axis and semiminor axis.
- If `padfPrjParams[0]` and `padfPrjParams[1]` equals zero, the semimajor axis is set to 6370997.0 and the semiminor axis is set to zero.

If a datum code is zero or greater, the semimajor and semiminor axis are defined by the datum code as found in the following table:

<h4>Supported Datums</h4>

```

0: Clarke 1866 (default)
1: Clarke 1880
2: Bessel
3: International 1967
4: International 1909
5: WGS 72
6: Everest
7: WGS 66
8: GRS 1980/WGS 84
9: Airy
10: Modified Everest
11: Modified Airy
12: Walbeck
13: Southeast Asia
14: Australian National
15: Krassovsky
16: Hough
17: Mercury 1960
18: Modified Mercury 1968
19: Sphere of Radius 6370997 meters

```

#### Parameters

<i>nUSGSAngleFormat</i>	one of USGS_ANGLE_DECIMALDEGREES, USGS_ANGLE_PACKEDDMS, or USGS_ANGLE_RADIANS (default is USGS_ANGLE_PACKEDDMS).
-------------------------	--

#### Returns

OGRERR\_NONE on success or an error code in case of failure.

< Corrupt data

< Angle is in decimal degrees.

< Angle is in radians.

< Corrupt data

< Success

< Success

< Success

< Success

#### Parameters

<i>nUSGSAngleFormat</i>	Angle is in packed degree minute second.
-------------------------	--

References CPLError(), CPLPackedDMSToDec(), OGRERR\_CORRUPT\_DATA, SetUTM(), USGS\_ANGLE\_DE↵CIMALDEGREES, and USGS\_ANGLE\_RADIANS.

Referenced by OSRImportFromUSGS().

#### 11.96.3.69 importFromWkt() [1/3]

```
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.

Consult also the [OGC WKT Coordinate System Issues](#) page for implementation details of WKT in OGR.

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

#### Parameters

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

#### Returns

OGRERR\_NONE if import succeeds, or OGRERR\_CORRUPT\_DATA if it fails for any reason.

**Deprecated** GDAL 2.3. Use **importFromWkt(const char\*\*)** (p. ??) or **importFromWkt(const char\*)** (p. ??)

Referenced by importFromESRI(), importFromWkt(), morphToESRI(), OGRSpatialReference(), SetFromUser↵Input(), and SetWellKnownGeogCS().

11.96.3.70 `importFromWkt()` [2/3]

```
OGRERR OGRSpatialReference::importFromWkt (
    const 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.

Consult also the [OGC WKT Coordinate System Issues](#) page for implementation details of WKT in OGR.

This method is the same as the C function **OSRImportFromWkt()** (p. ??).

## Parameters

<i>ppszInput</i>	Pointer to pointer to input. The pointer is updated to point to remaining unused input text.
------------------	--

## Returns

OGRERR\_NONE if import succeeds, or OGRERR\_CORRUPT\_DATA if it fails for any reason.

## Since

GDAL 2.3

< Failure

< Success

< Success

References `OGR_SRSNode::AddChild()`, `Clear()`, `OGRERR_FAILURE`, and `OGRERR_NONE`.

11.96.3.71 `importFromWkt()` [3/3]

```
OGRERR OGRSpatialReference::importFromWkt (
    const char * pszInput )
```

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.

Consult also the [OGC WKT Coordinate System Issues](#) page for implementation details of WKT in OGR.

## Parameters

<i>pszInput</i>	Input WKT
-----------------	-----------

**Returns**

OGRERR\_NONE if import succeeds, or OGRERR\_CORRUPT\_DATA if it fails for any reason.

**Since**

GDAL 2.3

References importFromWkt().

**11.96.3.72 importFromWMSAUTO()**

```
OGRERR OGRSpatialReference::importFromWMSAUTO (
    const char * pszDefinition )
```

Initialize from WMSAUTO string.

Note that the WMS 1.3 specification does not include the units code, while apparently earlier specs do. We try to guess around this.

**Parameters**

<i>pszDefinition</i>	the WMSAUTO string
----------------------	--------------------

**Returns**

OGRERR\_NONE on success or an error code.

< Failure

< Failure

< Failure

< Success

References CPLAtof(), CPLError(), CSLCount(), CSLDestroy(), CSLTokenizeStringComplex(), and STARTS\_WI↔TH\_CI.

Referenced by SetFromUserInput().

**11.96.3.73 importFromXML()**

```
OGRERR OGRSpatialReference::importFromXML (
    const char * pszXML )
```

Import coordinate system from XML format (GML only currently).

This method is the same as the C function **OSRImportFromXML()** (p. ??)



## Parameters

<i>pszXML</i>	XML string to import
---------------	----------------------

## Returns

OGRERR\_NONE on success or OGRERR\_CORRUPT\_DATA on failure.

< Corrupt data  
< Unsupported SRS

References Clear(), CPLDestroyXMLNode(), CPLParseXMLString(), CPLStripXMLNamespace(), EQUAL, OGRERR\_CORRUPT\_DATA, OGRERR\_UNSUPPORTED\_SRS, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by OSRImportFromXML(), and SetFromUserInput().

## 11.96.3.74 IsAngularParameter()

```
int OGRSpatialReference::IsAngularParameter (
    const char * pszParameterName ) [static]
```

Is the passed projection parameter an angular one?

## Returns

TRUE or FALSE

References EQUAL, SRS\_PP\_AZIMUTH, SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_RECTIFIED\_GRID\_ANGLE, and STARTS\_WITH\_CI.

Referenced by GetNormProjParm(), and SetNormProjParm().

## 11.96.3.75 IsCompound()

```
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 EQUAL, and OGR\_SRSNode::GetValue().

**11.96.3.76 IsGeocentric()**

```
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 EQUAL, and OGR\_SRSNode::GetValue().

Referenced by CloneGeogCS(), CopyGeogCSFrom(), and SetGeogCS().

**11.96.3.77 IsGeographic()**

```
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 EQUAL, GetAttrNode(), GetRoot(), and OGR\_SRSNode::GetValue().

Referenced by AutoidentifyEPSG(), EPSGTreatsAsLatLong(), exportToERM(), exportToXML(), importFromPCI(), SetVertCS(), and SetWellKnownGeogCS().

**11.96.3.78 IsLinearParameter()**

```
int OGRSpatialReference::IsLinearParameter (
    const char * pszParameterName ) [static]
```

Is the passed projection parameter an linear one measured in meters or some similar linear measure.

**Returns**

TRUE or FALSE

References EQUAL, SRS\_PP\_SATELLITE\_HEIGHT, and STARTS\_WITH\_CI.

Referenced by GetNormProjParm(), SetLinearUnitsAndUpdateParameters(), and SetNormProjParm().

**11.96.3.79 IsLocal()**

```
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.

References EQUAL, and GetRoot().

Referenced by exportToPanorama(), exportToPCI(), exportToUSGS(), importFromERM(), importFromOzi(), importFromPanorama(), and IsSame().

**11.96.3.80 IsLongitudeParameter()**

```
int OGRSpatialReference::IsLongitudeParameter (
    const char * pszParameterName ) [static]
```

Is the passed projection parameter an angular longitude (relative to a prime meridian)?

**Returns**

TRUE or FALSE

References EQUAL, SRS\_PP\_CENTRAL\_MERIDIAN, and STARTS\_WITH\_CI.

Referenced by GetNormProjParm(), and SetNormProjParm().

**11.96.3.81 IsProjected()**

```
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 EQUAL, GetAttrNode(), and OGR\_SRSNode::GetValue().

Referenced by AutoidentifyEPSG(), EPSGTreatsAsNorthingEasting(), exportToERM(), exportToXML(), importFromPCI(), IsSame(), and SetVertCS().

**11.96.3.82 IsSame()** [1/2]

```
int OGRSpatialReference::IsSame (
    const OGRSpatialReference * poOtherSRS ) const
```

Do these two spatial references describe the same system ?

## Parameters

<i>poOtherSRS</i>	the SRS being compared to.
-------------------	----------------------------

## Returns

TRUE if equivalent or FALSE otherwise.

Referenced by OGRGeomFieldDefn::IsSame(), IsSame(), and OGRGeometryFactory::transformWithOptions().

## 11.96.3.83 IsSame() [2/2]

```
int OGRSpatialReference::IsSame (
    const OGRSpatialReference * poOtherSRS,
    const char *const * papszOptions ) const
```

Do these two spatial references describe the same system ?

## Parameters

<i>poOtherSRS</i>	the SRS being compared to.
<i>papszOptions</i>	options. DATUM=STRICT/IGNORE. TOWGS84=STRICT/ONLY_IF_IN_BOTH/IGNORE

## Returns

TRUE if equivalent or FALSE otherwise.

References convertToOtherProjection(), CPLDebug(), EQUAL, GetAttrNode(), GetAttrValue(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetExtension(), GetLinearUnits(), GetProjParm(), GetRoot(), OGR\_SRSNode::GetValue(), IsLocal(), IsProjected(), IsSame(), IsSameGeogCS(), IsSameVertCS(), IsVertical(), SRS\_PP\_AZIMUTH, SRS\_PP\_RECTIFIED\_GRID\_ANGLE, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP, and STARTS\_WITH\_CI.

## 11.96.3.84 IsSameGeogCS() [1/2]

```
int OGRSpatialReference::IsSameGeogCS (
    const OGRSpatialReference * poOther ) const
```

Do the GeogCS'es match?

This method is the same as the C function **OSRIsSameGeogCS()** (p. ??).

## Parameters

<i>poOther</i>	the SRS being compared against.
----------------	---------------------------------

**Returns**

TRUE if they are the same or FALSE otherwise.

Referenced by `IsSame()`.

**11.96.3.85 IsSameGeogCS()** [2/2]

```
int OGRSpatialReference::IsSameGeogCS (
    const OGRSpatialReference * poOther,
    const char *const * papszOptions ) const
```

Do the GeogCS'es match?

This method is the same as the C function **OSRIsSameGeogCS()** (p. ??).

**Parameters**

<i>poOther</i>	the SRS being compared against.
<i>papszOptions</i>	options. DATUM=STRICT/IGNORE. TOWGS84=STRICT/ONLY_IF_IN_BOTH/IGNORE

**Returns**

TRUE if they are the same or FALSE otherwise.

< Success

< Success

References `CPLAtof()`, `CPLDebug()`, `CSLFetchNameValueDef()`, `EQUAL`, `GetAttrValue()`, `GetTOWGS84()`, `OGR↵`  
`ERR_NONE`, and `SRS_UA_DEGREE_CONV`.

**11.96.3.86 IsSameVertCS()**

```
int OGRSpatialReference::IsSameVertCS (
    const OGRSpatialReference * poOther ) const
```

Do the VertCS'es match?

This method is the same as the C function **OSRIsSameVertCS()** (p. ??).

**Parameters**

<i>poOther</i>	the SRS being compared against.
----------------	---------------------------------

**Returns**

TRUE if they are the same or FALSE otherwise.

References CPLAtof(), EQUAL, and GetAttrValue().

Referenced by IsSame().

**11.96.3.87 IsVertical()**

```
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 EQUAL, GetAttrNode(), and OGR\_SRSNode::GetValue().

Referenced by GetTargetLinearUnits(), IsSame(), SetCompoundCS(), and SetTargetLinearUnits().

**11.96.3.88 morphFromESRI()**

```
OGRERR OGRSpatialReference::morphFromESRI ( )
```

Convert in place from ESRI WKT format.

The value notes of this coordinate system are modified in various manners to adhere more closely to the WKT standard. This mostly involves translating a variety of ESRI names for projections, arguments and datums to "standard" names, as defined by Adam Gawne-Cain's reference translation of EPSG to WKT for the CT specification.

Starting with GDAL 1.9.0, missing parameters in TOWGS84, DATUM or GEOGCS nodes can be added to the WKT, comparing existing WKT parameters to GDAL's databases. Note that this optional procedure is very conservative and should not introduce false information into the WKT definition (although caution should be advised when activating it). Needs the Configuration Option GDAL\_FIX\_ESRI\_WKT be set to one of the following values (TOWGS84 is recommended for proper datum shift calculations):

**GDAL\_FIX\_ESRI\_WKT values**

<b>TOWGS84</b>	Adds missing TOWGS84 parameters (necessary for datum transformations), based on named datum and spheroid values.
<b>DATUM</b>	Adds EPSG AUTHORITY nodes and sets SPHEROID name to OGR spec.
<b>GEOGCS</b>	Adds EPSG AUTHORITY nodes and sets GEOGCS, DATUM and SPHEROID names to OGR spec. Effectively replaces GEOGCS node with the result of importFromEPSG(n), using EPSG code n corresponding to the existing GEOGCS. Does not impact PROJCS values.

This does the same as the C function **OSRMorphFromESRI()** (p. ??).

#### Returns

OGRERR\_NONE unless something goes badly wrong.

< Success

< Success

< Success

References OGR\_SRSNode::applyRemapper(), CloneGeogCS(), CPL\_ARRAYSIZE, CPLDebug(), CPLFree, CPLGetConfigOption(), CPLStrdup(), CSLGetField(), OGR\_SRSNode::DestroyChild(), EQUAL, FindProjParm(), FixupOrdering(), GetAttrNode(), GetAttrValue(), OGR\_SRSNode::GetChild(), GetProjParm(), GetRoot(), OGR\_SRSNode::GetValue(), importFromEPSG(), OGRERR\_NONE, SetNode(), SetProjParm(), OGR\_SRSNode::SetValue(), SRS\_PP\_AZIMUTH, SRS\_PP\_RECTIFIED\_GRID\_ANGLE, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, SRS\_PT\_AZIMUTHAL\_EQUIDISTANT, SRS\_PT\_EQUIDISTANT\_CONIC, SRS\_PT\_LAMBERT\_AZIMUTHAL\_EQUAL\_AREA, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP, SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP, SRS\_PT\_MERCATOR\_2SP, SRS\_PT\_MERCATOR\_AUXILIARY\_SPHERE, SRS\_PT\_OBLIQUE\_STEREOGRAPHIC, SRS\_PT\_POLAR\_STEREOGRAPHIC, SRS\_PT\_ROBINSON, SRS\_PT\_SINUSOIDAL, STARTS\_WITH\_CI, and StripCTParms().

Referenced by importFromESRI(), and SetFromUserInput().

#### 11.96.3.89 morphToESRI()

**OGRERR** OGRSpatialReference::morphToESRI ( )

Convert in place to ESRI WKT format.

The value nodes of this coordinate system are modified in various manners more closely map onto the ESRI concept of WKT format. This includes renaming a variety of projections and arguments, and stripping out nodes not recognised by ESRI (like AUTHORITY and AXIS).

This does the same as the C function **OSRMorphToESRI()** (p. ??).

#### Returns

OGRERR\_NONE unless something goes badly wrong.

< Success

< Success

< Success

< Success

References OGR\_SRSNode::applyRemapper(), Clear(), convertToOtherProjection(), CPLFree, CPLStrdup(), OGR\_SRSNode::DestroyChild(), EQUAL, FindProjParm(), Fixup(), GetAngularUnits(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), GetRoot(), GetUTMZone(), OGR\_SRSNode::GetValue(), importFromWkt(), OGRERR\_NONE, SetNode(), OGR\_SRSNode::SetValue(), SRS\_PP\_AZIMUTH, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_RECTIFIED\_GRID\_ANGLE, SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR, SRS\_PT\_MERCATOR\_1SP, SRS\_PT\_MERCATOR\_2SP, SRS\_PT\_OBLIQUE\_STEREOGRAPHIC, SRS\_PT\_POLAR\_STEREOGRAPHIC, STARTS\_WITH\_CI, and StripCTParms().

**11.96.3.90 operator=()**

```
OGRSpatialReference & OGRSpatialReference::operator= (
    const OGRSpatialReference & oSource )
```

Assignment operator.

**Parameters**

<i>oSource</i>	SRS to assign to *this
----------------	------------------------

**Returns**

\*this

References Clear(), and OGR\_SRSNode::Clone().

**11.96.3.91 Reference()**

```
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(), OGRGeometry::OGRGeometry(), and OGRGeom↵  
FieldDefn::SetSpatialRef().

**11.96.3.92 Release()**

```
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().



## 11.96.3.93 SetACEA()

```

OGRERR OGRSpatialReference::SetACEA (
    double dfStdP1,
    double dfStdP2,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Albers Conic Equal Area < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_CENTER, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, and SRS\_PT\_ALBERS\_CONIC\_EQUAL\_AREA.

Referenced by importFromOzi(), and importFromPCI().

## 11.96.3.94 SetAE()

```

OGRERR OGRSpatialReference::SetAE (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Azimuthal Equidistant < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_CENTER, and SRS\_PT\_AZIMUTHAL\_EQUIDISTANT.

Referenced by importFromPanorama(), and importFromPCI().

## 11.96.3.95 SetAngularUnits()

```

OGRERR OGRSpatialReference::SetAngularUnits (
    const char * pszUnitsName,
    double dfInRadians )

```

Set the angular units for the geographic coordinate system.

This method creates a UNIT subnode with the specified values as a child of the GEOGCS node.

This method does the same as the C function **OSRSetAngularUnits()** (p. ??).

## Parameters

<i>pszUnitsName</i>	the units name to be used. Some preferred units names can be found in <b>ogr_srs_api.h</b> (p. ??) such as SRS_UA_DEGREE.
<i>dfInRadians</i> Generated by Doxygen	the value to multiply by an angle in the indicated units to transform to radians. Some standard conversion factors can be found in <b>ogr_srs_api.h</b> (p. ??).

**Returns**

OGRERR\_NONE on success.

< Failure

< Failure

< Success

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGRERR\_FAILURE, OGRERR\_NONE, and OGR\_SRSNode::SetValue().

Referenced by CloneGeogCS(), and Fixup().

**11.96.3.96 SetAuthority()**

```
OGRErr OGRSpatialReference::SetAuthority (
    const char * pszTargetKey,
    const char * pszAuthority,
    int nCode )
```

Set the authority for a node.

This method is the same as the C function **OSRSetAuthority()** (p. ??).

**Parameters**

<i>pszTargetKey</i>	the partial or complete path to the node to set an authority on. i.e. "PROJCS", "GEOGCS" or "GEOGCS UNIT".
<i>pszAuthority</i>	authority name, such as "EPSG".
<i>nCode</i>	code for value with this authority.

**Returns**

OGRERR\_NONE on success.

< Failure

< Success

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGRERR\_FAILURE, and OGRERR\_NONE.

Referenced by AutoIdentifyEPSG(), and importFromPanorama().

## 11.96.3.97 SetAxes()

```

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()** (p. ??).

## Parameters

<i>pszTargetKey</i>	either "PROJCS" or "GEOGCS", must already exist in SRS.
<i>pszXAxisName</i>	name of first axis, normally "Long" or "Easting".
<i>eXAxisOrientation</i>	normally OAO_East.
<i>pszYAxisName</i>	name of second axis, normally "Lat" or "Northing".
<i>eYAxisOrientation</i>	normally OAO_North.

## Returns

OGRERR\_NONE on success or an error code.

< Failure  
< Success

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGRERR\_FAILURE, OGRERR\_NONE, and OSRAxisEnumToName().

## 11.96.3.98 SetBonne()

```

OGRErr OGRSpatialReference::SetBonne (
    double dfStdPl,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Bonne < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_STANDARD\_PARALLEL\_1, and SRS\_PT\_BONNE.

Referenced by importFromProj4().

## 11.96.3.99 SetCEA()

```

OGRERR OGRSpatialReference::SetCEA (
    double dfStdPl,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Cylindrical Equal Area < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_STANDARD\_PARALLEL\_1, and SRS\_PT\_CYLI↵NDRICAL\_EQUAL\_AREA.

Referenced by importFromPanorama(), and importFromProj4().

## 11.96.3.100 SetCompoundCS()

```

OGRERR OGRSpatialReference::SetCompoundCS (
    const char * pszName,
    const OGRSpatialReference * poHorizSRS,
    const OGRSpatialReference * poVertSRS )

```

Setup a compound coordinate system.

This method is the same as the C function **OSRSetCompoundCS()** (p. ??).

This method is replace the current SRS with a COMPD\_CS coordinate system consisting of the passed in horizontal and vertical coordinate systems.

## Parameters

<i>pszName</i>	the name of the compound coordinate system.
<i>poHorizSRS</i>	the horizontal SRS (PROJCS or GEOGCS).
<i>poVertSRS</i>	the vertical SRS (VERT_CS).

## Returns

OGRERR\_NONE on success.

< Failure

< Failure

< Success

References CPLError(), and IsVertical().

## 11.96.3.101 SetCS()

```
OGRERR OGRSpatialReference::SetCS (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Cassini-Soldner < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PT\_CASSINI\_SOLDNER.

Referenced by importFromPCI(), and importFromProj4().

## 11.96.3.102 SetEC()

```
OGRERR OGRSpatialReference::SetEC (
    double dfStdP1,
    double dfStdP2,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Equidistant Conic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_CENTER, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, and SRS\_PT\_EQUIDISTANT\_CONIC.

Referenced by importFromPanorama(), and importFromPCI().

## 11.96.3.103 SetEckert()

```
OGRERR OGRSpatialReference::SetEckert (
    int nVariation,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Eckert I < Unsupported SRS  
< Success

References CPLError(), SetProjection(), SRS\_PT\_ECKERT\_I, SRS\_PT\_ECKERT\_II, SRS\_PT\_ECKERT\_III, SRS\_PT\_ECKERT\_IV, SRS\_PT\_ECKERT\_V, and SRS\_PT\_ECKERT\_VI.

**11.96.3.104 SetEckertIV()**

```
OGRERR OGRSpatialReference::SetEckertIV (
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Eckert IV < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, and SRS\_PT\_ECKERT\_IV.

**11.96.3.105 SetEckertVI()**

```
OGRERR OGRSpatialReference::SetEckertVI (
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Eckert VI < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, and SRS\_PT\_ECKERT\_VI.

**11.96.3.106 SetEquirectangular()**

```
OGRERR OGRSpatialReference::SetEquirectangular (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Equirectangular < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PT\_EQUIRECTANGULAR.

Referenced by importFromPanorama().

## 11.96.3.107 SetEquirectangular2()

```

OGRERR OGRSpatialReference::SetEquirectangular2 (
    double dfCenterLat,
    double dfCenterLong,
    double dfPseudoStdParallel1,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Equirectangular generalized form : < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_STANDARD\_↵\_PARALLEL\_1, and SRS\_PT\_EQUIRECTANGULAR.

Referenced by importFromPCI().

## 11.96.3.108 SetExtension()

```

OGRERR OGRSpatialReference::SetExtension (
    const char * pszTargetKey,
    const char * pszName,
    const char * pszValue )

```

Set extension value.

Set the value of the named EXTENSION item for the identified target node.

## Parameters

<i>pszTargetKey</i>	the name or path to the parent node of the EXTENSION.
<i>pszName</i>	the name of the extension being fetched.
<i>pszValue</i>	the value to set

## Returns

OGRERR\_NONE on success

< Failure

< Success

< Success

References OGR\_SRSNode::AddChild(), EQUAL, GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode\_↵::GetChildCount(), OGR\_SRSNode::GetValue(), OGRERR\_FAILURE, OGRERR\_NONE, and OGR\_SRSNode::\_↵SetValue().

## 11.96.3.109 SetFromUserInput()

```
OGRERR OGRSpatialReference::SetFromUserInput (
    const char * pszDefinition )
```

Set spatial reference from various text formats.

This method will examine the provided input, and try to deduce the format, and then use it to initialize the spatial reference system. It may take the following forms:

1. Well Known Text definition - passed on to **importFromWkt()** (p. ??).
2. "EPSG:n" - number passed on to **importFromEPSG()** (p. ??).
3. "EPSGA:n" - number passed on to **importFromEPSGA()** (p. ??).
4. "AUTO:proj\_id,unit\_id,lon0,lat0" - WMS auto projections.
5. "urn:ogc:def:crs:EPSG::n" - ogc urns
6. PROJ.4 definitions - passed on to **importFromProj4()** (p. ??).
7. filename - file read for WKT, XML or PROJ.4 definition.
8. well known name accepted by **SetWellKnownGeogCS()** (p. ??), such as NAD27, NAD83, WGS84 or WGS72.
9. WKT (directly or in a file) in ESRI format should be prefixed with ESRI:: to trigger an automatic **morphFromESRI()** (p. ??).
10. "IGNF:xxx" - "+init=IGNF:xxx" passed on to **importFromProj4()** (p. ??).

It is expected that this method will be extended in the future to support XML and perhaps a simplified "minilanguage" for indicating common UTM and State Plane definitions.

This method is intended to be flexible, but by its nature it is imprecise as it must guess information about the format intended. When possible applications should call the specific method appropriate if the input is known to be in a particular format.

This method does the same thing as the **OSRSetFromUserInput()** (p. ??) function.

## Parameters

<i>pszDefinition</i>	text definition to try to deduce SRS from.
----------------------	--

## Returns

OGRERR\_NONE on success, or an error code if the name isn't recognised, the definition is corrupt, or an EPSG value can't be successfully looked up.

< Success



< Success

< Success

< Success

< Success

< Corrupt data

< Failure

< Success

< Success

References OGR\_SRSNode::AddChild(), Clear(), OGR\_SRSNode::Clone(), CPLDebug(), CPLFree, CPLMalloc(), CPLStrdup(), EQUAL, OGR\_SRSNode::GetChild(), GetRoot(), OGR\_SRSNode::GetValue(), importFromCRSU↵RL(), importFromDict(), importFromEPSG(), importFromEPSGA(), importFromProj4(), importFromUrl(), import↵FromURN(), importFromWkt(), importFromWMSAUTO(), importFromXML(), morphFromESRI(), OGRERR\_COR↵RUPT\_DATA, OGRERR\_FAILURE, OGRERR\_NONE, SetNode(), SetWellKnownGeogCS(), STARTS\_WITH\_CI, VSIFCloseL(), VSIFOpenL(), and VSIFReadL().

#### 11.96.3.110 SetGaussSchreiberTMercator()

```
OGRERR OGRSpatialReference::SetGaussSchreiberTMercator (
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Gauss Schreiber Transverse Mercator < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FAC↵TOR, and SRS\_PT\_GAUSSSCHREIBERTMERCATOR.

## 11.96.3.111 SetGeocCS()

```
OGRERR OGRSpatialReference::SetGeocCS (
    const char * pszName )
```

Set the user visible GEOCCS name.

This method is the same as the C function **OSRSetGeocCS()** (p. ??).

This method will ensure a GEOCCS node is created as the root, and set the provided name on it. If used on a GEOGCS coordinate system, the DATUM and PRIMEM nodes from the GEOGCS will be transferred over to the GEOGCS.

## Parameters

<i>pszName</i>	the user visible name to assign. Not used as a key.
----------------	---

## Returns

OGRERR\_NONE on success.

## Since

OGR 1.9.0

< Failure  
< Success

References OGR\_SRSNode::Clone(), CPLDebug(), EQUAL, GetAttrNode(), OGR\_SRSNode::GetNode(), GetRoot(), OGR\_SRSNode::GetValue(), OGR\_SRSNode::InsertChild(), OGRERR\_FAILURE, OGRERR\_NONE, and SetNode().

Referenced by importFromProj4().

## 11.96.3.112 SetGeogCS()

```
OGRERR OGRSpatialReference::SetGeogCS (
    const char * pszGeogName,
    const char * pszDatumName,
    const char * pszSpheroidName,
    double dfSemiMajor,
    double dfInvFlattening,
    const char * pszPMName = nullptr,
    double dfPMOffset = 0.0,
    const char * pszAngularUnits = nullptr,
    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 its own to establish a geographic spatial reference, or applied to a projected coordinate system to establish the underlying geographic coordinate system.

This method does the same as the C function **OSRSetGeogCS()** (p. ??).

## Parameters

<i>pszGeogName</i>	user visible name for the geographic coordinate system (not to serve as a key).
<i>pszDatumName</i>	key name for this datum. The OpenGIS specification lists some known values, and otherwise EPSG datum names with a standard transformation are considered legal keys.
<i>pszSpheroidName</i>	user visible spheroid name (not to serve as a key)
<i>dfSemiMajor</i>	the semi major axis of the spheroid.
<i>dfInvFlattening</i>	the inverse flattening for the spheroid. This can be computed from the semi minor axis as $1/f = 1.0 / (1.0 - \text{semiminor}/\text{semimajor})$ .
<i>pszPMName</i>	the name of the prime meridian (not to serve as a key) If this is NULL a default value of "Greenwich" will be used.
<i>dfPMOffset</i>	the longitude of Greenwich relative to this prime meridian.
<i>pszAngularUnits</i>	the angular units name (see <code>ogr_srs_api.h</code> (p. ??) for some standard names). If NULL a value of "degrees" will be assumed.
<i>dfConvertToRadians</i>	value to multiply angular units by to transform them to radians. A value of <code>SRS_UA_DEGREE_CONV</code> will be used if <code>pszAngularUnits</code> is NULL.

## Returns

OGRERR\_NONE on success.

< Failure  
< Success

References `OGR_SRSNode::AddChild()`, `Clear()`, `CopyGeogCSFrom()`, `CPLAtof()`, `OGR_SRSNode::DestroyChild()`, `EQUAL`, `OGR_SRSNode::FindChild()`, `GetAttrNode()`, `GetRoot()`, `OGR_SRSNode::InsertChild()`, `IsGeocentric()`, `OGRERR_FAILURE`, `OGRERR_NONE`, `SetGeogCS()`, `SetRoot()`, `SRS_PM_GREENWICH`, `SRS_UA_DEGREE`, and `SRS_UA_DEGREE_CONV`.

Referenced by `importFromPanorama()`, and `SetGeogCS()`.

## 11.96.3.113 SetGEOS()

```
OGRERR OGRSpatialReference::SetGEOS (
    double dfCentralMeridian,
    double dfSatelliteHeight,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Geostationary Satellite < Success

References `OGRERR_NONE`, `SetNormProjParm()`, `SetProjection()`, `SRS_PP_CENTRAL_MERIDIAN`, `SRS_PP_FALSE_EASTING`, `SRS_PP_FALSE_NORTHING`, `SRS_PP_SATELLITE_HEIGHT`, and `SRS_PT_GEOSTATIONARY_SATELLITE`.

**11.96.3.114 SetGH()**

```
OGRERR OGRSpatialReference::SetGH (
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Goode Homolosine < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, and SRS\_PT\_GOODE\_HOMOLOSIONE.

**11.96.3.115 SetGnomonic()**

```
OGRERR OGRSpatialReference::SetGnomonic (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Gnomonic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PT\_GNOMO\_↵NIC.

Referenced by importFromPanorama(), and importFromPCI().

**11.96.3.116 SetGS()**

```
OGRERR OGRSpatialReference::SetGS (
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Gall Stereographic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, and SRS\_PT\_GALL\_STEREOGRAPHIC.

**11.96.3.117 SetHOM()**

```
OGRERR OGRSpatialReference::SetHOM (
    double dfCenterLat,
    double dfCenterLong,
    double dfAzimuth,
    double dfRectToSkew,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Set a Hotine Oblique Mercator projection using azimuth angle.

Hotine Oblique Mercator

This projection corresponds to EPSG projection method 9812, also sometimes known as hotine oblique mercator (variant A)..

This method does the same thing as the C function **OSRSetHOM()** (p. ??).

## Parameters

<i>dfCenterLat</i>	Latitude of the projection origin.
<i>dfCenterLong</i>	Longitude of the projection origin.
<i>dfAzimuth</i>	Azimuth, measured clockwise from North, of the projection centerline.
<i>dfRectToSkew</i>	?
<i>dfScale</i>	Scale factor applies to the projection origin.
<i>dfFalseEasting</i>	False easting.
<i>dfFalseNorthing</i>	False northing.

## Returns

OGRERR\_NONE on success.

< Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_AZIMUTH, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_CENTER, SRS\_PP\_RECTIFIED\_GRID\_ANGLE, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR.

Referenced by importFromPCI().

## 11.96.3.118 SetHOM2PNO()

```

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.

Hotine Oblique Mercator 2 points

This method does the same thing as the C function **OSRSetHOM2PNO()** (p. ??).

## Parameters

<i>dfCenterLat</i>	Latitude of the projection origin.
<i>dfLat1</i>	Latitude of the first point on center line.
<i>dfLong1</i>	Longitude of the first point on center line.
<i>dfLat2</i>	Latitude of the second point on center line.
<i>dfLong2</i>	Longitude of the second point on center line.
<i>dfScale</i>	Scale factor applies to the projection origin.
<i>dfFalseEasting</i>	False easting.
<i>dfFalseNorthing</i>	False northing.

**Returns**

OGRERR\_NONE on success.

< Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LATITUDE\_OF\_POINT\_1, SRS\_PP\_LATITUDE\_OF\_POINT\_2, SRS\_PP\_LONGITUDE\_OF\_POINT\_1, SRS\_PP\_LONGITUDE\_OF\_POINT\_2, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR\_TWO\_POINT\_NATURAL\_ORIGIN.

Referenced by importFromPCI().

**11.96.3.119 SetHOMAC()**

```
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()** (p. ??).

**Parameters**

<i>dfCenterLat</i>	Latitude of the projection origin.
<i>dfCenterLong</i>	Longitude of the projection origin.
<i>dfAzimuth</i>	Azimuth, measured clockwise from North, of the projection centerline.
<i>dfRectToSkew</i>	?
<i>dfScale</i>	Scale factor applies to the projection origin.
<i>dfFalseEasting</i>	False easting.
<i>dfFalseNorthing</i>	False northing.

**Returns**

OGRERR\_NONE on success.

< Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_AZIMUTH, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_CENTER, SRS\_PP\_RECTIFIED\_GRID\_ANGLE, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR\_AZIMUTH\_CENTER.

**11.96.3.120 SetIGH()**

**OGRERR** OGRSpatialReference::SetIGH ( )

Interrupted Goode Homolosine < Success

References OGRERR\_NONE, SetProjection(), and SRS\_PT\_IGH.

**11.96.3.121 SetIWMPolyconic()**

**OGRERR** OGRSpatialReference::SetIWMPolyconic (   
     double *dfLat1*,   
     double *dfLat2*,   
     double *dfCenterLong*,   
     double *dfFalseEasting*,   
     double *dfFalseNorthing* )

International Map of the World Polyconic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_1ST\_POINT, SRS\_PP\_LATITUD↵E\_OF\_2ND\_POINT, and SRS\_PT\_IMW\_POLYCONIC.

Referenced by importFromPanorama().

**11.96.3.122 SetKrovak()**

**OGRERR** OGRSpatialReference::SetKrovak (   
     double *dfCenterLat*,   
     double *dfCenterLong*,   
     double *dfAzimuth*,   
     double *dfPseudoStdParallelLat*,   
     double *dfScale*,   
     double *dfFalseEasting*,   
     double *dfFalseNorthing* )

Krovak Oblique Conic Conformal < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_AZIMUTH, SRS\_PP\_FALSE\_EA↵STING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_CEN↵TER, SRS\_PP\_PSEUDO\_STD\_PARALLEL\_1, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_KROVAK.

**11.96.3.123 SetLAEA()**

```
OGRERR OGRSpatialReference::SetLAEA (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Lambert Azimuthal Equal-Area < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_CENTER, and SRS\_PT\_LAMBERT\_AZIMUTHAL\_EQUAL\_AREA.

Referenced by importFromPanorama(), and importFromPCI().

**11.96.3.124 SetLCC()**

```
OGRERR OGRSpatialReference::SetLCC (
    double dfStdP1,
    double dfStdP2,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Lambert Conformal Conic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, and SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP.

Referenced by convertToOtherProjection(), importFromOzi(), importFromPanorama(), and importFromPCI().

**11.96.3.125 SetLCC1SP()**

```
OGRERR OGRSpatialReference::SetLCC1SP (
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Lambert Conformal Conic 1SP < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP.

Referenced by importFromOzi(), and importFromPCI().



## 11.96.3.126 SetLCCB()

```

OGRERR OGRSpatialReference::SetLCCB (
    double dfStdP1,
    double dfStdP2,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Lambert Conformal Conic (Belgium) < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_STANDARD\_PARALLEL\_1, SRS\_PP\_STANDARD\_PARALLEL\_2, and SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP\_BELGIUM.

## 11.96.3.127 SetLinearUnits()

```

OGRERR OGRSpatialReference::SetLinearUnits (
    const char * pszUnitsName,
    double dfInMeters )

```

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the PROJCS, GEOCCS or LOCAL\_CS node.

This method does the same as the C function **OSRSetLinearUnits()** (p. ??).

## Parameters

<i>pszUnitsName</i>	the units name to be used. Some preferred units names can be found in <b>ogr_srs_api.h</b> (p. ??) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
<i>dfInMeters</i>	the value to multiply by a length in the indicated units to transform to meters. Some standard conversion factors can be found in <b>ogr_srs_api.h</b> (p. ??).

## Returns

OGRERR\_NONE on success.

References SetTargetLinearUnits().

Referenced by Fixup(), importFromERM(), importFromPCI(), and SetLinearUnitsAndUpdateParameters().

**11.96.3.128 SetLinearUnitsAndUpdateParameters()**

```
OGRERR OGRSpatialReference::SetLinearUnitsAndUpdateParameters (
    const char * pszName,
    double dfInMeters )
```

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the PROJCS or LOCAL\_CS node. It works the same as the **SetLinearUnits()** (p. ??) method, but it also updates all existing linear projection parameter values from the old units to the new units.

**Parameters**

<i>pszName</i>	the units name to be used. Some preferred units names can be found in <b>ogr_srs_api.h</b> (p. ??) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
<i>dfInMeters</i>	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in <b>ogr_srs_api.h</b> (p. ??).

**Returns**

OGRERR\_NONE on success.

< Failure

References CPLFree, CPLStrdup(), EQUAL, GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), OGR\_SRSNode::GetValue(), IsLinearParameter(), OGRERR\_FAILURE, SetLinearUnits(), and SetProjParm().

Referenced by importFromPCI().

**11.96.3.129 SetLocalCS()**

```
OGRERR OGRSpatialReference::SetLocalCS (
    const char * pszName )
```

Set the user visible LOCAL\_CS name.

This method is the same as the C function **OSRSetLocalCS()** (p. ??).

This method will ensure a LOCAL\_CS node is created as the root, and set the provided name on it. It must be used before **SetLinearUnits()** (p. ??).

**Parameters**

<i>pszName</i>	the user visible name to assign. Not used as a key.
----------------	---

**Returns**

OGRERR\_NONE on success.

< Failure  
 < Success

References CPLDebug(), GetAttrNode(), GetRoot(), OGRERR\_FAILURE, OGRERR\_NONE, and SetNode().

Referenced by importFromOzi(), importFromPanorama(), and importFromPCI().

### 11.96.3.130 SetMC()

```
OGRERR OGRSpatialReference::SetMC (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Miller Cylindrical < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, SRS\_PP\_LONGITUDE\_OF\_CENTER, and SRS\_PT\_MILLER\_CYLINDRICAL.

Referenced by importFromPanorama(), and importFromPCI().

### 11.96.3.131 SetMercator()

```
OGRERR OGRSpatialReference::SetMercator (
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Mercator 1SP < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_MERCATOR\_1SP.

Referenced by convertToOtherProjection(), importFromOzi(), importFromPanorama(), and importFromPCI().

**11.96.3.132 SetMercator2SP()**

```

OGRERR OGRSpatialReference::SetMercator2SP (
    double dfStdPl,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Mercator 2SP < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_STANDARD\_↵\_PARALLEL\_1, and SRS\_PT\_MERCATOR\_2SP.

Referenced by convertToOtherProjection().

**11.96.3.133 SetMollweide()**

```

OGRERR OGRSpatialReference::SetMollweide (
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Mollweide < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, and SRS\_PT\_MOLLWEIDE.

Referenced by importFromPanorama().

**11.96.3.134 SetNode()** [1/2]

```

OGRERR OGRSpatialReference::SetNode (
    const char * pszNodePath,
    const char * pszNewNodeValue )

```

Set attribute value in spatial reference.

Missing intermediate nodes in the path will be created if not already in existence. If the attribute has no children one will be created and assigned the value otherwise the zeroth child will be assigned the value.

This method does the same as the C function **OSRSetAttrValue()** (p. ??).

**Parameters**

<i>pszNodePath</i>	full path to attribute to be set. For instance "PROJCS GEOGCS UNIT".
<i>pszNewNodeValue</i>	value to be assigned to node, such as "meter". This may be NULL if you just want to force creation of the intermediate path.

**Returns**

OGRERR\_NONE on success.

< Failure  
< Success

References OGR\_SRSNode::AddChild(), CSLCount(), CSLDestroy(), CSLTokenizeStringComplex(), EQUAL, OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetRoot(), OGR\_SRSNode::GetValue(), OGRERR\_FAILURE, OGRERR\_NONE, SetRoot(), and OGR\_SRSNode::SetValue().

Referenced by morphFromESRI(), morphToESRI(), SetFromUserInput(), SetGeocCS(), SetLocalCS(), SetNode(), SetProjCS(), and SetProjection().

**11.96.3.135 SetNode()** [2/2]

```
OGRERR OGRSpatialReference::SetNode (
    const char * pszNodePath,
    double dfValue )
```

Set attribute value in spatial reference.

Missing intermediate nodes in the path will be created if not already in existence. If the attribute has no children one will be created and assigned the value otherwise the zeroth child will be assigned the value.

This method does the same as the C function **OSRSetAttrValue()** (p. ??).

**Parameters**

<i>pszNodePath</i>	full path to attribute to be set. For instance "PROJCS GEOGCS UNIT".
<i>dfValue</i>	value to be assigned to node.

**Returns**

OGRERR\_NONE on success.

References SetNode().

**11.96.3.136 SetNormProjParm()**

```
OGRERR OGRSpatialReference::SetNormProjParm (
    const char * pszName,
    double dfValue )
```

Set a projection parameter with a normalized value.

This method is the same as **SetProjParm()** (p. ??) except that the value of the parameter passed in is assumed to be in "normalized" form (decimal degrees for angular values, meters for linear values. The values are converted in a form suitable for the GEOGCS and linear units in effect.

This method is the same as the C function **OSRSetNormProjParm()** (p. ??).

## Parameters

<i>pszName</i>	the parameter name, which should be selected from the macros in <b>ogr_srs_api.h</b> (p. ??), such as SRS_PP_CENTRAL_MERIDIAN.
<i>dfValue</i>	value to assign.

## Returns

OGRERR\_NONE on success.

References IsAngularParameter(), IsLinearParameter(), IsLongitudeParameter(), and SetProjParm().

Referenced by SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckertIV(), SetEckertV(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetHOMAC(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMercator2SP(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetQSC(), SetRobinson(), SetSCH(), SetSinusoidal(), SetSOC(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetVDG(), and SetWagner().

## 11.96.3.137 SetNZMG()

```
OGRERR OGRSpatialReference::SetNZMG (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

New Zealand Map Grid < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PT\_NEW\_ZEALAND\_MAP\_GRID.

Referenced by importFromProj4().

## 11.96.3.138 SetOrthographic()

```
OGRERR OGRSpatialReference::SetOrthographic (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Orthographic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PT\_ORTHOGRAPHIC.

Referenced by importFromPCI().

## 11.96.3.139 SetOS()

```
OGRERR OGRSpatialReference::SetOS (
    double dfOriginLat,
    double dfCMeridian,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Oblique Stereographic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FAC↵TOR, and SRS\_PT\_OBLIQUE\_STEREOGRAPHIC.

Referenced by importFromPCI().

## 11.96.3.140 SetPolyconic()

```
OGRERR OGRSpatialReference::SetPolyconic (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Polyconic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_↵\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PT\_POLYC↵ONIC.

Referenced by importFromPanorama(), and importFromPCI().

## 11.96.3.141 SetProjCS()

```
OGRERR OGRSpatialReference::SetProjCS (
    const char * pszName )
```

Set the user visible PROJCS name.

This method is the same as the C function **OSRSetProjCS()** (p. ??).

This method will ensure a PROJCS node is created as the root, and set the provided name on it. If used on a GEOGCS coordinate system, the GEOGCS node will be demoted to be a child of the new PROJCS root.

## Parameters

<i>pszName</i>	the user visible name to assign. Not used as a key.
----------------	---

**Returns**

OGRERR\_NONE on success.

< Failure  
< Success

References CPLDebug(), EQUAL, GetAttrNode(), GetRoot(), OGR\_SRSNode::GetValue(), OGR\_SRSNode::InsertChild(), OGRERR\_FAILURE, OGRERR\_NONE, and SetNode().

**11.96.3.142 SetProjection()**

```
OGRERR OGRSpatialReference::SetProjection (
    const char * pszProjection )
```

Set a projection name.

This method is the same as the C function **OSRSetProjection()** (p. ??).

**Parameters**

<i>pszProjection</i>	the projection name, which should be selected from the macros in <b>ogr_srs_api.h</b> (p. ??), such as SRS_PT_TRANSVERSE_MERCATOR.
----------------------	--

**Returns**

OGRERR\_NONE on success.

< Success  
< Success

References EQUAL, GetAttrNode(), OGR\_SRSNode::GetValue(), OGR\_SRSNode::InsertChild(), OGRERR\_NONE, and SetNode().

Referenced by SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEckertIV(), SetEckertVI(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetHOMAC(), SetIGH(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMercator2SP(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetQSC(), SetRobinson(), SetSCH(), SetSinusoidal(), SetSOC(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetVDG(), and SetWagner().

**11.96.3.143 SetProjParm()**

```
OGRERR OGRSpatialReference::SetProjParm (
    const char * pszParmName,
    double dfValue )
```



Set a projection parameter value.

Adds a new PARAMETER under the PROJCS with the indicated name and value.

This method is the same as the C function **OSRSetProjParm()** (p. ??).

Please check [http://www.remotesensing.org/geotiff/proj\\_list](http://www.remotesensing.org/geotiff/proj_list) pages for legal parameter names for specific projections.

## Parameters

<i>pszParmName</i>	the parameter name, which should be selected from the macros in <b>ogr_srs_api.h</b> (p. ??), such as SRS_PP_CENTRAL_MERIDIAN.
<i>dfValue</i>	value to assign.

## Returns

OGRERR\_NONE on success.

< Failure

< Success

< Success

References OGR\_SRSNode::AddChild(), EQUAL, GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), OGRERR\_FAILURE, OGRERR\_NONE, and OGR\_SRSNode::SetValue().

Referenced by morphFromESRI(), SetLinearUnitsAndUpdateParameters(), and SetNormProjParm().

## 11.96.3.144 SetPS()

```
OGRERR OGRSpatialReference::SetPS (
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Polar Stereographic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_POLAR\_STEREOGRAPHIC.

Referenced by importFromPanorama(), and importFromPCI().

## 11.96.3.145 SetQSC()

```
OGRERR OGRSpatialReference::SetQSC (
    double dfCenterLat,
    double dfCenterLong )
```

Quadrilateralized Spherical Cube < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PT\_QSC.

## 11.96.3.146 SetRobinson()

```

OGRERR OGRSpatialReference::SetRobinson (
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Robinson < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LONGITUDE\_OF\_CENTER, and SRS\_PT\_ROBINSON.

Referenced by importFromPCI().

## 11.96.3.147 SetRoot()

```

void OGRSpatialReference::SetRoot (
    OGR_SRSNode * poNewRoot )

```

Set the root SRS node.

If the object has an existing tree of OGR\_SRSNodes, they are destroyed as part of assigning the new root. Ownership of the passed **OGR\_SRSNode** (p. ??) is assumed by the **OGRSpatialReference** (p. ??).

## Parameters

<i>poNewRoot</i>	object to assign as root.
------------------	---------------------------

Referenced by CloneGeogCS(), CopyGeogCSFrom(), SetGeogCS(), SetNode(), SetVertCS(), and StripVertical().

## 11.96.3.148 SetSCH()

```

OGRERR OGRSpatialReference::SetSCH (
    double dfPegLat,
    double dfPegLong,
    double dfPegHeading,
    double dfPegHgt )

```

Spherical, Cross-track, Height < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_PEG\_POINT\_HEADING, SRS\_PP\_PEG\_POINT\_HEIGHT, SRS\_PP\_PEG\_POINT\_LATITUDE, SRS\_PP\_PEG\_POINT\_LONGITUDE, and SRS\_PT\_SCH.

**11.96.3.149 SetSinusoidal()**

```
OGRERR OGRSpatialReference::SetSinusoidal (
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Sinusoidal < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LONGITUDE\_OF\_CENTER, and SRS\_PT\_SINUSOIDAL.

Referenced by importFromOzi(), and importFromPCI().

**11.96.3.150 SetSOC()**

```
OGRERR OGRSpatialReference::SetSOC (
    double dfLatitudeOfOrigin,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Swiss Oblique Cylindrical < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_CENTER, and SRS\_PT\_SWISS\_OBLIQUE\_CYLINDRICAL.

**11.96.3.151 SetStatePlane()**

```
OGRERR OGRSpatialReference::SetStatePlane (
    int nZone,
    int bNAD83 = TRUE,
    const char * pszOverrideUnitName = nullptr,
    double dfOverrideUnit = 0.0 )
```

Set State Plane projection definition.

State Plane

This will attempt to generate a complete definition of a state plane zone based on generating the entire SRS from the EPSG tables. If the EPSG tables are unavailable, it will produce a stubbed LOCAL\_CS definition and return OGRERR\_FAILURE.

This method is the same as the C function **OSRSetStatePlaneWithUnits()** (p. ??).

**Parameters**

<i>nZone</i>	State plane zone number, in the USGS numbering scheme (as distinct from the Arc/Info and Erdas numbering scheme).
<i>bNAD83</i>	TRUE if the NAD83 zone definition should be used or FALSE if the NAD27 zone definition should be used. <span style="float: right;">Generated by Doxygen</span>
<i>pszOverrideUnitName</i>	Linear unit name to apply overriding the legal definition for this zone.
<i>dfOverrideUnit</i>	Linear unit conversion factor to apply overriding the legal definition for this zone.

## Returns

OGRERR\_NONE on success, or OGRERR\_FAILURE on failure, mostly likely due to the EPSG tables not being accessible.

< Failure

< Failure

< Success

< Success

References CPLError(), and OGRERR\_FAILURE.

Referenced by importFromPCI().

## 11.96.3.152 SetStereographic()

```
OGRERR OGRSpatialReference::SetStereographic (
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Stereographic < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_STEREOGRAPHIC.

Referenced by importFromPanorama(), and importFromPCI().

## 11.96.3.153 SetTargetLinearUnits()

```
OGRERR OGRSpatialReference::SetTargetLinearUnits (
    const char * pszTargetKey,
    const char * pszUnitsName,
    double dfInMeters )
```

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the target node.

This method does the same as the C function **OSRSetTargetLinearUnits()** (p. ??).

## Parameters

<i>pszTargetKey</i>	the keyword to set the linear units for. i.e. "PROJCS" or "VERT_CS"
<i>pszUnitsName</i>	the units name to be used. Some preferred units names can be found in <b>ogr_srs_api.h</b> (p. ??) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
<b>Generated by Doxygen</b> <i>dfInMeters</i>	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in <b>ogr_srs_api.h</b> (p. ??).

**Returns**

OGRERR\_NONE on success.

**Since**

OGR 1.9.0

< Failure

< Failure

< Failure

< Success

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), IsVertical(), OGRERR\_FAILURE, OGRERR\_NONE, and OGR\_SRSNode::SetValue().

Referenced by SetLinearUnits().

**11.96.3.154 SetTM()**

```
OGRERR OGRSpatialReference::SetTM (
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Transverse Mercator < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_TRANSVERSE\_MERCATOR.

Referenced by importFromOzi(), importFromPanorama(), importFromPCI(), and importFromProj4().

**11.96.3.155 SetTMG()**

```
OGRERR OGRSpatialReference::SetTMG (
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Tunesia Mining Grid < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PT\_TUNISIA\_MINING\_GRID.

## 11.96.3.156 SetTMSO()

```
OGRERR OGRSpatialReference::SetTMSO (
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Transverse Mercator (South Oriented) < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PP\_SCALE\_FACTOR, and SRS\_PT\_TRANSVERSE\_MERCATOR\_SOUTH\_ORIENTED.

Referenced by importFromProj4().

## 11.96.3.157 SetTMVariant()

```
OGRERR OGRSpatialReference::SetTMVariant (
    const char * pszVariantName,
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Transverse Mercator variants. < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_ORIGIN, and SRS\_PP\_SCALE\_FACTOR.

## 11.96.3.158 SetTOWGS84()

```
OGRERR OGRSpatialReference::SetTOWGS84 (
    double dfDX,
    double dfDY,
    double dfDZ,
    double dfEX = 0.0,
    double dfEY = 0.0,
    double dfEZ = 0.0,
    double dfPPM = 0.0 )
```

Set the Bursa-Wolf conversion to WGS84.

This will create the TOWGS84 node as a child of the DATUM. It will fail if there is no existing DATUM node. Unlike most **OGRSpatialReference** (p. ??) methods it will insert itself in the appropriate order, and will replace an existing TOWGS84 node if there is one.

The parameters have the same meaning as EPSG transformation 9606 (Position Vector 7-param. transformation).

This method is the same as the C function **OSRSetTOWGS84()** (p. ??).

## Parameters

<i>dfDX</i>	X child in meters.
<i>dfDY</i>	Y child in meters.
<i>dfDZ</i>	Z child in meters.
<i>dfEX</i>	X rotation in arc seconds (optional, defaults to zero).
<i>dfEY</i>	Y rotation in arc seconds (optional, defaults to zero).
<i>dfEZ</i>	Z rotation in arc seconds (optional, defaults to zero).
<i>dfPPM</i>	scaling factor (parts per million).

## Returns

OGRERR\_NONE on success.

< Failure

< Success

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::InsertChild(), OGRERR\_FAILURE, and OGRERR\_NONE.

## 11.96.3.159 SetTPED()

```
OGRERR OGRSpatialReference::SetTPED (
    double dfLat1,
    double dfLong1,
    double dfLat2,
    double dfLong2,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Two Point Equidistant < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, SRS\_PP\_LATITUDE\_OF\_1ST\_POINT, SRS\_PP\_LATITUDE\_OF\_2ND\_POINT, SRS\_PP\_LONGITUDE\_OF\_1ST\_POINT, SRS\_PP\_LONGITUDE\_OF\_2ND\_POINT, and SRS\_PT\_TWO\_POINT\_EQUIDISTANT.

## 11.96.3.160 SetUTM()

```
OGRERR OGRSpatialReference::SetUTM (
    int nZone,
    int bNorth = TRUE )
```

Set UTM projection definition.

Universal Transverse Mercator

This will generate a projection definition with the full set of transverse mercator projection parameters for the given UTM zone. If no PROJCS[] description is set yet, one will be set to look like "UTM Zone %d, {Northern, Southern} Hemisphere".

This method is the same as the C function **OSRSetUTM()** (p. ??).



## Parameters

<i>nZone</i>	UTM zone.
<i>bNorth</i>	TRUE for northern hemisphere, or FALSE for southern hemisphere.

## Returns

OGRERR\_NONE on success.

< Failure  
< Success

References CPLError().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), and importFromUSGS().

## 11.96.3.161 SetVDG()

```
OGRERR OGRSpatialReference::SetVDG (
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

VanDerGrinten < Success

References OGRERR\_NONE, SetNormProjParm(), SetProjection(), SRS\_PP\_CENTRAL\_MERIDIAN, SRS\_PP\_FALSE\_EASTING, SRS\_PP\_FALSE\_NORTHING, and SRS\_PT\_VANDERGRINTEN.

Referenced by importFromPCI().

## 11.96.3.162 SetVertCS()

```
OGRERR OGRSpatialReference::SetVertCS (
    const char * pszVertCSName,
    const char * pszVertDatumName,
    int nVertDatumType = 2005 )
```

Set the user visible VERT\_CS name.

This method is the same as the C function **OSRSetVertCS()** (p. ??).

This method will ensure a VERT\_CS node is created if needed. If the existing coordinate system is GEOGCS or PROJCS rooted, then it will be turned into a COMPD\_CS.

## Parameters

<i>pszVertCSName</i>	the user visible name of the vertical coordinate system. Not used as a key.
<i>pszVertDatumName</i>	the user visible name of the vertical datum. It is helpful if this matches the EPSG name.
<i>nVertDatumType</i>	the OGC vertical datum type, usually 2005.

**Returns**

OGRERR\_NONE on success.

**Since**

OGR 1.9.0

< Success

References OGR\_SRSNode::AddChild(), Clear(), OGR\_SRSNode::ClearChildren(), EQUAL, GetAttrNode(), OGR\_SRSNode::GetValue(), IsGeographic(), IsProjected(), OGRERR\_NONE, CPLString::Printf(), and SetRoot().

**11.96.3.163 SetWagner()**

```
OGRERR OGRSpatialReference::SetWagner (
    int nVariation,
    double dfCenterLat,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Wagner I – VII < Unsupported SRS

< Success

References CPLError(), SetNormProjParm(), SetProjection(), SRS\_PP\_LATITUDE\_OF\_ORIGIN, SRS\_PT\_WAGNER\_I, SRS\_PT\_WAGNER\_II, SRS\_PT\_WAGNER\_III, SRS\_PT\_WAGNER\_IV, SRS\_PT\_WAGNER\_V, SRS\_PT\_WAGNER\_VI, and SRS\_PT\_WAGNER\_VII.

Referenced by importFromPanorama().

**11.96.3.164 SetWellKnownGeogCS()**

```
OGRERR OGRSpatialReference::SetWellKnownGeogCS (
    const char * pszName )
```

Set a GeogCS based on well known name.

This may be called on an empty **OGRSpatialReference** (p. ??) to make a geographic coordinate system, or on something with an existing PROJCS node to set the underlying geographic coordinate system of a projected coordinate system.

The following well known text values are currently supported:

- "WGS84": same as "EPSG:4326" but has no dependence on EPSG data files.
- "WGS72": same as "EPSG:4322" but has no dependence on EPSG data files.
- "NAD27": same as "EPSG:4267" but has no dependence on EPSG data files.
- "NAD83": same as "EPSG:4269" but has no dependence on EPSG data files.
- "EPSG:n": same as doing an ImportFromEPSG(n).

## Parameters

<i>pszName</i>	name of well known geographic coordinate system.
----------------	--

## Returns

OGRERR\_NONE on success, or OGRERR\_FAILURE if the name isn't recognised, the target object is already initialized, or an EPSG value can't be successfully looked up.

< Success

< Failure

< Success

< Failure

< Failure

< Success

References CopyGeogCSFrom(), EQUAL, importFromEPSG(), importFromEPSGA(), importFromWkt(), IsGeographic(), OGRERR\_FAILURE, OGRERR\_NONE, SRS\_WKT\_WGS84, and STARTS\_WITH\_CI.

Referenced by SetFromUserInput(), and OGRGeometryFactory::transformWithOptions().

## 11.96.3.165 StripCTParms()

```
OGRERR OGRSpatialReference::StripCTParms (
    OGR_SRSNode * poCurrent = nullptr )
```

Strip OGC CT Parameters.

This method will remove all components of the coordinate system that are specific to the OGC CT Specification. That is it will attempt to strip it down to being compatible with the Simple Features 1.0 specification.

This method is the same as the C function **OSRStripCTParms()** (p. ??).

## Parameters

<i>poCurrent</i>	node to operate on. NULL to operate on whole tree.
------------------	--

## Returns

OGRERR\_NONE on success or an error code.

< Success

< Success

< Success

References EQUAL, GetRoot(), OGR\_SRSNode::GetValue(), OGRERR\_NONE, OGR\_SRSNode::StripNodes(), and StripVertical().

Referenced by morphFromESRI(), and morphToESRI().

#### 11.96.3.166 StripVertical()

```
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

< Success

< Success

References OGR\_SRSNode::Clone(), EQUAL, OGR\_SRSNode::GetChild(), GetRoot(), OGRERR\_NONE, and SetRoot().

Referenced by StripCTParms().

#### 11.96.3.167 ToHandle()

```
static OGRSpatialReferenceH OGRSpatialReference::ToHandle (
    OGRSpatialReference * poSRS ) [inline], [static]
```

Convert a OGRSpatialReference\* to a OGRSpatialReferenceH.

Since

GDAL 2.3

Referenced by OGR\_G\_GetSpatialReference(), and OGR\_L\_GetSpatialRef().

## 11.96.3.168 Validate()

```
OGRErr OGRSpatialReference::Validate ( ) const
```

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.

< Corrupt data

< Success

< Corrupt data

References CPLDebug(), CPLFree, CPLGetConfigOption(), CPLTestBool(), exportToWkt(), OGRERR\_CORRUPT\_DATA, and OGRERR\_NONE.

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\_ozzi.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

## 11.97 OGRStyleMgr Class Reference

```
#include <ogr_featurestyle.h>
```

## Public Member Functions

- **OGRStyleMgr** ( **OGRStyleTable** \*poDataSetStyleTable=nullptr)  
*Constructor.*
- **~OGRStyleMgr** ()  
*Destructor.*
- **GBool SetFeatureStyleString** ( **OGRFeature** \*, const char \*pszStyleString=nullptr, **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=nullptr)  
*Initialize style manager from the style string.*
- const char \* **GetStyleName** (const char \*pszStyleString=nullptr)  
*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=nullptr)  
*Add a style to the current style table.*
- const char \* **GetStyleString** ( **OGRFeature** \*==nullptr)  
*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=nullptr)  
*Get the number of parts in a style.*
- **OGRStyleTool** \* **GetPart** (int hPartId, const char \*pszStyleString=nullptr)  
*Fetch a part (style tool) from the current style.*

### 11.97.1 Detailed Description

This class represents a style manager

### 11.97.2 Constructor & Destructor Documentation

#### 11.97.2.1 OGRStyleMgr()

```
OGRStyleMgr::OGRStyleMgr (
    OGRStyleTable * poDataSetStyleTable = nullptr ) [explicit]
```

Constructor.

This method is the same as the C function **OGR\_SM\_Create()** (p. ??)

## Parameters

<i>poDataSetStyleTable</i>	(currently unused, reserved for future use), pointer to <b>OGRStyleTable</b> (p. ??). Pass NULL for now.
----------------------------	--

## 11.97.2.2 ~OGRStyleMgr()

```
OGRStyleMgr::~OGRStyleMgr ( )
```

Destructor.

This method is the same as the C function **OGR\_SM\_Destroy()** (p. ??).

References CPLFree.

## 11.97.3 Member Function Documentation

## 11.97.3.1 AddPart() [1/2]

```
GBool OGRStyleMgr::AddPart (
    OGRStyleTool * poStyleTool )
```

Add a part (style tool) to the current style.

This method is the same as the C function **OGR\_SM\_AddPart()** (p. ??).

## Parameters

<i>poStyleTool</i>	the style tool defining the part to add.
--------------------	--

## Returns

TRUE on success, FALSE on errors.

References CPLFree, CPLStrdup(), and OGRStyleTool::GetStyleString().

## 11.97.3.2 AddPart() [2/2]

```
GBool OGRStyleMgr::AddPart (
    const char * pszPart )
```

Add a part (style string) to the current style.

## Parameters

<i>pszPart</i>	the style string defining the part to add.
----------------	--

## Returns

TRUE on success, FALSE on errors.

References CPLFree, and CPLStrdup().

## 11.97.3.3 AddStyle()

```
GBool OGRStyleMgr::AddStyle (
    const char * pszStyleName,
    const char * pszStyleString = nullptr )
```

Add a style to the current style table.

This method is the same as the C function **OGR\_SM\_AddStyle()** (p. ??).

## Parameters

<i>pszStyleName</i>	the name of the style to add.
<i>pszStyleString</i>	the style string to use, or NULL to use the style stored in the manager.

## Returns

TRUE on success, FALSE on errors.

References OGRStyleTable::AddStyle().

## 11.97.3.4 GetPart()

```
OGRStyleTool * OGRStyleMgr::GetPart (
    int nPartId,
    const char * pszStyleString = nullptr )
```

Fetch a part (style tool) from the current style.

This method is the same as the C function **OGR\_SM\_GetPart()** (p. ??).

This method instantiates a new object that should be freed with **OGR\_ST\_Destroy()** (p. ??).

## Parameters

<i>nPartId</i>	the part number (0-based index).
<i>pszStyleString</i>	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.



**Returns**

**OGRStyleTool** (p. ??) of the requested part (style tools) or NULL on error.

References CSLDestroy(), CSLGetField(), CSLT\_HONOURSTRINGS, CSLT\_PRESERVEESCAPES, CSLT\_PRESERVEQUOTES, CSLTokenizeString2(), and OGRStyleTool::SetStyleString().

**11.97.3.5 GetPartCount()**

```
int OGRStyleMgr::GetPartCount (
    const char * pszStyleString = nullptr )
```

Get the number of parts in a style.

This method is the same as the C function **OGR\_SM\_GetPartCount()** (p. ??).

**Parameters**

<i>pszStyleString</i>	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.
-----------------------	---

**Returns**

the number of parts (style tools) in the style.

**11.97.3.6 GetStyleByName()**

```
const char * OGRStyleMgr::GetStyleByName (
    const char * pszStyleName )
```

find a style in the current style table.

**Parameters**

<i>pszStyleName</i>	the name of the style to add.
---------------------	-------------------------------

**Returns**

the style string matching the name or NULL if not found or error.

References OGRStyleTable::Find().

Referenced by InitStyleString().

### 11.97.3.7 GetStyleName()

```
const char * OGRStyleMgr::GetStyleName (
    const char * pszStyleString = nullptr )
```

Get the name of a style from the style table.

#### Parameters

<i>pszStyleString</i>	the style to search for, or NULL to use the style currently stored in the manager.
-----------------------	--

#### Returns

The name if found, or NULL on error.

References OGRStyleTable::GetStyleName().

Referenced by SetFeatureStyleString().

### 11.97.3.8 GetStyleString()

```
const char * OGRStyleMgr::GetStyleString (
    OGRFeature * poFeature = nullptr )
```

Get the style string from the style manager.

#### Parameters

<i>poFeature</i>	feature object from which to read the style or NULL to get the style string stored in the manager.
------------------	--

#### Returns

the style string stored in the feature or the style string stored in the style manager if poFeature is NULL

NOTE: this method will call **OGRStyleMgr::InitFromFeature()** (p. ??) if poFeature is not NULL and replace the style string stored in the style manager

References InitFromFeature().

### 11.97.3.9 InitFromFeature()

```
const char * OGRStyleMgr::InitFromFeature (
    OGRFeature * poFeature )
```

Initialize style manager from the style string of a feature.

This method is the same as the C function **OGR\_SM\_InitFromFeature()** (p. ??).

## Parameters

<i>poFeature</i>	feature object from which to read the style.
------------------	--

## Returns

a reference to the style string read from the feature, or NULL in case of error..

References CPLFree, OGRFeature::GetStyleString(), and InitStyleString().

Referenced by GetStyleString().

## 11.97.3.10 InitStyleString()

```
GBool OGRStyleMgr::InitStyleString (
    const char * pszStyleString = nullptr )
```

Initialize style manager from the style string.

This method is the same as the C function **OGR\_SM\_InitStyleString()** (p. ??).

## Parameters

<i>pszStyleString</i>	the style string to use (can be NULL).
-----------------------	--

## Returns

TRUE on success, FALSE on errors.

References CPLFree, CPLStrdup(), and GetStyleByName().

Referenced by InitFromFeature().

## 11.97.3.11 SetFeatureStyleString()

```
GBool OGRStyleMgr::SetFeatureStyleString (
    OGRFeature * poFeature,
    const char * pszStyleString = nullptr,
    GBool bNoMatching = FALSE )
```

Set a style in a feature.

## Parameters

<i>poFeature</i>	the feature object to store the style in
<i>pszStyleString</i>	the style to store
<i>bNoMatching</i>	TRUE to lookup the style in the style table and add the name to the feature

Generated by Doxygen

**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**

## 11.98 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.*
- void **ResetStyleStringReading** ()
- const char \* **GetNextStyle** ()  
*Get the next style string from the table.*
- const char \* **GetLastStyleName** ()

### 11.98.1 Detailed Description

This class represents a style table

## 11.98.2 Member Function Documentation

### 11.98.2.1 AddStyle()

```
GBool OGRStyleTable::AddStyle (
    const char * pszName,
    const char * pszStyleString )
```

Add a new style in the table. No comparison will be done on the Style string, only on the name.

#### Parameters

<i>pszName</i>	the name the style to add.
<i>pszStyleString</i>	the style string to add.

#### Returns

TRUE on success, FALSE on error

References CSLAddString(), and IsExist().

Referenced by OGRStyleMgr::AddStyle(), and ModifyStyle().

### 11.98.2.2 Clone()

```
OGRStyleTable * OGRStyleTable::Clone ( )
```

Duplicate style table.

The newly created style table is owned by the caller, and will have its own reference to the **OGRStyleTable** (p. ??).

#### Returns

new style table, exactly matching this style table.

References CSLDuplicate().

Referenced by OGRLayer::SetStyleTable(), and OGRFeature::SetStyleTable().

### 11.98.2.3 Find()

```
const char * OGRStyleTable::Find (
    const char * pszName )
```

Get a style string by name.

**Parameters**

<i>pszName</i>	the name of the style string to find.
----------------	---------------------------------------

**Returns**

the style string matching the name, NULL if not found or error.

References CSLGetField(), and IsExist().

Referenced by OGRStyleMgr::GetStyleByName().

**11.98.2.4 GetLastStyleName()**

```
const char * OGRStyleTable::GetLastStyleName ( )
```

Get the style name of the last style string fetched with OGR\_STBL\_GetNextStyle.

**Returns**

the Name of the last style string or NULL on error.

**11.98.2.5 GetNextStyle()**

```
const char * OGRStyleTable::GetNextStyle ( )
```

Get the next style string from the table.

**Returns**

the next style string or NULL on error.

References CSLCount(), and CSLGetField().

**11.98.2.6 GetStyleName()**

```
const char * OGRStyleTable::GetStyleName (
    const char * pszStyleString )
```

Get style name by style string.

**Parameters**

<i>pszStyleString</i>	the style string to look up.
-----------------------	------------------------------

**Returns**

the Name of the matching style string or NULL on error.

References CSLCount(), and EQUAL.

Referenced by OGRStyleMgr::GetStyleName().

**11.98.2.7 IsExist()**

```
int OGRStyleTable::IsExist (
    const char * pszName )
```

Get the index of a style in the table by its name.

**Parameters**

<i>pszName</i>	the name to look for.
----------------	-----------------------

**Returns**

The index of the style if found, -1 if not found or error.

References CPLSPrintf(), and CSLCount().

Referenced by AddStyle(), Find(), and RemoveStyle().

**11.98.2.8 LoadStyleTable()**

```
GBool OGRStyleTable::LoadStyleTable (
    const char * pszFilename )
```

Load a style table from a file.

**Parameters**

<i>pszFilename</i>	the name of the file to load from.
--------------------	------------------------------------

**Returns**

TRUE on success, FALSE on error

References CSLDestroy(), and CSLLoad().

**11.98.2.9 ModifyStyle()**

```
GBool OGRStyleTable::ModifyStyle (
    const char * pszName,
    const char * pszStyleString )
```

Modify a style in the table by its name. If the style does not exist, it will be added.

**Parameters**

<i>pszName</i>	the name of the style to modify.
<i>pszStyleString</i>	the style string.

**Returns**

TRUE on success, FALSE on error

References AddStyle(), and RemoveStyle().

**11.98.2.10 Print()**

```
void OGRStyleTable::Print (
    FILE * fpOut )
```

Print a style table to a FILE pointer.

**Parameters**

<i>fpOut</i>	the FILE pointer to print to.
--------------	-------------------------------

**11.98.2.11 RemoveStyle()**

```
GBool OGRStyleTable::RemoveStyle (
    const char * pszName )
```

Remove a style in the table by its name.



**Parameters**

<i>pszName</i>	the name of the style to remove.
----------------	----------------------------------

**Returns**

TRUE on success, FALSE on error

References CSLRemoveStrings(), and IsExist().

Referenced by ModifyStyle().

**11.98.2.12 ResetStyleStringReading()**

```
void OGRStyleTable::ResetStyleStringReading ( )
```

Reset the next style pointer to 0

**11.98.2.13 SaveStyleTable()**

```
GBool OGRStyleTable::SaveStyleTable (
    const char * pszFilename )
```

Save a style table to a file.

**Parameters**

<i>pszFilename</i>	the name of the file to save to.
--------------------	----------------------------------

**Returns**

TRUE on success, FALSE on error

References CSLSave().

The documentation for this class was generated from the following files:

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

**11.99 OGRStyleTool Class Reference**

```
#include <ogr_featurestyle.h>
```

## Public Member Functions

- **OGRStyleTool** ( **OGRSTClassId** eClassId)
- **OGRSTClassId** **GetType** ()  
*Determine type of Style Tool.*
- void **SetUnit** ( **OGRSTUnitId**, double dfScale=1.0)  
*Set Style Tool units.*
- **OGRSTUnitId** **GetUnit** ()  
*Get Style Tool units.*
- virtual const char \* **GetStyleString** ()=0  
*Get the style string for this Style Tool.*
- void **SetStyleString** (const char \*pszStyleString)
- const char \* **GetStyleString** (const OGRStyleParamId \*pasStyleParam, OGRStyleValue \*pasStyleValue, int nSize)
- const char \* **GetParamStr** (const OGRStyleParamId &sStyleParam, OGRStyleValue &sStyleValue, **GBool** &bValuesIsNull)
- int **GetParamNum** (const OGRStyleParamId &sStyleParam, OGRStyleValue &sStyleValue, **GBool** &bValuesIsNull)
- double **GetParamDbf** (const OGRStyleParamId &sStyleParam, OGRStyleValue &sStyleValue, **GBool** &bValuesIsNull)
- void **SetParamStr** (const OGRStyleParamId &sStyleParam, OGRStyleValue &sStyleValue, const char \*pszParamString)
- void **SetParamNum** (const OGRStyleParamId &sStyleParam, OGRStyleValue &sStyleValue, int nParam)
- void **SetParamDbf** (const OGRStyleParamId &sStyleParam, OGRStyleValue &sStyleValue, double dfParam)

## Static Public Member Functions

- static **GBool** **GetRGBFromString** (const char \*pszColor, int &nRed, int &nGreen, int &nBlue, int &nTransparency)  
*Return the r,g,b,a components of a color encoded in #RRGGBB[AA] format.*
- static int **GetSpecifid** (const char \*pszId, const char \*pszWanted)

### 11.99.1 Detailed Description

This class represents a style tool

### 11.99.2 Constructor & Destructor Documentation

#### 11.99.2.1 OGRStyleTool()

```
OGRStyleTool::OGRStyleTool (
    OGRSTClassId eClassId ) [explicit]
```

Constructor

### 11.99.3 Member Function Documentation

#### 11.99.3.1 GetParamDbf()

```
double OGRStyleTool::GetParamDbf (
    const OGRStyleParamId & sStyleParam,
    OGRStyleValue & sStyleValue,
    GBool & bValueIsNull )
```

Undocumented

##### Parameters

<i>sStyleParam</i>	undocumented.
<i>sStyleValue</i>	undocumented.
<i>bValueIsNull</i>	undocumented.

##### Returns

Undocumented.

Referenced by GetParamNum().

#### 11.99.3.2 GetParamNum()

```
int OGRStyleTool::GetParamNum (
    const OGRStyleParamId & sStyleParam,
    OGRStyleValue & sStyleValue,
    GBool & bValueIsNull )
```

Undocumented

##### Parameters

<i>sStyleParam</i>	undocumented.
<i>sStyleValue</i>	undocumented.
<i>bValueIsNull</i>	undocumented.

##### Returns

Undocumented.

References GetParamDbf().

### 11.99.3.3 GetParamStr()

```
const char * OGRStyleTool::GetParamStr (
    const OGRStyleParamId & sStyleParam,
    OGRStyleValue & sStyleValue,
    GBool & bValueIsNull )
```

Undocumented

#### Parameters

<i>sStyleParam</i>	undocumented.
<i>sStyleValue</i>	undocumented.
<i>bValueIsNull</i>	undocumented.

#### Returns

Undocumented.

### 11.99.3.4 GetRGBFromString()

```
GBool OGRStyleTool::GetRGBFromString (
    const char * pszColor,
    int & nRed,
    int & nGreen,
    int & nBlue,
    int & nTransparance ) [static]
```

Return the r,g,b,a components of a color encoded in #RRGGBB[AA] format.

Maps to **OGRStyleTool::GetRGBFromString()** (p. ??).

#### Parameters

<i>pszColor</i>	the color to parse
<i>nRed</i>	reference to an int in which the red value will be returned.
<i>nGreen</i>	reference to an int in which the green value will be returned.
<i>nBlue</i>	reference to an int in which the blue value will be returned.
<i>nTransparance</i>	reference to an int in which the (optional) alpha value will be returned.

#### Returns

TRUE if the color could be successfully parsed, or FALSE in case of errors.

## 11.99.3.5 GetSpecificId()

```
int OGRStyleTool::GetSpecificId (
    const char * pszId,
    const char * pszWanted ) [static]
```

Undocumented

## Parameters

<i>pszId</i>	Undocumented
<i>pszWanted</i>	Undocumented

## Returns

Undocumented

## 11.99.3.6 GetStyleString() [1/2]

```
OGRStyleTool::GetStyleString ( ) [pure virtual]
```

Get the style string for this Style Tool.

Maps to the **OGRStyleTool** (p. ??) subclasses' **GetStyleString()** (p. ??) methods.

## Returns

the style string for this style tool or "" if the hST is invalid.

Referenced by OGRStyleMgr::AddPart().

## 11.99.3.7 GetStyleString() [2/2]

```
const char * OGRStyleTool::GetStyleString (
    const OGRStyleParamId * pasStyleParam,
    OGRStyleValue * pasStyleValue,
    int nSize )
```

Undocumented

## Parameters

<i>pasStyleParam</i>	undocumented.
<i>pasStyleValue</i>	undocumented.
<i>nSize</i>	undocumented.

**Returns**

undocumented.

References CPLFree, GetType(), OGRSTCBush, OGRSTCLabel, OGRSTCPen, and OGRSTCSymbol.

**11.99.3.8 GetType()**

```
OGRSTCClassId OGRStyleTool::GetType ( )
```

Determine type of Style Tool.

**Returns**

the style tool type, one of OGRSTCPen (1), OGRSTCBush (2), OGRSTCSymbol (3) or OGRSTCLabel (4). Returns OGRSTCNone (0) if the OGRStyleToolH is invalid.

Referenced by GetStyleString().

**11.99.3.9 GetUnit()**

```
OGRStyleTool::GetUnit ( ) [inline]
```

Get Style Tool units.

**Returns**

the style tool units.

Referenced by SetParamDbl(), SetParamNum(), and SetParamStr().

**11.99.3.10 SetParamDbl()**

```
void OGRStyleTool::SetParamDbl (
    const OGRStyleParamId & sStyleParam,
    OGRStyleValue & sStyleValue,
    double dfParam )
```

Undocumented

**Parameters**

<i>sStyleParam</i>	undocumented.
<i>sStyleValue</i>	undocumented.
<i>dfParam</i>	undocumented.

References `GetUnit()`.

### 11.99.3.11 SetParamNum()

```
void OGRStyleTool::SetParamNum (
    const OGRStyleParamId & sStyleParam,
    OGRStyleValue & sStyleValue,
    int nParam )
```

Undocumented

#### Parameters

<i>sStyleParam</i>	undocumented.
<i>sStyleValue</i>	undocumented.
<i>nParam</i>	undocumented.

References `GetUnit()`.

### 11.99.3.12 SetParamStr()

```
void OGRStyleTool::SetParamStr (
    const OGRStyleParamId & sStyleParam,
    OGRStyleValue & sStyleValue,
    const char * pszParamString )
```

Undocumented

#### Parameters

<i>sStyleParam</i>	undocumented.
<i>sStyleValue</i>	undocumented.
<i>pszParamString</i>	undocumented.

References `GetUnit()`.

### 11.99.3.13 SetStyleString()

```
void OGRStyleTool::SetStyleString (
    const char * pszStyleString )
```

Undocumented

## Parameters

<i>pszStyleString</i>	undocumented.
-----------------------	---------------

References CPLStrdup().

Referenced by OGRStyleMgr::GetPart().

## 11.99.3.14 SetUnit()

```
void OGRStyleTool::SetUnit (
    OGRSTUnitId eUnit,
    double dfGroundPaperScale = 1.0 )
```

Set Style Tool units.

## Parameters

<i>eUnit</i>	the new unit.
<i>dfGroundPaperScale</i>	ground to paper scale factor.

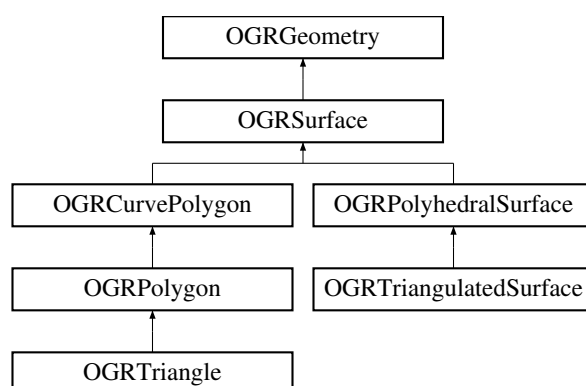
The documentation for this class was generated from the following files:

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

## 11.100 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  
*This method relates to the SFCOM ISurface::get\_PointOnSurface() method.*

## Additional Inherited Members

### 11.100.1 Detailed Description

Abstract base class for 2 dimensional objects like polygons or curve polygons.

### 11.100.2 Member Function Documentation

#### 11.100.2.1 get\_Area()

```
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 **OGRPolyhedralSurface** (p. ??), and **OGRCurvePolygon** (p. ??).

Referenced by OGRGeometryCollection::get\_Area().

#### 11.100.2.2 PointOnSurface()

```
OGRErr OGRSurface::PointOnSurface (
    OGRPoint * poPoint ) const [inline], [virtual]
```

This method relates to the SFCOM ISurface::get\_PointOnSurface() method.

NOTE: Only implemented when GEOS included in build.

## Parameters

<i>poPoint</i>	point to be set with an internal point.
----------------	---

## Returns

OGRERR\_NONE if it succeeds or OGRERR\_FAILURE otherwise.

Reimplemented in **OGRPolyhedralSurface** (p. ??).

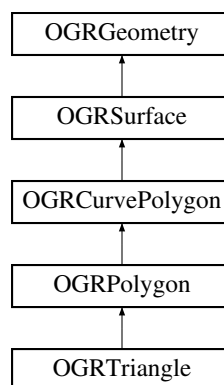
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrsurface.cpp

## 11.101 OGRTriangle Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRTriangle:



### Public Member Functions

- **OGRTriangle** ()  
*Constructor.*
- **OGRTriangle** (const **OGRPoint** &p, const **OGRPoint** &q, const **OGRPoint** &r)  
*Construct a triangle from points.*
- **OGRTriangle** (const **OGRTriangle** &other)  
*Copy constructor.*
- **OGRTriangle** (const **OGRPolygon** &other, **OGRERR** &eErr)  
*Constructs an **OGRTriangle** (p. ??) from a valid **OGRPolygon** (p. ??). In case of error, NULL is returned.*
- **OGRTriangle** & **operator=** (const **OGRTriangle** &other)  
*Assignment operator.*
- **~OGRTriangle** () override  
*Destructor.*
- virtual const char \* **getGeometryName** () const override

- Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override
- Fetch geometry type.*
- virtual **OGRErr** **importFromWkb** (const unsigned char \*, int, **OGRwkbVariant**, int &nBytesConsumedOut) override
- Assign geometry from well known binary data.*
- virtual **OGRErr** **addRingDirectly** ( **OGRCurve** \*poNewRing) override
- Add a ring to a polygon.*
- **OGRPolygon** \* **toUpperClass** ()
- const **OGRPolygon** \* **toUpperClass** () const
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override

## Additional Inherited Members

### 11.101.1 Detailed Description

Triangle class.

Since

GDAL 2.2

### 11.101.2 Constructor & Destructor Documentation

#### 11.101.2.1 OGRTriangle() [1/2]

```
OGRTriangle::OGRTriangle (
    const OGRPoint & p,
    const OGRPoint & q,
    const OGRPoint & r )
```

Construct a triangle from points.

Parameters

<i>p</i>	Point 1
<i>q</i>	Point 2
<i>r</i>	Point 3

References [OGRSimpleCurve::addPoint\(\)](#).

### 11.101.2.2 OGRTriangle() [2/2]

```
OGRTriangle::OGRTriangle (
    const OGRPolygon & other,
    OGRERR & eErr )
```

Constructs an **OGRTriangle** (p. ??) from a valid **OGRPolygon** (p. ??). In case of error, NULL is returned.

#### Parameters

<i>other</i>	the Polygon we wish to construct a triangle from
<i>eErr</i>	encapsulates an error code; contains OGRERR_NONE if the triangle is constructed successfully

< Success

References **OGRCurvePolygon::addRing()**, **CPLError()**, **OGRCurve::get\_IsClosed()**, **OGRCurvePolygon::getExteriorRingCurve()**, **OGRCurvePolygon::getNumInteriorRings()**, **OGRCurve::getNumPoints()**, and **OGRERR\_NONE**.

## 11.101.3 Member Function Documentation

### 11.101.3.1 accept() [1/2]

```
virtual void OGRTriangle::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

Reimplemented from **OGRPolygon** (p. ??).

References **IOGRGeometryVisitor::visit()**.

### 11.101.3.2 accept() [2/2]

```
virtual void OGRTriangle::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

Reimplemented from **OGRPolygon** (p. ??).

References **IOGRConstGeometryVisitor::visit()**.

### 11.101.3.3 addRingDirectly()

```
OGRERR OGRTriangle::addRingDirectly (
    OGRCurve * poNewRing ) [override], [virtual]
```

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

<i>poNewRing</i>	ring to be added to the polygon.
------------------	----------------------------------

## Returns

OGRERR\_NONE in case of success

< Failure

Reimplemented from **OGRCurvePolygon** (p. ??).

References OGRERR\_FAILURE.

## 11.101.3.4 getGeometryName()

```
const char * OGRTriangle::getGeometryName ( ) const [override], [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 **OGRPolygon** (p. ??).

## 11.101.3.5 getGeometryType()

```
OGRwkbGeometryType OGRTriangle::getGeometryType ( ) const [override], [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 **OGRPolygon** (p. ??).

References wkbTriangle, wkbTriangleM, wkbTriangleZ, and wkbTriangleZM.

## 11.101.3.6 importFromWkb()

```

OGRERR OGRTriangle::importFromWkb (
    const unsigned char * pabyData,
    int nSize,
    OGRwkbVariant eWkbVariant,
    int & nBytesConsumedOut ) [override], [virtual]

```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.
<i>eWkbVariant</i>	if wkbVariantPostGIS1, special interpretation is done for curve geometries code
<i>nBytesConsumedOut</i>	output parameter. Number of bytes consumed.

## 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.

## Since

GDAL 2.3

< Success

< Corrupt data

< Success

< Success

< Success

< Success

< Success

< Success

Reimplemented from **OGRPolygon** (p. ??).

References CPLDebug(), OGRCurvePolygon::empty(), OGRPolygon::importFromWkb(), OGRERR\_CORRUPT\_DATA, and OGRERR\_NONE.

## 11.101.3.7 operator=()

```

OGRTriangle & OGRTriangle::operator= (
    const OGRTriangle & other )

```

Assignment operator.

## Parameters

<i>other</i>	A triangle passed as a parameter
--------------	----------------------------------

## Returns

**OGRTriangle** (p. ??) A copy of other

References OGRPolygon::operator=().

## 11.101.3.8 toUpperClass() [1/2]

```
OGRPolygon* OGRTriangle::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

## 11.101.3.9 toUpperClass() [2/2]

```
const OGRPolygon* OGRTriangle::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

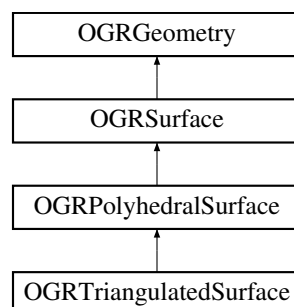
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrtriangle.cpp

## 11.102 OGRTriangulatedSurface Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRTriangulatedSurface:



## Public Types

- typedef **OGRTriangle** **ChildType**

## Public Member Functions

- **OGRTriangulatedSurface** ()  
*Constructor.*
- **OGRTriangulatedSurface** (const **OGRTriangulatedSurface** &other)  
*Copy constructor.*
- ~**OGRTriangulatedSurface** ()  
*Destructor.*
- **ChildType** \*\* **begin** ()
- **ChildType** \*\* **end** ()
- const **ChildType** \*const \* **begin** () const
- const **ChildType** \*const \* **end** () const
- **OGRTriangulatedSurface** & **operator=** (const **OGRTriangulatedSurface** &other)  
*Assignment operator.*
- virtual const char \* **getGeometryName** () const override  
*Returns the geometry name of the TriangulatedSurface.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const override  
*Returns the WKB Type of TriangulatedSurface.*
- virtual **OGRErr** **addGeometry** (const **OGRGeometry** \*) override  
*Add a new geometry to a collection.*
- **OGRPolyhedralSurface** \* **toUpperClass** ()
- const **OGRPolyhedralSurface** \* **toUpperClass** () const
- virtual void **accept** ( **IOGRGeometryVisitor** \*visitor) override
- virtual void **accept** ( **IOGRConstGeometryVisitor** \*visitor) const override

## Static Public Member Functions

- static **OGRPolyhedralSurface** \* **CastToPolyhedralSurface** ( **OGRTriangulatedSurface** \*poTS)  
*Casts the **OGRTriangulatedSurface** (p. ??) to an **OGRPolyhedralSurface** (p. ??).*

### 11.102.1 Detailed Description

TriangulatedSurface class.

Since

GDAL 2.2

### 11.102.2 Member Typedef Documentation



### 11.102.2.1 ChildType

```
typedef OGRTriangle OGRTriangulatedSurface::ChildType
```

Type of child elements.

## 11.102.3 Member Function Documentation

### 11.102.3.1 accept() [1/2]

```
virtual void OGRTriangulatedSurface::accept (
    IOGRGeometryVisitor * visitor ) [inline], [override], [virtual]
```

Accept a visitor.

Reimplemented from **OGRPolyhedralSurface** (p. ??).

References IOGRGeometryVisitor::visit().

### 11.102.3.2 accept() [2/2]

```
virtual void OGRTriangulatedSurface::accept (
    IOGRConstGeometryVisitor * visitor ) const [inline], [override], [virtual]
```

Accept a visitor.

Reimplemented from **OGRPolyhedralSurface** (p. ??).

References IOGRConstGeometryVisitor::visit().

### 11.102.3.3 addGeometry()

```
OGRErr OGRTriangulatedSurface::addGeometry (
    const OGRGeometry * poNewGeom ) [override], [virtual]
```

Add a new geometry to a collection.

Only a POLYGON can be added to a POLYHEDRALSURFACE.

#### Returns

OGRErr OGRERR\_NONE if the polygon is successfully added

< Failure

< Success

< Success

< Unsupported geometry type

< Unsupported geometry type

< Failure

< Success

Reimplemented from **OGRPolyhedralSurface** (p. ??).

References OGRPolyhedralSurface::addGeometry(), OGRPolyhedralSurface::addGeometryDirectly(), EQUAL, OGRGeometry::getGeometryName(), OGRERR\_FAILURE, OGRERR\_NONE, OGRERR\_UNSUPPORTED\_G↵  
EOMETRY\_TYPE, and OGRGeometry::toPolygon().

Referenced by operator=().

### 11.102.3.4 begin() [1/2]

```
ChildType** OGRTriangulatedSurface::begin ( ) [inline]
```

Return begin of iterator.

#### Since

GDAL 2.3

## 11.102.3.5 begin() [2/2]

```
const ChildType* const* OGRTriangulatedSurface::begin ( ) const [inline]
```

Return begin of iterator.

Since

GDAL 2.3

## 11.102.3.6 CastToPolyhedralSurface()

```
OGRPolyhedralSurface * OGRTriangulatedSurface::CastToPolyhedralSurface (
    OGRTriangulatedSurface * poTS ) [static]
```

Casts the **OGRTriangulatedSurface** (p. ??) to an **OGRPolyhedralSurface** (p. ??).

The passed in geometry is consumed and a new one returned (or NULL in case of failure)

Parameters

<i>poTS</i>	the input geometry - ownership is passed to the method.
-------------	---

Returns

new geometry.

References [OGRPolyhedralSurface::addGeometryDirectly\(\)](#), [OGRPolyhedralSurface::assignSpatialReference\(\)](#), [OGRGeometry::getSpatialReference\(\)](#), and [OGRPolyhedralSurface::OGRPolyhedralSurface\(\)](#).

Referenced by [OGRGeometryFactory::forceTo\(\)](#).

## 11.102.3.7 end() [1/2]

```
ChildType** OGRTriangulatedSurface::end ( ) [inline]
```

Return end of iterator

## 11.102.3.8 end() [2/2]

```
const ChildType* const* OGRTriangulatedSurface::end ( ) const [inline]
```

Return end of iterator

**11.102.3.9 getGeometryName()**

```
const char * OGRTriangulatedSurface::getGeometryName ( ) const [override], [virtual]
```

Returns the geometry name of the TriangulatedSurface.

**Returns**

"TIN"

Reimplemented from **OGRPolyhedralSurface** (p. ??).

**11.102.3.10 toUpperClass()** [1/2]

```
OGRPolyhedralSurface* OGRTriangulatedSurface::toUpperClass ( ) [inline]
```

Return pointer of this in upper class

Referenced by OGRDefaultGeometryVisitor::visit(), and OGRDefaultConstGeometryVisitor::visit().

**11.102.3.11 toUpperClass()** [2/2]

```
const OGRPolyhedralSurface* OGRTriangulatedSurface::toUpperClass ( ) const [inline]
```

Return pointer of this in upper class

The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrtriangulatedsurface.cpp

**11.103 osr\_cs\_wkt\_tokens Struct Reference**

The documentation for this struct was generated from the following file:

- osr\_cs\_wkt.c

**11.104 ParseContext Struct Reference**

The documentation for this struct was generated from the following file:

- cpl\_minixml.cpp

## 11.105 PCIDatums Struct Reference

The documentation for this struct was generated from the following file:

- ogr\_srs\_pci.cpp

## 11.106 OGRFeature::FieldValue::Private Struct Reference

The documentation for this struct was generated from the following file:

- ogrfeature.cpp

## 11.107 OGRLayer::FeatureIterator::Private Struct Reference

The documentation for this struct was generated from the following file:

- ogrlayer.cpp

## 11.108 OGRFeature::ConstFieldIterator::Private Struct Reference

The documentation for this struct was generated from the following file:

- ogrfeature.cpp

## 11.109 OGRCurve::ConstIterator::Private Struct Reference

The documentation for this struct was generated from the following file:

- ogrcurve.cpp

## 11.110 OGRSimpleCurve::Iterator::Private Struct Reference

The documentation for this struct was generated from the following file:

- ogrcurve.cpp

### 11.111 OGRSimpleCurve::ConstIterator::Private Struct Reference

The documentation for this struct was generated from the following file:

- ogrcurve.cpp

### 11.112 OGRLayer::Private Struct Reference

The documentation for this struct was generated from the following file:

- ogrlayer.cpp

### 11.113 SFRegion Class Reference

The documentation for this class was generated from the following file:

- cpl\_vsil\_sparsefile.cpp

### 11.114 StackContext Struct Reference

The documentation for this struct was generated from the following file:

- cpl\_minixml.cpp

### 11.115 TupleEnvVarOptionName Struct Reference

The documentation for this struct was generated from the following file:

- cpl\_http.cpp

### 11.116 unz\_file\_info\_internal\_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl\_minizip\_unzip.cpp

## 11.117 unz\_s Struct Reference

The documentation for this struct was generated from the following file:

- cpl\_minizip\_unzip.cpp

## 11.118 VSIDIR Struct Reference

The documentation for this struct was generated from the following file:

- vsipreload.cpp

## 11.119 VSSErrorContext Struct Reference

The documentation for this struct was generated from the following file:

- cpl\_vsi\_error.cpp

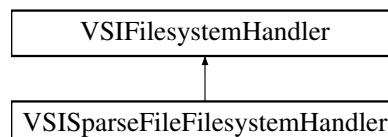
## 11.120 VSIReadDirRecursiveTask Struct Reference

The documentation for this struct was generated from the following file:

- cpl\_vsil.cpp

## 11.121 VSISparseFileFilesystemHandler Class Reference

Inheritance diagram for VSISparseFileFilesystemHandler:

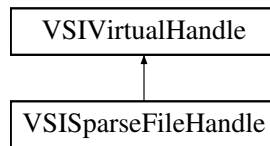


The documentation for this class was generated from the following file:

- cpl\_vsil\_sparsefile.cpp

## 11.122 VSISparseFileHandle Class Reference

Inheritance diagram for VSISparseFileHandle:



### Public Member Functions

- int **Seek** ( **vsi\_l\_offset** nOffset, int nWhence) override  
*Seek to requested offset.*
- **vsi\_l\_offset Tell** () override  
*Tell current file offset.*
- size\_t **Read** (void \*pBuffer, size\_t nSize, size\_t nMemb) override  
*Read bytes from file.*
- size\_t **Write** (const void \*pBuffer, size\_t nSize, size\_t nMemb) override  
*Write bytes to file.*
- int **Eof** () override  
*Test for end of file.*
- int **Close** () override  
*Close file.*

### 11.122.1 Member Function Documentation

#### 11.122.1.1 Close()

```
int VSISparseFileHandle::Close ( ) [override], [virtual]
```

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.

#### Returns

0 on success or -1 on failure.

Implements **VSIVirtualHandle** (p. ??).



## 11.122.1.2 Eof()

```
int VSISparseFileHandle::Eof ( ) [override], [virtual]
```

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful **VSIFSeekL()** (p. ??) call.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

## Returns

TRUE if at EOF else FALSE.

Implements **VSIVirtualHandle** (p. ??).

## 11.122.1.3 Read()

```
size_t VSISparseFileHandle::Read (
    void * pBuffer,
    size_t nSize,
    size_t nCount ) [override], [virtual]
```

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 VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

## Parameters

<i>pBuffer</i>	the buffer into which the data should be read (at least nCount * nSize bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.

## Returns

number of objects successfully read.

Implements **VSIVirtualHandle** (p. ??).

References CPLDebug(), VSIFOpenL(), VSIFReadL(), and VSIFSeekL().

#### 11.122.1.4 Seek()

```
int VSISparseFileHandle::Seek (
    vsi_l_offset nOffset,
    int nWhence ) [override], [virtual]
```

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.

Caution: vsi\_l\_offset is a unsigned type, so SEEK\_CUR can only be used for positive seek. If negative seek is needed, use handle->Seek( handle->Tell() (p. ??) + negative\_offset, SEEK\_SET ).

##### Parameters

<i>nOffset</i>	offset in bytes.
<i>nWhence</i>	one of SEEK_SET, SEEK_CUR or SEEK_END.

##### Returns

0 on success or -1 one failure.

Implements **VSIVirtualHandle** (p. ??).

#### 11.122.1.5 Tell()

```
vsi_l_offset VSISparseFileHandle::Tell ( ) [override], [virtual]
```

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.

##### Returns

file offset in bytes.

Implements **VSIVirtualHandle** (p. ??).

## 11.122.1.6 Write()

```
size_t VSISparseFileHandle::Write (
    const void * pBuffer,
    size_t nSize,
    size_t nCount ) [override], [virtual]
```

Write bytes to file.

Writes nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

## Parameters

<i>pBuffer</i>	the buffer from which the data should be written (at least nCount * nSize bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.

## Returns

number of objects successfully written.

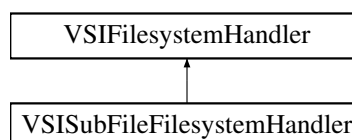
Implements **VSIVirtualHandle** (p. ??).

The documentation for this class was generated from the following file:

- cpl\_vsil\_sparsefile.cpp

## 11.123 VSISubFileFilesystemHandler Class Reference

Inheritance diagram for VSISubFileFilesystemHandler:

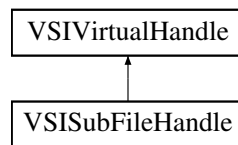


The documentation for this class was generated from the following file:

- cpl\_vsil\_subfile.cpp

## 11.124 VSISubFileHandle Class Reference

Inheritance diagram for VSISubFileHandle:



### Public Member Functions

- int **Seek** ( **vsi\_l\_offset** nOffset, int nWhence) override  
*Seek to requested offset.*
- **vsi\_l\_offset Tell** () override  
*Tell current file offset.*
- size\_t **Read** (void \*pBuffer, size\_t nSize, size\_t nMem) override  
*Read bytes from file.*
- size\_t **Write** (const void \*pBuffer, size\_t nSize, size\_t nMem) override  
*Write bytes to file.*
- int **Eof** () override  
*Test for end of file.*
- int **Close** () override  
*Close file.*

### 11.124.1 Member Function Documentation

#### 11.124.1.1 Close()

```
int VSISubFileHandle::Close ( ) [override], [virtual]
```

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.

#### Returns

0 on success or -1 on failure.

Implements **VSIVirtualHandle** (p. ??).

References VSIFCloseL().

## 11.124.1.2 Eof()

```
int VSISubFileHandle::Eof ( ) [override], [virtual]
```

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful **VSIFSeekL()** (p. ??) call.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

## Returns

TRUE if at EOF else FALSE.

Implements **VSIVirtualHandle** (p. ??).

## 11.124.1.3 Read()

```
size_t VSISubFileHandle::Read (
    void * pBuffer,
    size_t nSize,
    size_t nCount ) [override], [virtual]
```

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 VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

## Parameters

<i>pBuffer</i>	the buffer into which the data should be read (at least nCount * nSize bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.

## Returns

number of objects successfully read.

Implements **VSIVirtualHandle** (p. ??).

References VSIFReadL(), and VSIFTellL().

#### 11.124.1.4 Seek()

```
int VSISubFileHandle::Seek (
    vsi_l_offset nOffset,
    int nWhence ) [override], [virtual]
```

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.

Caution: vsi\_l\_offset is a unsigned type, so SEEK\_CUR can only be used for positive seek. If negative seek is needed, use handle->Seek( handle->Tell() (p. ??) + negative\_offset, SEEK\_SET ).

##### Parameters

<i>nOffset</i>	offset in bytes.
<i>nWhence</i>	one of SEEK_SET, SEEK_CUR or SEEK_END.

##### Returns

0 on success or -1 one failure.

Implements **VSIVirtualHandle** (p. ??).

References VSIFSeekL().

#### 11.124.1.5 Tell()

```
vsi_l_offset VSISubFileHandle::Tell ( ) [override], [virtual]
```

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.

##### Returns

file offset in bytes.

Implements **VSIVirtualHandle** (p. ??).

References VSIFTellL().

## 11.124.1.6 Write()

```
size_t VSISubFileHandle::Write (
    const void * pBuffer,
    size_t nSize,
    size_t nCount ) [override], [virtual]
```

Write bytes to file.

Writes nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

## Parameters

<i>pBuffer</i>	the buffer from which the data should be written (at least nCount * nSize bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.

## Returns

number of objects successfully written.

Implements **VSIVirtualHandle** (p. ??).

References VSIFTellL(), and VSIFWriteL().

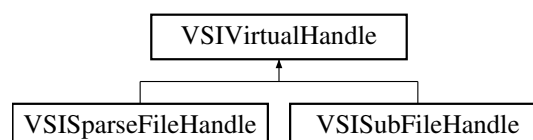
The documentation for this class was generated from the following file:

- cpl\_vsil\_subfile.cpp

## 11.125 VSIVirtualHandle Class Reference

```
#include <cpl_vsi_virtual.h>
```

Inheritance diagram for VSIVirtualHandle:



## Public Member Functions

- virtual int **Seek** ( **vsi\_I\_offset** nOffset, int nWhence)=0  
*Seek to requested offset.*
- virtual **vsi\_I\_offset** **Tell** ()=0  
*Tell current file offset.*
- virtual size\_t **Read** (void \*pBuffer, size\_t nSize, size\_t nCount)=0  
*Read bytes from file.*
- virtual int **ReadMultiRange** (int nRanges, void \*\*ppData, const **vsi\_I\_offset** \*panOffsets, const size\_t \*panSizes)  
*Read several ranges of bytes from file.*
- virtual size\_t **Write** (const void \*pBuffer, size\_t nSize, size\_t nCount)=0  
*Write bytes to file.*
- virtual int **Eof** ()=0  
*Test for end of file.*
- virtual int **Flush** ()  
*Flush pending writes to disk.*
- virtual int **Close** ()=0  
*Close file.*
- virtual int **Truncate** ( **vsi\_I\_offset** nNewSize)  
*Truncate/expand the file to the specified size.*
- virtual void \* **GetNativeFileDescriptor** ()  
*Returns the "native" file descriptor for the virtual handle.*
- virtual **VSIRangeStatus** **GetRangeStatus** ( **vsi\_I\_offset** nOffset, **vsi\_I\_offset** nLength)  
*Return if a given file range contains data or holes filled with zeroes.*

### 11.125.1 Detailed Description

Virtual file handle

### 11.125.2 Member Function Documentation

#### 11.125.2.1 Close()

```
VSIVirtualHandle::Close ( ) [pure virtual]
```

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.

#### Returns

0 on success or -1 on failure.

Implemented in **VSISparseFileHandle** (p. ??), and **VSISubFileHandle** (p. ??).

Referenced by VSIFCloseL().



### 11.125.2.2 Eof()

```
VSIVirtualHandle::Eof ( ) [pure virtual]
```

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful **VSIFSeekL()** (p. ??) call.

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

#### Returns

TRUE if at EOF else FALSE.

Implemented in **VSISparseFileHandle** (p. ??), and **VSISubFileHandle** (p. ??).

Referenced by VSIFEofL().

### 11.125.2.3 Flush()

```
VSIVirtualHandle::Flush ( ) [inline], [virtual]
```

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 VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

#### Returns

0 on success or -1 on error.

Referenced by VSIFFlushL().

#### 11.125.2.4 GetNativeFileDescriptor()

```
VSIVirtualHandle::GetNativeFileDescriptor ( ) [inline], [virtual]
```

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.

##### Returns

the native file descriptor, or NULL.

Referenced by VSIFGetNativeFileDescriptorL().

#### 11.125.2.5 GetRangeStatus()

```
VSIVirtualHandle::GetRangeStatus (
    vsi_l_offset nOffset,
    vsi_l_offset nLength ) [inline], [virtual]
```

Return if a given file range contains data or holes filled with zeroes.

This uses the filesystem capabilities of querying which regions of a sparse file are allocated or not. This is currently only implemented for Linux (and no other Unix derivatives) and Windows.

Note: A return of VSI\_RANGE\_STATUS\_DATA doesn't exclude that the extent is filled with zeroes! It must be interpreted as "may contain non-zero data".

##### Parameters

<i>nOffset</i>	offset of the start of the extent.
<i>nLength</i>	extent length.

##### Returns

extent status: VSI\_RANGE\_STATUS\_UNKNOWN, VSI\_RANGE\_STATUS\_DATA or VSI\_RANGE\_STATUS\_HOLE

##### Since

GDAL 2.2

References VSI\_RANGE\_STATUS\_UNKNOWN.

Referenced by VSIFGetRangeStatusL().

## 11.125.2.6 Read()

```
VSIVirtualHandle::Read (
    void * pBuffer,
    size_t nSize,
    size_t nCount ) [pure virtual]
```

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

<i>pBuffer</i>	the buffer into which the data should be read (at least <i>nCount</i> * <i>nSize</i> bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.

## Returns

number of objects successfully read.

Implemented in **VSISparseFileHandle** (p. ??), and **VSISubFileHandle** (p. ??).

Referenced by VSIFReadL().

## 11.125.2.7 ReadMultiRange()

```
VSIVirtualHandle::ReadMultiRange (
    int nRanges,
    void ** ppData,
    const vsi_l_offset * panOffsets,
    const size_t * panSizes ) [virtual]
```

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

<i>nRanges</i>	number of ranges to read.
<i>ppData</i>	array of <i>nRanges</i> buffer into which the data should be read ( <i>ppData</i> [ <i>i</i> ] must be at list <i>panSizes</i> [ <i>i</i> ] bytes).
<i>panOffsets</i>	array of <i>nRanges</i> offsets at which the data should be read.
<i>panSizes</i>	array of <i>nRanges</i> sizes of objects to read (in bytes).

**Returns**

0 in case of success, -1 otherwise.

**Since**

GDAL 1.9.0

Referenced by VSIFReadMultiRangeL().

**11.125.2.8 Seek()**

```
int VSIVirtualHandle::Seek (
    vsi_l_offset nOffset,
    int nWhence ) [pure virtual]
```

Seek to requested offset.

Seek to the desired offset (nOffset) in the indicated file.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Caution: vsi\_l\_offset is a unsigned type, so SEEK\_CUR can only be used for positive seek. If negative seek is needed, use handle->Seek( handle->Tell() (p. ??) + negative\_offset, SEEK\_SET ).

**Parameters**

<i>nOffset</i>	offset in bytes.
<i>nWhence</i>	one of SEEK_SET, SEEK_CUR or SEEK_END.

**Returns**

0 on success or -1 one failure.

Implemented in **VSISparseFileHandle** (p. ??), and **VSISubFileHandle** (p. ??).

Referenced by VSIFSeekL().

**11.125.2.9 Tell()**

```
VSIVirtualHandle::Tell ( ) [pure virtual]
```

Tell current file offset.

Returns the current file read/write offset in bytes from the beginning of the file.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftell() call.

**Returns**

file offset in bytes.

Implemented in **VSISparseFileHandle** (p. ??), and **VSISubFileHandle** (p. ??).

Referenced by VSIFTellL().

**11.125.2.10 Truncate()**

```
VSIVirtualHandle::Truncate (
    vsi_l_offset nNewSize ) [virtual]
```

Truncate/expand the file to the specified size.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftruncate() call.

**Parameters**

<i>nNewSize</i>	new size in bytes.
-----------------	--------------------

**Returns**

0 on success

**Since**

GDAL 1.9.0

Referenced by VSIFTruncateL().

**11.125.2.11 Write()**

```
VSIVirtualHandle::Write (
    const void * pBuffer,
    size_t nSize,
    size_t nCount ) [pure virtual]
```

Write bytes to file.

Writes nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

**Parameters**

<i>pBuffer</i>	the buffer from which the data should be written (at least nCount * nSize bytes in size.
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.

**Returns**

number of objects successfully written.

Implemented in **VSISparseFileHandle** (p. ??), and **VSISubFileHandle** (p. ??).

Referenced by VSIFWriteL().

The documentation for this class was generated from the following files:

- cpl\_vsi\_virtual.h
- cpl\_vsil.cpp

## 11.126 yyalloc Union Reference

The documentation for this union was generated from the following files:

- osr\_cs\_wkt\_parser.c
- swq\_parser.cpp

## 11.127 zip\_internal Struct Reference

The documentation for this struct was generated from the following file:

- cpl\_minizip\_zip.cpp

# Chapter 12

## File Documentation

### 12.1 cpl\_conv.h File Reference

```
#include "cpl_port.h"
#include "cpl_vsi.h"
#include "cpl_error.h"
```

#### Classes

- struct **CPLSharedFileInfo**

#### Macros

- #define **CPLFree VSIFree**

#### Typedefs

- typedef const char \*(\* **CPLFileFinder**) (const char \*, const char \*)

#### Functions

- const char \* **CPLGetConfigOption** (const char \*, const char \*)
- const char \* **CPLGetThreadLocalConfigOption** (const char \*, const char \*)
- void **CPLSetConfigOption** (const char \*, const char \*)
- void **CPLSetThreadLocalConfigOption** (const char \*pszKey, const char \*pszValue)
- char \*\* **CPLGetConfigOptions** (void)
- void **CPLSetConfigOptions** (const char \*const \*papszConfigOptions)
- char \*\* **CPLGetThreadLocalConfigOptions** (void)
- void **CPLSetThreadLocalConfigOptions** (const char \*const \*papszConfigOptions)
- 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, CSLConstList)
- const char \* **CPLReadLine3L** ( VSILFILE \*, int, int \*, CSLConstList)
- 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)
- const char \* **CPLExpandTilde** (const char \*pszFilename)
- const char \* **CPLGetHomeDir** (void)
- const char \* **CPLFindFile** (const char \*pszClass, const char \*pszBasename)
- const char \* **CPLDefaultFindFile** (const char \*pszClass, const char \*pszBasename)
- void **CPLPushFileFinder** ( **CPLFileFinder** pfnFinder)
- **CPLFileFinder** **CPLPopFileFinder** (void)



- void **CPLPushFinderLocation** (const char \*)
- void **CPLPopFinderLocation** (void)
- void **CPLFinderClean** (void)
- int **CPLStat** (const char \*, VSISBuf \*)
- FILE \* **CPLOpenShared** (const char \*, const char \*, int)
- void **CPLCloseShared** (FILE \*)
- **CPLSharedFileInfo** \* **CPLGetSharedList** (int \*)
- void **CPLDumpSharedList** (FILE \*)
- double **CPLDMSToDec** (const char \*is)
- const char \* **CPLDecToDMS** (double dfAngle, const char \*pszAxis, int nPrecision)
- double **CPLPackedDMSToDec** (double)
- double **CPLDecToPackedDMS** (double dfDec)
- void **CPLStringToComplex** (const char \*pszString, double \*pdfReal, double \*pdfImag)
- int **CPLUnlinkTree** (const char \*)
- int **CPLCopyFile** (const char \*pszNewPath, const char \*pszOldPath)
- int **CPLCopyTree** (const char \*pszNewPath, const char \*pszOldPath)
- int **CPLMoveFile** (const char \*pszNewPath, const char \*pszOldPath)
- int **CPLSymlink** (const char \*pszOldPath, const char \*pszNewPath, **CSLConstList** papszOptions)
- void \* **CPLCreateZip** (const char \*pszZipFilename, char \*\*papszOptions)
- **CPL**Err **CPLCreateFileInZip** (void \*hZip, const char \*pszFilename, char \*\*papszOptions)
- **CPL**Err **CPLWriteFileInZip** (void \*hZip, const void \*pBuffer, int nBufferSize)
- **CPL**Err **CPLCloseFileInZip** (void \*hZip)
- **CPL**Err **CPLCloseZip** (void \*hZip)
- void \* **CPLZLibInflate** (const void \*ptr, size\_t nBytes, void \*outptr, size\_t nOutAvailableBytes, size\_t \*pnOutBytes)
  - ↳ *Uncompress a buffer compressed with ZLib DEFLATE compression.*
- char \* **CPLsetlocale** (int category, const char \*locale)
- int **CPLIsPowerOfTwo** (unsigned int i)
- template<typename To, typename From >  
To **cpl::down\_cast** (From \*f)

### 12.1.1 Detailed Description

Various convenience functions for CPL.

### 12.1.2 Macro Definition Documentation

#### 12.1.2.1 CPLFree

```
#define CPLFree VSIFree
```

Alias of **VSIFree()** (p. ??)

Referenced by **OGRStyleMgr::AddPart()**, **CPLDBCStatement::AppendEscaped()**, **CPLDBCStatement::Clear()**, **OGR\_SRSNode::ClearChildren()**, **CPLDBCStatement::ClearColumnData()**, **CPLDestroyXMLNode()**,

CPLFormCIFilename(), CPLHashSetDestroy(), CPLHTTPDestroyMultiResult(), CPLHTTPDestroyResult(), CPLListDestroy(), CPLListRemove(), CPLParseXMLFile(), CPLPrintTime(), CPLQuadTreeDestroy(), CPLScanDouble(), CPLSetXMLValue(), CPLStrtodDelim(), CPLStrtofDelim(), CPLUnescapeString(), CPLVirtualMemFree(), CSLAddNameValue(), CSLMerge(), CSLRemoveStrings(), CSLSetNameValue(), CSLSetNameValueSeparator(), CSLTokenizeString2(), OGRSpatialReference::dumpReadable(), OGRGeometry::dumpReadable(), OGRGeometryCollection::empty(), OGRPolyhedralSurface::empty(), OGRPolygon::exportToWkt(), OGRFeature::FillUnsetWithDefault(), OGR\_SRSNode::FixupOrdering(), OGRLayer::GetFeature(), OGRFeature::GetFieldAsString(), OGRStyleMgr::GetStyleString(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::importFromProj4(), OGRPoint::importFromWkt(), OGRPolygon::importFromWkt(), OGRMultiPoint::importFromWkt(), OGRStyleMgr::InitFromFeature(), OGRStyleMgr::InitStyleString(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGR\_G\_ExportToGMLTree(), OGR\_G\_ExportToKML(), OGRGeocodeDestroySession(), OSRFreeSRSArray(), CPLString::Recode(), OGRFeatureDefn::ReorderFieldDefns(), CPLString::Seize(), OGRLayer::SetAttributeFilter(), OGRFieldDefn::SetDefault(), OGRFeature::SetField(), OGRFeature::SetFieldNull(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRFieldDefn::SetName(), OGRGeomFieldDefn::SetName(), OGRFeatureDefn::SetName(), CPLStringList::SetNameValue(), OGRFeature::SetNativeData(), OGRFeature::SetNativeMediaType(), OGRSimpleCurve::setNumPoints(), OGRFeature::SetStyleString(), OGRFeature::SetStyleStringDirectly(), OGR\_SRSNode::SetValue(), CPLWorkerThreadPool::SubmitJob(), CPLWorkerThreadPool::SubmitJobs(), OGRFeature::UnsetField(), OGRSpatialReference::Validate(), VSIRReadDirRecursive(), VSISetCryptKey(), and OGRStyleMgr::~~OGRStyleMgr().

### 12.1.3 Typedef Documentation

#### 12.1.3.1 CPLFileFinder

```
typedef const char*(* CPLFileFinder) (const char *, const char *)
```

Callback for CPLPushFileFinder

### 12.1.4 Function Documentation

#### 12.1.4.1 CPLAtof()

```
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

<i>nptr</i>	Pointer to string to convert.
-------------	-------------------------------

#### Returns

Converted value, if any.

References `CPLStrtod()`.

Referenced by `OGRSpatialReference::CloneGeogCS()`, `CPLGetValueType()`, `CPLScanDouble()`, `CPLStringToComplex()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::Fixup()`, `OGRSpatialReference::GetAngularUnits()`, `OGRFeature::GetFieldAsDouble()`, `OGRSpatialReference::GetInvFlattening()`, `OGRSpatialReference::GetPrimeMeridian()`, `OGRSpatialReference::GetProjParm()`, `OGRSpatialReference::GetSemiMajor()`, `OGRSpatialReference::GetTargetLinearUnits()`, `OGRSpatialReference::GetTOWGS84()`, `OGRSpatialReference::importFromERM()`, `OGRSpatialReference::importFromOzi()`, `OGRSpatialReference::importFromPCI()`, `OGRSpatialReference::importFromWMSAUTO()`, `OGRSpatialReference::IsSameGeogCS()`, `OGRSpatialReference::IsSameVertCS()`, and `OGRSpatialReference::SetGeogCS()`.

#### 12.1.4.2 CPLAtofDelim()

```
double CPLAtofDelim (
    const char * nptr,
    char point )
```

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by `nptr` to double floating point representation. The behaviour is the same as

`CPLStrtodDelim(nptr, (char **)NULL, point);`

This function does the same as standard `atof(3)`, but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. ??) function.

#### Parameters

<i>nptr</i>	Pointer to string to convert.
<i>point</i>	Decimal delimiter.

**Returns**

Converted value, if any.

References CPLStrtodDelim().

**12.1.4.3 CPLAtofM()**

```
double CPLAtofM (
    const char * nptr )
```

Converts ASCII string to floating point number using any numeric locale.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. This function does the same as standard `atof()`, but it allows a variety of locale representations. That is it supports numeric values with either a comma or a period for the decimal delimiter.

PS. The M stands for Multi-lingual.

**Parameters**

<i>nptr</i>	The string to convert.
-------------	------------------------

**Returns**

Converted value, if any. Zero on failure.

References CPLStrtodDelim().

Referenced by OGRSpatialReference::importFromOzi(), and OGRGeometryFactory::transformWithOptions().

**12.1.4.4 CPLAtoGIntBig()**

```
GIntBig CPLAtoGIntBig (
    const char * pszString )
```

Convert a string to a 64 bit signed integer.

**Parameters**

<i>pszString</i>	String containing 64 bit signed integer.
------------------	--

**Returns**

64 bit signed integer.

Since

GDAL 2.0

Referenced by VSInstallCurlFileHandler().

#### 12.1.4.5 CPLAtoGIntBigEx()

```
GIntBig CPLAtoGIntBigEx (
    const char * pszString,
    int bWarn,
    int * pbOverflow )
```

Convert a string to a 64 bit signed integer.

Parameters

<i>pszString</i>	String containing 64 bit signed integer.
<i>bWarn</i>	Issue a warning if an overflow occurs during conversion
<i>pbOverflow</i>	Pointer to an integer to store if an overflow occurred, or NULL

Returns

64 bit signed integer.

Since

GDAL 2.0

References CPLError().

Referenced by OGRFeature::GetFieldAsInteger64().

#### 12.1.4.6 CPLCalloc()

```
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()** (p. ??) to get the memory, so any hooking of **VSICalloc()** (p. ??) will apply to **CPLCalloc()** (p. ??) as well. **CPLFree()** (p. ??) or **VSIFree()** (p. ??) can be used free memory allocated by **CPLCalloc()** (p. ??).

**Parameters**

<i>nCount</i>	number of objects to allocate.
<i>nSize</i>	size (in bytes) of object to allocate.

**Returns**

pointer to newly allocated memory, only NULL if  $nSize * nCount$  is NULL.

References CPLMalloc().

Referenced by CPLODBCStatement::CollectResultsInfo(), CPLCreateXMLNode(), CPLHashSetNew(), CPLHTreeFetchEx(), CPLSerializeXMLTree(), CPLSPrintf(), CSLRemoveStrings(), CSLTokenizeString2(), OGR\_SRSNode::exportToPrettyWkt(), OGR\_SRSNode::exportToWkt(), OGRPolygon::exportToWkt(), OGR\_SRSNode::FixupOrdering(), CPLODBCStatement::GetColumns(), OGRGeocodeCreateSession(), and OPTGetParameterList().

**12.1.4.7 CPLCheckForFile()**

```
int CPLCheckForFile (
    char * pszFilename,
    char ** papszSiblingFiles )
```

Check for file existence.

The function checks if a named file exists in the filesystem, hopefully in an efficient fashion if a sibling file list is available. It exists primarily to do faster file checking for functions like GDAL open methods that get a list of files from the target directory.

If the sibling file list exists (is not NULL) it is assumed to be a list of files in the same directory as the target file, and it will be checked (case insensitively) for a match. If a match is found, pszFilename is updated with the correct case and TRUE is returned.

If papszSiblingFiles is NULL, a **VSIStatL()** (p. ??) is used to test for the files existence, and no case insensitive testing is done.

**Parameters**

<i>pszFilename</i>	name of file to check for - filename case updated in some cases.
<i>papszSiblingFiles</i>	a list of files in the same directory as pszFilename if available, or NULL. This list should have no path components.

**Returns**

TRUE if a match is found, or FALSE if not.

References CPLGetFilename(), EQUAL, and VSIStatL().

## 12.1.4.8 CPLCleanTrailingSlash()

```
const char* CPLCleanTrailingSlash (
    const char * pszPath )
```

Remove trailing forward/backward slash from the path for UNIX/Windows resp.

Returns a string containing the portion of the passed path string with trailing slash removed. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLCleanTrailingSlash( "abc/def/" ) == "abc/def"
CPLCleanTrailingSlash( "abc/def" ) == "abc/def"
CPLCleanTrailingSlash( "c:\abc\def\" ) == "c:\abc\def"
CPLCleanTrailingSlash( "c:\abc\def" ) == "c:\abc\def"
CPLCleanTrailingSlash( "abc" ) == "abc"
```

## Parameters

<i>pszPath</i>	the path to be cleaned up
----------------	---------------------------

## Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLAssert, and CPLStrncpy().

## 12.1.4.9 CPLCloseFileInZip()

```
CPLErr CPLCloseFileInZip (
    void * hZip )
```

Close current file inside ZIP file

## 12.1.4.10 CPLCloseShared()

```
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

<i>fp</i>	file handle from <b>CPLOpenShared()</b> (p. ??) to deaccess.
-----------	--

References `CPLError()`, and `CPLSharedFileInfo::fp`.

#### 12.1.4.11 `CPLCloseZip()`

```
CPLErr CPLCloseZip (
    void * hZip )
```

Close ZIP file

#### 12.1.4.12 `CPLCopyFile()`

```
int CPLCopyFile (
    const char * pszNewPath,
    const char * pszOldPath )
```

Copy a file

References `VSIFOpenL()`.

Referenced by `CPLMoveFile()`.

#### 12.1.4.13 `CPLCopyTree()`

```
int CPLCopyTree (
    const char * pszNewPath,
    const char * pszOldPath )
```

Recursively copy a tree

References `CPLError()`, and `VSISatL()`.

#### 12.1.4.14 `CPLCorrespondingPaths()`

```
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 correspondence 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

<i>pszOldFilename</i>	path to old prototype file.
<i>pszNewFilename</i>	path to new prototype file.
<i>papszFileList</i>	list of other files associated with pszOldFilename to rename similarly.

## Returns

a list of files corresponding to papszFileList but renamed to correspond to pszNewFilename.

References CPLError(), CPLGetBasename(), CPLGetFilename(), CPLGetPath(), CSLAddString(), CSLCount(), EQUAL, and EQUALN.

## 12.1.4.15 CPLCreateFileInZip()

```
CPLErr CPLCreateFileInZip (
    void * hZip,
    const char * pszFilename,
    char ** papszOptions )
```

Create a file in a ZIP file

## 12.1.4.16 CPLCreateZip()

```
void* CPLCreateZip (
    const char * pszZipFilename,
    char ** papszOptions )
```

Create ZIP file

References CPLError().

## 12.1.4.17 CPLDecToDMS()

```
const char* CPLDecToDMS (
    double dfAngle,
    const char * pszAxis,
    int nPrecision )
```

Translate a decimal degrees value to a DMS string with hemisphere.

References VALIDATE\_POINTER1.

## 12.1.4.18 CPLDecToPackedDMS()

```
double CPLDecToPackedDMS (
    double dfDec )
```

Convert decimal degrees into packed DMS value (DDDMMMSSS.SS).

This function converts a value, specified in decimal degrees into packed DMS angle. The standard packed DMS format is:

degrees \* 1000000 + minutes \* 1000 + seconds

See also **CPLPackedDMSToDec()** (p. ??).

## Parameters

<i>dfDec</i>	Angle in decimal degrees.
--------------	---------------------------

## Returns

Angle in packed DMS format.

Referenced by `OGRSpatialReference::exportToUSGS()`.

## 12.1.4.19 CPLDefaultFindFile()

```
const char* CPLDefaultFindFile (
    const char * ,
    const char * pszBasename )
```

## CPLDefaultFindFile

References `CPLFormFilename()`, `CSLCount()`, and `VSISatL()`.

## 12.1.4.20 CPLDMSToDec()

```
double CPLDMSToDec (
    const char * is )
```

## CPLDMSToDec

Referenced by `OGRSpatialReference::importFromProj4()`.

## 12.1.4.21 CPLDumpSharedList()

```
void CPLDumpSharedList (
    FILE * fp )
```

Report open shared files.

Dumps all open shared files to the indicated file handle. If the file handle is NULL information is sent via the **CPLDebug()** (p. ??) call.

## Parameters

<i>fp</i>	File handle to write to.
-----------	--------------------------

References CPLDebug().

#### 12.1.4.22 CPLEExpandTilde()

```
const char* CPLEExpandTilde (
    const char * pszFilename )
```

Expands ~/ at start of filename.

Assumes that the HOME configuration option is defined.

##### Parameters

<i>pszFilename</i>	filename potentially starting with ~/
--------------------	---------------------------------------

##### Returns

an expanded filename.

##### Since

GDAL 2.2

References CPLFormFilename(), CPLGetConfigOption(), and STARTS\_WITH\_CI.

#### 12.1.4.23 CPLExtractRelativePath()

```
const char* CPLExtractRelativePath (
    const char * pszBaseDir,
    const char * pszTarget,
    int * pbGotRelative )
```

Get relative path from directory to target file.

Computes a relative path for pszTarget relative to pszBaseDir. Currently this only works if they share a common base path. The returned path is normally into the pszTarget string. It should only be considered valid as long as pszTarget is valid or till the next call to this function, whichever comes first.

##### Parameters

<i>pszBaseDir</i>	the name of the directory relative to which the path should be computed. pszBaseDir may be NULL in which case the original target is returned without relativizing.
<i>pszTarget</i>	the filename to be changed to be relative to pszBaseDir.
<i>pbGotRelative</i>	Pointer to location in which a flag is placed indicating that the returned path is relative to the basename (TRUE) or not (FALSE). This pointer may be NULL if flag is not desired.

**Returns**

an adjusted path or the original if it could not be made relative to the pszBaseFile's path.

References CPLIsFilenameRelative(), EQUAL, and EQUALN.

**12.1.4.24 CPLFGets()**

```
char* CPLFGets (
    char * pszBuffer,
    int nBufferSize,
    FILE * fp )
```

Reads in at most one less than nBufferSize characters from the fp stream and stores them into the buffer pointed to by pszBuffer. Reading stops after an EOF or a newline. If a newline is read, it is *not* stored into the buffer. A '\0' is stored after the last character in the buffer. All three types of newline terminators recognized by the **CPLFGets()** (p. ??): single '\r' and '\n' and '\r\n' combination.

**Parameters**

<i>pszBuffer</i>	pointer to the targeting character buffer.
<i>nBufferSize</i>	maximum size of the string to read (not including terminating '\0').
<i>fp</i>	file pointer to read from.

**Returns**

pointer to the pszBuffer containing a string read from the file or NULL if the error or end of file was encountered.

References CPLError().

Referenced by CPLReadLine().

**12.1.4.25 CPLFinderClean()**

```
void CPLFinderClean (
    void )
```

CPLFinderClean

**12.1.4.26 CPLFindFile()**

```
const char* CPLFindFile (
    const char * pszClass,
    const char * pszBasename )
```

CPLFindFile

Referenced by OGRSpatialReference::importFromDict().

## 12.1.4.27 CPLFormCIFilename()

```
const char* CPLFormCIFilename (
    const char * pszPath,
    const char * pszBasename,
    const char * pszExtension )
```

Case insensitive file searching, returning full path.

This function tries to return the path to a file regardless of whether the file exactly matches the basename, and extension case, or is all upper case, or all lower case. The path is treated as case sensitive. This function is equivalent to **CPLFormFilename()** (p. ??) on case insensitive file systems (like Windows).

## Parameters

<i>pszPath</i>	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
<i>pszBasename</i>	file basename. May optionally have path and/or extension. May not be NULL.
<i>pszExtension</i>	file extension, optionally including the period. May be NULL.

## Returns

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References [CPLFormFilename\(\)](#), [CPLFree](#), [VSI\\_MALLOC\\_VERBOSE](#), [VSI\\_STAT\\_EXISTS\\_FLAG](#), [VSIIsCaseSensitiveFS\(\)](#), and [VSIStatExL\(\)](#).

## 12.1.4.28 CPLFormFilename()

```
const char* CPLFormFilename (
    const char * pszPath,
    const char * pszBasename,
    const char * pszExtension )
```

Build a full file path from a passed path, file basename and extension.

The path, and extension are optional. The basename may in fact contain an extension if desired.

```
CPLFormFilename("abc/xyz", "def", ".dat" ) == "abc/xyz/def.dat"
CPLFormFilename(NULL, "def", NULL ) == "def"
CPLFormFilename(NULL, "abc/def.dat", NULL ) == "abc/def.dat"
CPLFormFilename("/abc/xyz/", "def.dat", NULL ) == "/abc/xyz/def.dat"
```

## Parameters

<i>pszPath</i>	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
<i>pszBasename</i>	file basename. May optionally have path and/or extension. Must <i>NOT</i> be NULL.
<i>pszExtension</i>	file extension, optionally including the period. May be NULL.

**Returns**

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLAssert, CPLIsFilenameRelative(), CPLStrlcat(), and CPLStrlcpy().

Referenced by CPLDefaultFindFile(), CPLExpandTilde(), CPLFormCIFilename(), CPLGenerateTempFilename(), and VSIRmdirRecursive().

**12.1.4.29 CPLGenerateTempFilename()**

```
const char* CPLGenerateTempFilename (
    const char * pszStem )
```

Generate temporary file name.

Returns a filename that may be used for a temporary file. The location of the file tries to follow operating system semantics but may be forced via the CPL\_TMPDIR configuration option.

**Parameters**

<i>pszStem</i>	if non-NULL this will be part of the filename.
----------------	--

**Returns**

a filename which is valid till the next CPL call in this thread.

References CPLFormFilename(), CPLGetConfigOption(), and CPLString::Printf().

**12.1.4.30 CPLGetBasename()**

```
const char* CPLGetBasename (
    const char * pszFullFilename )
```

Extract basename (non-directory, non-extension) portion of filename.

Returns a string containing the file basename portion of the passed name. If there is no basename (passed value ends in trailing directory separator, or filename starts with a dot) an empty string is returned.

```
CPLGetBasename( "abc/def.xyz" ) == "def"
CPLGetBasename( "abc/def" ) == "def"
CPLGetBasename( "abc/def/" ) == ""
```

**Parameters**

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

**Returns**

just the non-directory, non-extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLAssert, and CPLStrncpy().

Referenced by CPLCorrespondingPaths().

**12.1.4.31 CPLGetConfigOption()**

```
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. ??), or **CPLSetThreadLocalConfigOption()** (p. ??) of the same thread. 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) (p. ??);
```

**Parameters**

<i>pszKey</i>	the key of the option to retrieve
<i>pszDefault</i>	a default value if the key does not match existing defined options (may be NULL)

**Returns**

the value associated to the key, or the default value if not found

**See also**

**CPLSetConfigOption()** (p. ??), <http://trac.osgeo.org/gdal/wiki/ConfigOptions>

References CSLFetchNameValue().

Referenced by CPLDebug(), CPLExpandTilde(), CPLGenerateTempFilename(), CPLGetHomeDir(), CPLHT↵  
 TPFetchEx(), CPLIsMachineForSureGCEInstance(), CPLIsMachinePotentiallyGCEInstance(), OGRGeometry↵  
 Factory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRFeatureDefn::GetGeomType(), G↵  
 OA2GetAccessTokenFromCloudEngineVM(), GOA2GetAuthorizationURL(), GOA2GetRefreshToken(), OGR↵  
 SpatialReference::morphFromESRI(), OGRGeometryFactory::organizePolygons(), OGRFeature::SetField(), OG↵  
 RSimpleCurve::transform(), OGRSpatialReference::Validate(), and VSIInstallCurlFileHandler().

#### 12.1.4.32 CPLGetConfigOptions()

```
char** CPLGetConfigOptions (
    void )
```

Return the list of configuration options as KEY=VALUE pairs.

The list is the one set through the **CPLSetConfigOption()** (p. ??) API.

Options that through environment variables or with **CPLSetThreadLocalConfigOption()** (p. ??) will *not* be listed.

##### Returns

a copy of the list, to be freed with **CSLDestroy()** (p. ??).

##### Since

GDAL 2.2

References CSLDuplicate().

#### 12.1.4.33 CPLGetCurrentDir()

```
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()** (p. ??) function. If HAVE\_GETCWD macro is not defined, the function returns NULL.

References VSI\_MALLOC\_VERBOSE.

#### 12.1.4.34 CPLGetDirname()

```
const char* CPLGetDirname (
    const char * pszFilename )
```

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename the dot will be returned. It is the only difference from **CPLGetPath()** (p. ??).

```
CPLGetDirname( "abc/def.xyz" ) == "abc"
CPLGetDirname( "/abc/def/" ) == "/abc/def"
CPLGetDirname( "/" ) == "/"
CPLGetDirname( "/abc/def" ) == "/abc"
CPLGetDirname( "abc" ) == "."
```



**Parameters**

<i>pszFilename</i>	the filename potentially including a path.
--------------------	--

**Returns**

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLAssert, and CPLStrlcpy().

**12.1.4.35 CPLGetExecPath()**

```
int CPLGetExecPath (
    char * pszPathBuf,
    int nMaxLength )
```

Fetch path of executable.

The path to the executable currently running is returned. This path includes the name of the executable. Currently this only works on win32 and linux platforms. The returned path is UTF-8 encoded.

**Parameters**

<i>pszPathBuf</i>	the buffer into which the path is placed.
<i>nMaxLength</i>	the buffer size, MAX_PATH+1 is suggested.

**Returns**

FALSE on failure or TRUE on success.

**12.1.4.36 CPLGetExtension()**

```
const char* CPLGetExtension (
    const char * pszFullFilename )
```

Extract filename extension from full filename.

Returns a string containing the extension portion of the passed name. If there is no extension (the filename has no dot) an empty string is returned. The returned extension will not include the period.

```
CPLGetExtension( "abc/def.xyz" ) == "xyz"
CPLGetExtension( "abc/def" ) == ""
```

**Parameters**

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

**Returns**

just the extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLAssert, and CPLStrncpy().

Referenced by OGRGeocodeCreateSession().

**12.1.4.37 CPLGetFilename()**

```
const char* CPLGetFilename (
    const char * pszFullFilename )
```

Extract non-directory portion of filename.

Returns a string containing the bare filename portion of the passed filename. If there is no filename (passed value ends in trailing directory separator) an empty string is returned.

```
CPLGetFilename( "abc/def.xyz" ) == "def.xyz"
CPLGetFilename( "/abc/def/" ) == ""
CPLGetFilename( "abc/def" ) == "def"
```

**Parameters**

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

**Returns**

just the non-directory portion of the path (points back into original string).

Referenced by CPLCheckForFile(), and CPLCorrespondingPaths().

**12.1.4.38 CPLGetHomeDir()**

```
const char* CPLGetHomeDir (
    void )
```

Return the path to the home directory

That is the value of the USERPROFILE environment variable on Windows, or HOME on other platforms.

**Returns**

the home directory, or NULL.

**Since**

GDAL 2.3

References CPLGetConfigOption().

**12.1.4.39 CPLGetPath()**

```
const char* CPLGetPath (
    const char * pszFilename )
```

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLGetPath( "abc/def.xyz" ) == "abc"
CPLGetPath( "/abc/def/" ) == "/abc/def"
CPLGetPath( "/" ) == "/"
CPLGetPath( "/abc/def" ) == "/abc"
CPLGetPath( "abc" ) == ""
```

**Parameters**

<i>pszFilename</i>	the filename potentially including a path.
--------------------	--

**Returns**

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLAssert, and CPLStrncpy().

Referenced by CPLCorrespondingPaths(), and VSIMkdirRecursive().

**12.1.4.40 CPLGetSharedList()**

```
CPLSharedFileInfo* CPLGetSharedList (
    int * pnCount )
```

Fetch list of open shared files.

**Parameters**

<i>pnCount</i>	place to put the count of entries.
----------------	------------------------------------

**Returns**

the pointer to the first in the array of shared file info structures.

**12.1.4.41 CPLGetSymbol()**

```
void* CPLGetSymbol (
    const char * pszLibrary,
    const char * pszSymbolName )
```

Fetch a function pointer from a shared library / DLL.

This function is meant to abstract access to shared libraries and DLLs and performs functions similar to `dlopen()/dlsym()` on Unix and `LoadLibrary()` / `GetProcAddress()` on Windows.

If no support for loading entry points from a shared library is available this function will always return NULL. Rules on when this function issues a **CPLError()** (p. ??) or not are not currently well defined, and will have to be resolved in the future.

Currently **CPLGetSymbol()** (p. ??) doesn't try to:

- prevent the reference count on the library from going up for every request, or given any opportunity to unload the library.
- Attempt to look for the library in non-standard locations.
- Attempt to try variations on the symbol name, like pre-pending or post-pending an underscore.

Some of these issues may be worked on in the future.

**Parameters**

<i>pszLibrary</i>	the name of the shared library or DLL containing the function. May contain path to file. If not system supplies search paths will be used.
<i>pszSymbolName</i>	the name of the function to fetch a pointer to.

**Returns**

A pointer to the function if found, or NULL if the function isn't found, or the shared library can't be loaded.

References `CPLError()`.

## 12.1.4.42 CPLGetThreadLocalConfigOption()

```
const char* CPLGetThreadLocalConfigOption (
    const char * pszKey,
    const char * pszDefault )
```

Same as **CPLGetConfigOption()** (p. ??) but only with options set with **CPLSetThreadLocalConfigOption()** (p. ??)

References CSLFetchNameValue().

## 12.1.4.43 CPLGetThreadLocalConfigOptions()

```
char** CPLGetThreadLocalConfigOptions (
    void )
```

Return the list of thread local configuration options as KEY=VALUE pairs.

Options that through environment variables or with **CPLSetConfigOption()** (p. ??) will *not* be listed.

## Returns

a copy of the list, to be freed with **CSLDestroy()** (p. ??).

## Since

GDAL 2.2

References CSLDuplicate().

## 12.1.4.44 CPLIsFilenameRelative()

```
int CPLIsFilenameRelative (
    const char * pszFilename )
```

Is filename relative or absolute?

The test is filesystem convention agnostic. That is it will test for Unix style and windows style path conventions regardless of the actual system in use.

## Parameters

<i>pszFilename</i>	the filename with path to test.
--------------------	---------------------------------

## Returns

TRUE if the filename is relative or FALSE if it is absolute.

References STARTS\_WITH.

Referenced by CPLExtractRelativePath(), CPLFormFilename(), and CPLProjectRelativeFilename().

#### 12.1.4.45 CPLIsPowerOfTwo()

```
int CPLIsPowerOfTwo (
    unsigned int i )
```

**CPLIsPowerOfTwo()** (p. ??)

##### Parameters

<i>i</i>	- tested number
----------	-----------------

##### Returns

TRUE if i is power of two otherwise return FALSE

#### 12.1.4.46 CPLMalloc()

```
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 **VSMalloc()** (p. ??) to get the memory, so any hooking of **VSMalloc()** (p. ??) will apply to **CPLMalloc()** (p. ??) as well. **CPLFree()** (p. ??) or **VSIFree()** (p. ??) can be used free memory allocated by **CPLMalloc()** (p. ??).

##### Parameters

<i>nSize</i>	size (in bytes) of memory block to allocate.
--------------	--

##### Returns

pointer to newly allocated memory, only NULL if nSize is zero.

References CPLError().

Referenced by CPLStringList::AddNameValue(), CPLBinaryToHex(), CPLCalloc(), CPLEscapeString(), CPL↵ ForceToASCII(), CPLHashSetNew(), CPLHexToBinary(), CPLListAppend(), CPLListInsert(), CPLParseName↵ Value(), CPLPrintTime(), CPLQuadTreeCreate(), CPLScanDouble(), CPLScanString(), CPLStrdup(), CPL↵ UnescapeString(), CSLAddNameValue(), CSLDuplicate(), CSLSetNameValue(), CSLSetNameValueSeparator(),

OGRSpatialReference::exportToPCI(), OGR\_SRSNode::exportToPrettyWkt(), OGRSpatialReference::exportToU↵  
 SGS(), OGR\_SRSNode::exportToWkt(), OGRSpatialReference::importFromPanorama(), OGRSpatialReference↵  
 ::importFromPCI(), CPLDBCDriverInstaller::InstallDriver(), OGR\_G\_ExportToGMLEx(), OGR\_G\_ExportToKML(),  
 OGRFeatureDefn::OGRFeatureDefn(), OGRFeatureDefn::ReorderFieldDefns(), OGRSpatialReference::SetFrom↵  
 UserInput(), CPLStringList::SetNameValue(), OGRProj4CT::Transform(), CPLString::vPrintf(), and VSISetCrypt↵  
 Key().

#### 12.1.4.47 CPLMoveFile()

```
int CPLMoveFile (
    const char * pszNewPath,
    const char * pszOldPath )
```

Move a file

References CPLCopyFile(), VSIRename(), and VSIUnlink().

#### 12.1.4.48 CPLOpenShared()

```
FILE* CPLOpenShared (
    const char * pszFilename,
    const char * pszAccess,
    int bLargeIn )
```

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

<i>pszFilename</i>	the name of the file to open.
<i>pszAccess</i>	the normal fopen()/VSIFOpen() style access string.
<i>bLargeIn</i>	If TRUE <b>VSIFOpenL()</b> (p. ??) (for large files) will be used instead of VSIFOpen().

##### Returns

a file handle or NULL if opening fails.

**12.1.4.49 CPLPackedDMSToDec()**

```
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: angle = 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 = angle/1000000 (fractional portion truncated)
3. The minutes are separated out: min = (angle - deg \* 1000000) / 1000 (fractional portion truncated)
4. The seconds are then computed: sec = angle - 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**

<i>dfPacked</i>	Angle in packed DMS format.
-----------------	-----------------------------

**Returns**

Angle in decimal degrees.

Referenced by OGRSpatialReference::importFromUSGS().

**12.1.4.50 CPLPopFileFinder()**

```
CPLFileFinder CPLPopFileFinder (
    void )
```

CPLPopFileFinder

**12.1.4.51 CPLPopFinderLocation()**

```
void CPLPopFinderLocation (
    void )
```

CPLPopFinderLocation



## 12.1.4.52 CPLPrintDouble()

```
int CPLPrintDouble (
    char * pszBuffer,
    const char * pszFormat,
    double dfValue,
    const char * pszLocale )
```

Print double value into specified string buffer. Exponential character flag 'E' (or 'e') will be replaced with 'D', as in Fortran. Resulting string will not to be NULL-terminated.

## Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>pszFormat</i>	Format specifier (for example, "%16.9E").
<i>dfValue</i>	Numerical value to print.
<i>pszLocale</i>	Unused.

## Returns

Number of characters printed.

References CPLPrintString(), and CPLSnprintf().

## 12.1.4.53 CPLPrintInt32()

```
int CPLPrintInt32 (
    char * pszBuffer,
    GInt32 iValue,
    int nMaxLen )
```

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

## Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>iValue</i>	Numerical value to print.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

## Returns

Number of characters printed.

References CPLPrintString().

Referenced by OGRSpatialReference::exportToPCI().

#### 12.1.4.54 CPLPrintPointer()

```
int CPLPrintPointer (
    char * pszBuffer,
    void * pValue,
    int nMaxLen )
```

Print pointer value into specified string buffer. This string will not be NULL-terminated.

##### Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>pValue</i>	Pointer to ASCII encode.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

##### Returns

Number of characters printed.

References CPLPrintString(), and STARTS\_WITH\_CI.

#### 12.1.4.55 CPLPrintString()

```
int CPLPrintString (
    char * pszDest,
    const char * pszSrc,
    int nMaxLen )
```

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest.

##### Parameters

<i>pszDest</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
<i>pszSrc</i>	Pointer to the source buffer.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

##### Returns

Number of characters printed.

Referenced by CPLPrintDouble(), CPLPrintInt32(), CPLPrintPointer(), CPLPrintTime(), and CPLPrintUIntBig().

## 12.1.4.56 CPLPrintStringFill()

```
int CPLPrintStringFill (
    char * pszDest,
    const char * pszSrc,
    int nMaxLen )
```

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest. Remainder of the destination string will be filled with space characters. This is only difference from the PrintString().

## Parameters

<i>pszDest</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
<i>pszSrc</i>	Pointer to the source buffer.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

## Returns

Number of characters printed.

Referenced by OGRSpatialReference::exportToPCI().

## 12.1.4.57 CPLPrintTime()

```
int CPLPrintTime (
    char * pszBuffer,
    int nMaxLen,
    const char * pszFormat,
    const struct tm * poBrokenTime,
    const char * pszLocale )
```

Print specified time value accordingly to the format options and specified locale name. This function does following:

- if locale parameter is not NULL, the current locale setting will be stored and replaced with the specified one;
- format time value with the strftime(3) function;
- restore back current locale, if was saved.

## Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.
<i>pszFormat</i>	Controls the output format. Options are the same as for strftime(3) function.
<i>poBrokenTime</i>	Pointer to the broken-down time structure. May be requested with the VSIGMTime() and VSILocalTime() functions.
<i>pszLocale</i>	Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. Be aware that it may be unsuitable to use current locale for printing time, because all names will be printed in your native language, as well as time format settings also may be adjusted differently from the C/POSIX defaults. To solve these problems this option was introduced.
Generated by Doxygen	

**Returns**

Number of characters printed.

References CPLFree, CPLMalloc(), CPLPrintString(), CPLsetlocale(), and EQUAL.

**12.1.4.58 CPLPrintUIntBig()**

```
int CPLPrintUIntBig (
    char * pszBuffer,
    GUIntBig iValue,
    int nMaxLen )
```

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

**Parameters**

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>iValue</i>	Numerical value to print.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

**Returns**

Number of characters printed.

References CPLPrintString().

**12.1.4.59 CPLProjectRelativeFilename()**

```
const char* CPLProjectRelativeFilename (
    const char * pszProjectDir,
    const char * pszSecondaryFilename )
```

Find a file relative to a project file.

Given the path to a "project" directory, and a path to a secondary file referenced from that project, build a path to the secondary file that the current application can use. If the secondary path is already absolute, rather than relative, then it will be returned unaltered.

**Examples:**

```
CPLProjectRelativeFilename("abc/def", "tmp/abc.gif") == "abc/def/tmp/abc.gif"
CPLProjectRelativeFilename("abc/def", "/tmp/abc.gif") == "/tmp/abc.gif"
CPLProjectRelativeFilename("/xy", "abc.gif") == "/xy/abc.gif"
CPLProjectRelativeFilename("/abc/def", "../abc.gif") == "/abc/def/../abc.gif"
CPLProjectRelativeFilename("C:\\WIN", "abc.gif") == "C:\\WIN\\abc.gif"
```

## Parameters

<i>pszProjectDir</i>	the directory relative to which the secondary files path should be interpreted.
<i>pszSecondaryFilename</i>	the filename (potentially with path) that is to be interpreted relative to the project directory.

## Returns

a composed path to the secondary file. The returned string is internal and should not be altered, freed, or depending on past the next CPL call.

References CPLAssert, CPLIsFilenameRelative(), CPLStrlcat(), and CPLStrlcpy().

## 12.1.4.60 CPLPushFileFinder()

```
void CPLPushFileFinder (
    CPLFileFinder pfnFinder )
```

CPLPushFileFinder

References CPLRealloc().

## 12.1.4.61 CPLPushFinderLocation()

```
void CPLPushFinderLocation (
    const char * pszLocation )
```

CPLPushFinderLocation

References CSLAddStringMayFail(), and CSLFindStringCaseSensitive().

## 12.1.4.62 CPLReadLine()

```
const char* CPLReadLine (
    FILE * fp )
```

Simplified line reading from text file.

Read a line of text from the given file handle, taking care to capture CR and/or LF and strip off ... equivalent of DKReadLine(). Pointer to an internal buffer is returned. The application shouldn't free it, or depend on its value past the next call to **CPLReadLine()** (p. ??).

Note that **CPLReadLine()** (p. ??) uses VSIFGets(), so any hooking of VSI file services should apply to **CPLReadLine()** (p. ??) as well.

**CPLReadLine()** (p. ??) maintains an internal buffer, which will appear as a single block memory leak in some circumstances. **CPLReadLine()** (p. ??) may be called with a NULL FILE \* at any time to free this working buffer.

## Parameters

<i>fp</i>	file pointer opened with VSIFOpen().
-----------	--------------------------------------

## Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

References CPLFgets().

## 12.1.4.63 CPLReadLine2L()

```
const char* CPLReadLine2L (
    VSILFILE * fp,
    int nMaxCars,
    CSLConstList papszOptions )
```

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. ??), but reading from a large file API handle.

## Parameters

<i>fp</i>	file pointer opened with <b>VSIFOpenL()</b> (p. ??).
<i>nMaxCars</i>	maximum number of characters allowed, or -1 for no limit.
<i>papszOptions</i>	NULL-terminated array of options. Unused for now.

## Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered or the maximum number of characters allowed reached.

## Since

GDAL 1.7.0

References CPLReadLine3L().

Referenced by CPLReadLineL().

## 12.1.4.64 CPLReadLine3L()

```
const char* CPLReadLine3L (
    VSILFILE * fp,
    int nMaxCars,
    int * pnBufLength,
    CSLConstList papszOptions )
```

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. ??), but reading from a large file API handle.

## Parameters

	<i>fp</i>	file pointer opened with <b>VSIFOpenL()</b> (p. ??).
	<i>nMaxCars</i>	maximum number of characters allowed, or -1 for no limit.
	<i>papszOptions</i>	NULL-terminated array of options. Unused for now.
out	<i>pnBufLength</i>	size of output string (must be non-NULL)

## Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered or the maximum number of characters allowed reached.

## Since

GDAL 2.3.0

References CPLError().

Referenced by CPLReadLine2L().

## 12.1.4.65 CPLReadLineL()

```
const char* CPLReadLineL (
    VSILFILE * fp )
```

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. ??), but reading from a large file API handle.

## Parameters

<i>fp</i>	file pointer opened with <b>VSIFOpenL()</b> (p. ??).
-----------	--

## Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

References CPLReadLine2L().

Referenced by CPLIsMachineForSureGCEInstance(), and OGRSpatialReference::importFromDict().

## 12.1.4.66 CPLRealloc()

```
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()`** (p. ??) to get the memory, so any hooking of **`VSIRealloc()`** (p. ??) will apply to **`CPLRealloc()`** (p. ??) as well. **`CPLFree()`** (p. ??) or **`VSIFree()`** (p. ??) 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()`** (p. ??) to allocate a new block.

#### Parameters

<i>pData</i>	existing memory block which should be copied to the new block.
<i>nNewSize</i>	new size (in bytes) of memory block to allocate.

#### Returns

pointer to allocated memory, only `NULL` if `nNewSize` is zero.

References `CPLError()`, and `VSIFree()`.

Referenced by `OGRFeatureDefn::AddFieldDefn()`, `OGRFeatureDefn::AddGeomFieldDefn()`, `CPLODBC↵Statement::Append()`, `CPLHashSetClear()`, `CPLPushFileFinder()`, `CSLInsertStrings()`, `CSLTokenizeString2()`, `O↵GRMultiPoint::exportToWkt()`, `OGRSpatialReference::importFromESRI()`, and `OGR_SRSNode::InsertChild()`.

#### 12.1.4.67 CPLResetExtension()

```
const char* CPLResetExtension (
    const char * pszPath,
    const char * pszExt )
```

Replace the extension with the provided one.

#### Parameters

<i>pszPath</i>	the input path, this string is not altered.
<i>pszExt</i>	the new extension to apply to the given path.

#### Returns

an altered filename with the new extension. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References `CPLAssert`, `CPLStrlcat()`, and `CPLStrlcpy()`.



#### 12.1.4.68 CPLScanDouble()

```
double CPLScanDouble (
    const char * pszString,
    int nMaxLength )
```

Extract double from string.

Scan up to a maximum number of characters from a string and convert the result to a double. This function uses **CPLAtof()** (p. ??) to convert string to double value, so it uses a comma as a decimal delimiter.

##### Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

##### Returns

Double value, converted from its ASCII form.

References CPLAtof(), CPLFree, and CPLMalloc().

#### 12.1.4.69 CPLScanLong()

```
long CPLScanLong (
    const char * pszString,
    int nMaxLength )
```

Scan up to a maximum number of characters from a string and convert the result to a long.

##### Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

##### Returns

Long value, converted from its ASCII form.

References CPLAssert, and CPLStrnlen().

Referenced by OGRSpatialReference::importFromPCI().

#### 12.1.4.70 CPLScanPointer()

```
void* CPLScanPointer (
    const char * pszString,
    int nMaxLength )
```

Extract pointer from string.

Scan up to a maximum number of characters from a string and convert the result to a pointer.

##### Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

##### Returns

pointer value, converted from its ASCII form.

References CPLScanUIntBig(), CPLScanULong(), and STARTS\_WITH\_CI.

#### 12.1.4.71 CPLScanString()

```
char* CPLScanString (
    const char * pszString,
    int nMaxLength,
    int bTrimSpaces,
    int bNormalize )
```

Scan up to a maximum number of characters from a given string, allocate a buffer for a new string and fill it with scanned characters.

##### Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to read. Less characters will be read if a null character is encountered.
<i>bTrimSpaces</i>	If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.
<i>bNormalize</i>	If TRUE, replace ':' symbol with the '_'. It is needed if resulting string will be used in CPL dictionaries.

##### Returns

Pointer to the resulting string buffer. Caller responsible to free this buffer with **CPLFree()** (p. ??).

References CPLMalloc(), and CPLStrdup().

## 12.1.4.72 CPLScanUIntBig()

```
GUIntBig CPLScanUIntBig (
    const char * pszString,
    int nMaxLength )
```

Extract big integer from string.

Scan up to a maximum number of characters from a string and convert the result to a GUIntBig.

## Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

## Returns

GUIntBig value, converted from its ASCII form.

References CPLAssert, and CPLStrnlen().

Referenced by CPLScanPointer().

## 12.1.4.73 CPLScanULong()

```
unsigned long CPLScanULong (
    const char * pszString,
    int nMaxLength )
```

Scan up to a maximum number of characters from a string and convert the result to a unsigned long.

## Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

## Returns

Unsigned long value, converted from its ASCII form.

References CPLAssert, and CPLStrnlen().

Referenced by CPLScanPointer().

#### 12.1.4.74 CPLSetConfigOption()

```
void CPLSetConfigOption (
    const char * pszKey,
    const char * pszValue )
```

Set a configuration option for GDAL/OGR use.

Those options are defined as a (key, value) couple. The value corresponding to a key can be got later with the **CPLGetConfigOption()** (p. ??) method.

This mechanism is similar to environment variables, but options set with **CPLSetConfigOption()** (p. ??) overrides, for **CPLGetConfigOption()** (p. ??) point of view, values defined in the environment.

If **CPLSetConfigOption()** (p. ??) is called several times with the same key, the value provided during the last call will be used.

Options can also be passed on the command line of most GDAL utilities with the with '-config KEY VALUE'. For example, ogrinfo -config CPL\_DEBUG ON ~/data/test/point.shp

This function can also be used to clear a setting by passing NULL as the value (note: passing NULL will not unset an existing environment variable; it will just unset a value previously set by **CPLSetConfigOption()** (p. ??)).

##### Parameters

<i>pszKey</i>	the key of the option
<i>pszValue</i>	the value of the option, or NULL to clear a setting.

##### See also

<http://trac.osgeo.org/gdal/wiki/ConfigOptions>

References CSLSetNameValue().

#### 12.1.4.75 CPLSetConfigOptions()

```
void CPLSetConfigOptions (
    const char *const * papszConfigOptions )
```

Replace the full list of configuration options with the passed list of KEY=VALUE pairs.

This has the same effect of clearing the existing list, and setting individually each pair with the **CPLSetConfigOption()** (p. ??) API.

This does not affect options set through environment variables or with **CPLSetThreadLocalConfigOption()** (p. ??).

The passed list is copied by the function.

##### Parameters

<i>papszConfigOptions</i>	the new list (or NULL).
---------------------------	-------------------------

Since

GDAL 2.2

References CSLDestroy(), and CSLDuplicate().

#### 12.1.4.76 CPLsetlocale()

```
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

<i>category</i>	See your compiler's documentation on setlocale.
<i>locale</i>	See your compiler's documentation on setlocale.

##### Returns

See your compiler's documentation on setlocale.

References CPLPrintf().

Referenced by CPLPrintTime().

#### 12.1.4.77 CPLSetThreadLocalConfigOption()

```
void CPLSetThreadLocalConfigOption (
    const char * pszKey,
    const char * pszValue )
```

Set a configuration option for GDAL/OGR use.

Those options are defined as a (key, value) couple. The value corresponding to a key can be got later with the **CPLGetConfigOption()** (p. ??) method.

This function sets the configuration option that only applies in the current thread, as opposed to **CPLSetConfigOption()** (p. ??) which sets an option that applies on all threads. **CPLSetThreadLocalConfigOption()** (p. ??) will override the effect of CPLSetConfigOption() for the current thread.

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 or a value set through **CPLSetConfigOption()** (p. ??); it will just unset a value previously set by **CPLSetThreadLocalConfigOption()** (p. ??)).

## Parameters

<i>pszKey</i>	the key of the option
<i>pszValue</i>	the value of the option, or NULL to clear a setting.

References CSLSetNameValue().

#### 12.1.4.78 CPLSetThreadLocalConfigOptions()

```
void CPLSetThreadLocalConfigOptions (
    const char *const * papszConfigOptions )
```

Replace the full list of thread local configuration options with the passed list of KEY=VALUE pairs.

This has the same effect of clearing the existing list, and setting individually each pair with the **CPLSetThreadLocalConfigOption()** (p. ??) API.

This does not affect options set through environment variables or with **CPLSetConfigOption()** (p. ??).

The passed list is copied by the function.

## Parameters

<i>papszConfigOptions</i>	the new list (or NULL).
---------------------------	-------------------------

Since

GDAL 2.2

References CSLDestroy(), and CSLDuplicate().

#### 12.1.4.79 CPLStat()

```
int CPLStat (
    const char * pszPath,
    VSISStatBuf * psStatBuf )
```

Same as VSISStat() except it works on "C:" as if it were "C:\".

#### 12.1.4.80 CPLStrdup()

```
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. Memory allocated with **CPLStrdup()** (p. ??) can be freed with **CPLFree()** (p. ??) or **VSIFree()** (p. ??).

It is also safe to pass a NULL string into **CPLStrdup()** (p. ??). **CPLStrdup()** (p. ??) will allocate and return a zero length string (as opposed to a NULL string).

## Parameters

<i>pszString</i>	input string to be duplicated. May be NULL.
------------------	---

## Returns

pointer to a newly allocated copy of the string. Free with **CPLFree()** (p. ??) or **VSIFree()** (p. ??).

References CPLMalloc().

Referenced by OGRStyleMgr::AddPart(), CPLStringList::AddString(), CPLODBCStatement::CollectResultsInfo(), CPLBase64Encode(), CPLCreateXMLNode(), CPLHTTPFetchEx(), CPLRecode(), CPLScanString(), CPLSetXMLValue(), CPLVASPrintf(), CSLDuplicate(), CSLInsertStrings(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToPrettyWkt(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::exportToWkt(), OGRPoint::exportToWkt(), OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), CPLODBCStatement::GetColumns(), OGRLayer::GetFeature(), OGRFeature::GetFieldAsSerializedJSON(), GOA2GetAccessToken(), GOA2GetAuthorizationURL(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::importFromProj4(), OGRStyleMgr::InitStyleString(), CPLStringList::InsertString(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGR\_G\_ExportToGMLEx(), OGR\_G\_ExportToJsonEx(), OGR\_G\_ExportToKML(), OGRFeatureDefn::OGRFeatureDefn(), OGRLayer::SetAttributeFilter(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRFieldDefn::SetName(), OGRGeomFieldDefn::SetName(), OGRFeatureDefn::SetName(), OGRStyleTool::SetStyleString(), OGR\_SRSNode::SetValue(), and VSIFReadDirRecursive().

## 12.1.4.81 CPLStringToComplex()

```
void CPLStringToComplex (
    const char * pszString,
    double * pdfReal,
    double * pdfImag )
```

Fetch the real and imaginary part of a serialized complex number

References CPLAtof().

## 12.1.4.82 CPLStrlwr()

```
char* CPLStrlwr (
    char * pszString )
```

Convert each characters of the string to lower case.

For example, "ABcdE" will be converted to "abcde". This function is locale dependent.

## Parameters

<i>pszString</i>	input string to be converted.
------------------	-------------------------------

**Returns**

pointer to the same string, pszString.

**12.1.4.83 CPLStrtod()**

```
double CPLStrtod (
    const char * nptr,
    char ** endptr )
```

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. This function does the same as standard `strtod(3)`, but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLStrtodDelim()** (p. ??) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. ??) function.

**Parameters**

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .

**Returns**

Converted value, if any.

References CPLStrtodDelim().

Referenced by CPLAtof(), and OGRFieldDefn::IsDefaultDriverSpecific().

**12.1.4.84 CPLStrtodDelim()**

```
double CPLStrtodDelim (
    const char * nptr,
    char ** endptr,
    char point )
```

Converts ASCII string to floating point number using specified delimiter.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. This function does the same as standard `strtod(3)`, but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. ??) function.

**Parameters**

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .
<i>point</i>	Decimal delimiter.



**Returns**

Converted value, if any.

References CPLFree, STARTS\_WITH, and STARTS\_WITH\_CI.

Referenced by CPLAtofDelim(), CPLAtofM(), CPLStrtod(), and CPLStrtofDelim().

**12.1.4.85 CPLStrtof()**

```
float CPLStrtof (
    const char * nptr,
    char ** endptr )
```

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by *nptr* to single floating point representation. This function does the same as standard `strtof(3)`, but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLStrtofDelim()** (p. ??) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. ??) function.

**Parameters**

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .

**Returns**

Converted value, if any.

References CPLStrtofDelim().

**12.1.4.86 CPLStrtofDelim()**

```
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

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.
<i>point</i>	Decimal delimiter.

## Returns

Converted value, if any.

References CPLFree, and CPLStrtodDelim().

Referenced by CPLStrtof().

## 12.1.4.87 CPLSymlink()

```
int CPLSymlink (
    const char * pszOldPath,
    const char * pszNewPath,
    CSLConstList )
```

Create a symbolic link

## 12.1.4.88 CPLUnlinkTree()

```
int CPLUnlinkTree (
    const char * pszPath )
```

Recursively unlink a directory.

## Returns

0 on successful completion, -1 if function fails.

References CPLError(), and VSISatL().

## 12.1.4.89 CPLWriteFileInZip()

```
CPLErr CPLWriteFileInZip (
    void * hZip,
    const void * pBuffer,
    int nBufferSize )
```

Write in current file inside a ZIP file

## 12.1.4.90 CPLZLibInflate()

```
void* CPLZLibInflate (
    const void * ptr,
    size_t nBytes,
    void * outptr,
    size_t nOutAvailableBytes,
    size_t * pnOutBytes )
```

Uncompress a buffer compressed with ZLib DEFLATE compression.

## Parameters

<i>ptr</i>	input buffer.
<i>nBytes</i>	size of input buffer in bytes.
<i>outptr</i>	output buffer, or NULL to let the function allocate it.
<i>nOutAvailableBytes</i>	size of output buffer if provided, or ignored.
<i>pnOutBytes</i>	pointer to a size_t, where to store the size of the output buffer.

## Returns

the output buffer (to be freed with **VSIFree()** (p. ??) if not provided) or NULL in case of error.

## Since

GDAL 1.10.0

## 12.1.4.91 down\_cast()

```
template<typename To , typename From >
To cpl::down_cast (
    From * f ) [inline]
```

Use `cpl::down_cast<Derived*>(pointer_to_base)` as equivalent of `static_cast<Derived*>(pointer_to_base)` with safe checking in debug mode.

Only works if no virtual inheritance is involved.

## Parameters

<i>f</i>	pointer to a base class
----------	-------------------------

## Returns

pointer to a derived class

References `CPLAssert`, and `cpl::down_cast()`.

Referenced by `cpl::down_cast()`.

## 12.2 cpl\_error.h File Reference

```
#include "cpl_port.h"
#include <stdarg.h>
#include <stddef.h>
```

## Macros

- `#define CPLE_None 0`
- `#define CPLE_AppDefined 1`
- `#define CPLE_OutOfMemory 2`
- `#define CPLE_FileIO 3`
- `#define CPLE_OpenFailed 4`
- `#define CPLE_IllegalArg 5`
- `#define CPLE_NotSupported 6`
- `#define CPLE_AssertionFailed 7`
- `#define CPLE_NoWriteAccess 8`
- `#define CPLE_UserInterrupt 9`
- `#define CPLE_ObjectNull 10`
- `#define CPLE_HttpResponse 11`
- `#define CPLE_AWSBucketNotFound 12`
- `#define CPLE_AWSObjectNotFound 13`
- `#define CPLE_AWSAccessDenied 14`
- `#define CPLE_AWSInvalidCredentials 15`
- `#define CPLE_AWSSignatureDoesNotMatch 16`
- `#define CPLAssert(expr)`
- `#define VALIDATE_POINTER0(ptr, func)`
- `#define VALIDATE_POINTER1(ptr, func, rc)`

## Typedefs

- `typedef int CPLErrorNum`
- `typedef void(* CPLErrorHandler) ( CPLErr, CPLErrorNum, const char *)`

## Enumerations

- `enum CPLErr`

## Functions

- `void CPLError ( CPLErr eErrClass, CPLErrorNum err_no, const char *fmt,...)`
- `void CPLErrorV ( CPLErr, CPLErrorNum, const char *, va_list)`
- `void CPLEmergencyError (const char *)`
- `void CPLErrorReset (void)`
- `CPLErrorNum CPLGetLastErrorNo (void)`
- `CPLErr CPLGetLastErrorType (void)`
- `const char * CPLGetLastErrorMsg (void)`
- `GUInt32 CPLGetErrorCounter (void)`
- `void * CPLGetErrorHandlerUserData (void)`
- `void CPLErrorSetState ( CPLErr eErrClass, CPLErrorNum err_no, const char *pszMsg)`
- `void CPLLoggingErrorHandler ( CPLErr, CPLErrorNum, const char *)`
- `void CPLDefaultErrorHandler ( CPLErr, CPLErrorNum, const char *)`
- `void CPLQuietErrorHandler ( CPLErr, CPLErrorNum, const char *)`
- `void CPLTurnFailureIntoWarning (int bOn)`
- `CPLErrorHandler CPLSetErrorHandler ( CPLErrorHandler)`
- `CPLErrorHandler CPLSetErrorHandlerEx ( CPLErrorHandler, void *)`
- `void CPLPushErrorHandler ( CPLErrorHandler)`
- `void CPLPushErrorHandlerEx ( CPLErrorHandler, void *)`
- `void CPLSetCurrentErrorHandlerCatchDebug (int bCatchDebug)`
- `void CPLPopErrorHandler (void)`
- `void CPLDebug (const char *, const char *,...)`
- `void _CPLAssert (const char *, const char *, int)`

### 12.2.1 Detailed Description

CPL error handling services.

### 12.2.2 Macro Definition Documentation

#### 12.2.2.1 CPLAssert

```
#define CPLAssert(  
    expr )
```

Assert on an expression. Only enabled in DEBUG mode

Referenced by CPLStringList::AddNameValue(), OGRSimpleCurve::addSubLineString(), CPLAddXMLAttributeAndValue(), CPLCleanTrailingSlash(), CPLFormFilename(), CPLGetBasename(), CPLGetDirname(), CPLGetExtension(), CPLGetPath(), CPLGetXMLValue(), CPLHashSetForeach(), CPLHashSetInsert(), CPLHashSetLookup(), CPLHashSetSize(), CPLProjectRelativeFilename(), CPLQuadTreeCreate(), CPLQuadTreeDestroy(), CPLQuadTreeForeach(), CPLQuadTreeGetStats(), CPLQuadTreeSearch(), CPLResetExtension(), CPLScanLong(), CPLScanUIntBig(), CPLScanULong(), CPLvsprintf(), cpl::down\_cast(), OGRCompoundCurve::EndPoint(), OGRSpatialReference::exportToPanorama(), OGRPolygon::exportToWkt(), CPLStringList::FetchNameValue(), OGRGeometryFactory::forceToMultiLineString(), OGRSimpleCurve::getPoint(), OGRPolygon::importFromWkb(), OGRPolyhedralSurface::importFromWkb(), CPLODBCDriverInstaller::InstallDriver(), CPLODBCDriverInstaller::RemoveDriver(), OGRSimpleCurve::setNumPoints(), CPLWorkerThreadPool::Setup(), OGRCompoundCurve::StartPoint(), CPLWorkerThreadPool::SubmitJob(), CPLWorkerThreadPool::SubmitJobs(), and VSISetCryptKey().

#### 12.2.2.2 CPLE\_AppDefined

```
#define CPLE_AppDefined 1
```

Application defined error

#### 12.2.2.3 CPLE\_AssertionFailed

```
#define CPLE_AssertionFailed 7
```

Assertion failed

#### 12.2.2.4 CPLE\_AWSAccessDenied

```
#define CPLE_AWSAccessDenied 14
```

AWSAccessDenied

#### 12.2.2.5 CPLE\_AWSBucketNotFound

```
#define CPLE_AWSBucketNotFound 12
```

AWSBucketNotFound

#### 12.2.2.6 CPLE\_AWSInvalidCredentials

```
#define CPLE_AWSInvalidCredentials 15
```

AWSInvalidCredentials

#### 12.2.2.7 CPLE\_AWSObjectNotFound

```
#define CPLE_AWSObjectNotFound 13
```

AWSObjectNotFound

#### 12.2.2.8 CPLE\_AWSSignatureDoesNotMatch

```
#define CPLE_AWSSignatureDoesNotMatch 16
```

AWSSignatureDoesNotMatch

#### 12.2.2.9 CPLE\_FileIO

```
#define CPLE_FileIO 3
```

File I/O error

#### 12.2.2.10 CPLE\_HttpResponse

```
#define CPLE_HttpResponse 11
```

HTTP response

#### 12.2.2.11 CPLE\_IllegalArg

```
#define CPLE_IllegalArg 5
```

Illegal argument

#### 12.2.2.12 CPLE\_None

```
#define CPLE_None 0
```

No error

#### 12.2.2.13 CPLE\_NotSupported

```
#define CPLE_NotSupported 6
```

Not supported

#### 12.2.2.14 CPLE\_NoWriteAccess

```
#define CPLE_NoWriteAccess 8
```

No write access

#### 12.2.2.15 CPLE\_ObjectNull

```
#define CPLE_ObjectNull 10
```

NULL object

#### 12.2.2.16 CPLE\_OpenFailed

```
#define CPLE_OpenFailed 4
```

Open failed

#### 12.2.2.17 CPLE\_OutOfMemory

```
#define CPLE_OutOfMemory 2
```

Out of memory error

#### 12.2.2.18 CPLE\_UserInterrupt

```
#define CPLE_UserInterrupt 9
```

User interrupted

## 12.2.2.19 VALIDATE\_POINTER0

```
#define VALIDATE_POINTER0(
    ptr,
    func )
```

**Value:**

```
do { if( CPL_NULLPTR == ptr ) \
    { \
        CPLErr const ret = VALIDATE_POINTER_ERR; \
        CPLError( ret, CPLE_ObjectNull, \
            "Pointer \'%s\' is NULL in \'%s\'.\n", #ptr, (func)); \
        return; } } while(0)
```

Validate that a pointer is not NULL

Referenced by OGR\_DS\_ReleaseResultSet(), OGR\_DS\_SetStyleTable(), OGR\_DS\_SetStyleTableDirectly(), OGR\_F\_DumpReadable(), OGR\_F\_FillUnsetWithDefault(), OGR\_F\_SetFieldBinary(), OGR\_F\_SetFieldDateTime(), OGR\_F\_SetFieldDateTimeEx(), OGR\_F\_SetFieldDouble(), OGR\_F\_SetFieldDoubleList(), OGR\_F\_SetFieldInteger(), OGR\_F\_SetFieldInteger64(), OGR\_F\_SetFieldInteger64List(), OGR\_F\_SetFieldIntegerList(), OGR\_F\_SetFieldNull(), OGR\_F\_SetFieldRaw(), OGR\_F\_SetFieldString(), OGR\_F\_SetFieldStringList(), OGR\_F\_SetNativeData(), OGR\_F\_SetNativeMediaType(), OGR\_F\_SetStyleString(), OGR\_F\_SetStyleStringDirectly(), OGR\_F\_SetStyleTable(), OGR\_F\_SetStyleTableDirectly(), OGR\_F\_UnsetField(), OGR\_G\_AddPoint(), OGR\_G\_AddPoint\_2D(), OGR\_G\_AddPointM(), OGR\_G\_AddPointZM(), OGR\_G\_AssignSpatialReference(), OGR\_G\_CloseRings(), OGR\_G\_DumpReadable(), OGR\_G\_Empty(), OGR\_G\_GetEnvelope(), OGR\_G\_GetEnvelope3D(), OGR\_G\_GetPoint(), OGR\_G\_GetPointZM(), OGR\_G\_Segmentize(), OGR\_G\_Set3D(), OGR\_G\_SetCoordinateDimension(), OGR\_G\_SetMeasured(), OGR\_G\_SetPoint(), OGR\_G\_SetPoint\_2D(), OGR\_G\_SetPointCount(), OGR\_G\_SetPointM(), OGR\_G\_SetPoints(), OGR\_G\_SetPointsZM(), OGR\_G\_SetPointZM(), OGR\_G\_SwapXY(), OGR\_GFid\_Destroy(), OGR\_GFid\_SetIgnored(), OGR\_GFid\_SetName(), OGR\_GFid\_SetSpatialRef(), OGR\_GFid\_SetType(), OGR\_L\_ResetReading(), OGR\_L\_SetSpatialFilter(), OGR\_L\_SetSpatialFilterEx(), OGR\_L\_SetSpatialFilterRect(), OGR\_L\_SetSpatialFilterRectEx(), OGR\_L\_SetStyleTable(), OGR\_L\_SetStyleTableDirectly(), OGR\_ST\_SetParamDbl(), OGR\_ST\_SetParamNum(), OGR\_ST\_SetParamStr(), OGR\_ST\_SetUnit(), OGR\_ST\_ResetStyleStringReading(), and OSRRRelease().

## 12.2.2.20 VALIDATE\_POINTER1

```
#define VALIDATE_POINTER1(
    ptr,
    func,
    rc )
```

**Value:**

```
do { if( CPL_NULLPTR == ptr ) \
    { \
        CPLErr const ret = VALIDATE_POINTER_ERR; \
        CPLError( ret, CPLE_ObjectNull, \
            "Pointer \'%s\' is NULL in \'%s\'.\n", #ptr, (func)); \
        return (rc); } } while(0)
```



Validate that a pointer is not NULL, and return rc if it is NULL

Referenced by CPLDecToDMS(), OGR\_DS\_CopyLayer(), OGR\_DS\_CreateLayer(), OGR\_DS\_DeleteLayer(), OGR\_DS\_ExecuteSQL(), OGR\_DS\_GetDriver(), OGR\_DS\_GetLayer(), OGR\_DS\_GetLayerByName(), OGR\_DS\_GetLayerCount(), OGR\_DS\_GetName(), OGR\_DS\_GetStyleTable(), OGR\_DS\_SyncToDisk(), OGR\_DS\_TestCapability(), OGR\_F\_Clone(), OGR\_F\_Create(), OGR\_F\_Equal(), OGR\_F\_GetDefnRef(), OGR\_F\_GetFID(), OGR\_F\_GetFieldAsBinary(), OGR\_F\_GetFieldAsDateTime(), OGR\_F\_GetFieldAsDateTimeEx(), OGR\_F\_GetFieldAsDouble(), OGR\_F\_GetFieldAsDoubleList(), OGR\_F\_GetFieldAsInteger(), OGR\_F\_GetFieldAsInteger64(), OGR\_F\_GetFieldAsInteger64List(), OGR\_F\_GetFieldAsIntegerList(), OGR\_F\_GetFieldAsString(), OGR\_F\_GetFieldAsStringList(), OGR\_F\_GetFieldCount(), OGR\_F\_GetFieldDefnRef(), OGR\_F\_GetFieldIndex(), OGR\_F\_GetGeometryRef(), OGR\_F\_GetGeomFieldCount(), OGR\_F\_GetGeomFieldDefnRef(), OGR\_F\_GetGeomFieldIndex(), OGR\_F\_GetGeomFieldRef(), OGR\_F\_GetNativeData(), OGR\_F\_GetNativeMediaType(), OGR\_F\_GetRawFieldRef(), OGR\_F\_GetStyleString(), OGR\_F\_GetStyleTable(), OGR\_F\_IsFieldNull(), OGR\_F\_IsFieldSet(), OGR\_F\_IsFieldSetAndNotNull(), OGR\_F\_SetFID(), OGR\_F\_SetFrom(), OGR\_F\_SetFromWithMap(), OGR\_F\_SetGeometry(), OGR\_F\_SetGeometryDirectly(), OGR\_F\_SetGeomField(), OGR\_F\_SetGeomFieldDirectly(), OGR\_F\_StealGeometry(), OGR\_F\_Validate(), OGR\_FD\_IsSame(), OGR\_G\_AddGeometry(), OGR\_G\_AddGeometryDirectly(), OGR\_G\_Area(), OGR\_G\_Boundary(), OGR\_G\_Buffer(), OGR\_G\_Centroid(), OGR\_G\_Clone(), OGR\_G\_Contains(), OGR\_G\_ConvexHull(), OGR\_G\_CoordinateDimension(), OGR\_G\_Crosses(), OGR\_G\_DelaunayTriangulation(), OGR\_G\_Difference(), OGR\_G\_Disjoint(), OGR\_G\_Distance(), OGR\_G\_Distance3D(), OGR\_G\_Equals(), OGR\_G\_ExportToIsoWkb(), OGR\_G\_ExportToIsoWkt(), OGR\_G\_ExportToJsonEx(), OGR\_G\_ExportToWkb(), OGR\_G\_ExportToWkt(), OGR\_G\_GetCoordinateDimension(), OGR\_G\_GetCurveGeometry(), OGR\_G\_GetDimension(), OGR\_G\_GetGeometryCount(), OGR\_G\_GetGeometryName(), OGR\_G\_GetGeometryRef(), OGR\_G\_GetGeometryType(), OGR\_G\_GetLinearGeometry(), OGR\_G\_GetM(), OGR\_G\_GetPointCount(), OGR\_G\_GetPoints(), OGR\_G\_GetPointsZM(), OGR\_G\_GetSpatialReference(), OGR\_G\_GetX(), OGR\_G\_GetY(), OGR\_G\_GetZ(), OGR\_G\_HasCurveGeometry(), OGR\_G\_ImportFromWkb(), OGR\_G\_ImportFromWkt(), OGR\_G\_Intersection(), OGR\_G\_Intersects(), OGR\_G\_Is3D(), OGR\_G\_IsEmpty(), OGR\_G\_IsMeasured(), OGR\_G\_IsRing(), OGR\_G\_IsSimple(), OGR\_G\_IsValid(), OGR\_G\_Length(), OGR\_G\_Overlaps(), OGR\_G\_PointOnSurface(), OGR\_G\_Polygonize(), OGR\_G\_RemoveGeometry(), OGR\_G\_Simplify(), OGR\_G\_SimplifyPreserveTopology(), OGR\_G\_SymDifference(), OGR\_G\_Touches(), OGR\_G\_Transform(), OGR\_G\_TransformTo(), OGR\_G\_Union(), OGR\_G\_UnionCascaded(), OGR\_G\_Value(), OGR\_G\_Within(), OGR\_G\_WkbSize(), OGR\_GFid\_GetNameRef(), OGR\_GFid\_GetSpatialRef(), OGR\_GFid\_GetType(), OGR\_GFid\_IsIgnored(), OGR\_L\_AlterFieldDefn(), OGR\_L\_Clip(), OGR\_L\_CommitTransaction(), OGR\_L\_CreateFeature(), OGR\_L\_CreateField(), OGR\_L\_CreateGeomField(), OGR\_L\_DeleteFeature(), OGR\_L\_DeleteField(), OGR\_L\_Erase(), OGR\_L\_FindFieldIndex(), OGR\_L\_GetExtent(), OGR\_L\_GetExtentEx(), OGR\_L\_GetFeature(), OGR\_L\_GetFeatureCount(), OGR\_L\_GetFIDColumn(), OGR\_L\_GetGeometryColumn(), OGR\_L\_GetGeomType(), OGR\_L\_GetLayerDefn(), OGR\_L\_GetName(), OGR\_L\_GetNextFeature(), OGR\_L\_GetSpatialFilter(), OGR\_L\_GetSpatialRef(), OGR\_L\_GetStyleTable(), OGR\_L\_Identity(), OGR\_L\_Intersection(), OGR\_L\_ReorderField(), OGR\_L\_ReorderFields(), OGR\_L\_RollbackTransaction(), OGR\_L\_SetAttributeFilter(), OGR\_L\_SetFeature(), OGR\_L\_SetIgnoredFields(), OGR\_L\_SetNextByIndex(), OGR\_L\_StartTransaction(), OGR\_L\_SymDifference(), OGR\_L\_SyncToDisk(), OGR\_L\_TestCapability(), OGR\_L\_Union(), OGR\_L\_Update(), OGR\_SM\_AddPart(), OGR\_SM\_AddStyle(), OGR\_SM\_GetPart(), OGR\_SM\_GetPartCount(), OGR\_SM\_InitFromFeature(), OGR\_SM\_InitStyleString(), OGR\_ST\_GetParamDbf(), OGR\_ST\_GetParamNum(), OGR\_ST\_GetParamStr(), OGR\_ST\_GetRGBFromStyleString(), OGR\_ST\_GetStyleString(), OGR\_ST\_GetType(), OGR\_ST\_GetUnit(), OGR\_STBL\_AddStyle(), OGR\_STBL\_Find(), OGR\_STBL\_GetLastStyleName(), OGR\_STBL\_GetNextStyle(), OGR\_STBL\_LoadStyleTable(), OGR\_STBL\_SaveStyleTable(), OGRGeocode(), OGRGeocodeReverse(), OSRAutoIdentifyEPSG(), OSRConvertToOtherProjection(), OSRCopyGeogCSFrom(), OSRDereference(), OSREPSGTreatsAsLatLong(), OSREPSGTreatsAsNorthingEasting(), OSRExportToERM(), OSRExportToMICoordSys(), OSRExportToPanorama(), OSRExportToPCI(), OSRExportToUSGS(), OSRExportToXML(), OSRFindMatches(), OSRFixup(), OSRFixupOrdering(), OSRGetAngularUnits(), OSRGetAuthorityCode(), OSRGetAuthorityName(), OSRGetAxis(), OSRGetInvFlattening(), OSRGetLinearUnits(), OSRGetNormProjParm(), OSRGetPrimeMeridian(), OSRGetProjParm(), OSRGetSemiMajor(), OSRGetSemiMinor(), OSRGetTargetLinearUnits(), OSRGetTOWGS84(), OSRGetUTMZone(), OSRImportFromDict(), OSRImportFromERM(), OSRImportFromESRI(), OSRImportFromMICoordSys(), OSRImportFromOzi(), OSRImportFromPanorama(), OSRImportFromPCI(), OSRImportFromProj4(), OSRImportFromUrl(), OSRImportFromUSGS(), OSRImportFromWkt(), OSRImportFromXML(), OSRIsCompound(), OSRIsGeocentric(), OSRIsGeographic(), OSRIsLocal(), OSRIsProjected(), OSRIsSame(), OSRIsSameGeogCS(), OSRIsSameVertCS(), OSRIsVertical(), OSRMorphFromESRI(), OSRMorphToESRI(), OSRReference(), OSRSetACEA(), OSRSetAE(), OSRSetAngularUnits(), OSRSetAuthority(), OSRSetAxes(), OSRSetBonne(), OSRSetCEA(), OSRSetCompoundCS(), OSRSetCS(), OSRSetEC(), OSRSetEckert(), OSRSetEckertIV(), OSRSetEckertVI(), OSRSetEquirectangular(), OSRSetEquirectangular2(), OSRSetGaussSchreiberTMercator(), OSRSetGeocCS(),

OSRSetGeogCS(), OSRSetGEOS(), OSRSetGH(), OSRSetGnomonic(), OSRSetGS(), OSRSetHOM(), OSRSetHOM2PNO(), OSRSetHOMAC(), OSRSetIGH(), OSRSetIWMPolyconic(), OSRSetKrovak(), OSRSetLAE↵A(), OSRSetLCC(), OSRSetLCC1SP(), OSRSetLCCB(), OSRSetLinearUnits(), OSRSetLinearUnitsAndUpdate↵Parameters(), OSRSetLocalCS(), OSRSetMC(), OSRSetMercator(), OSRSetMercator2SP(), OSRSetMollweide(), OSRSetNormProjParm(), OSRSetNZMG(), OSRSetOrthographic(), OSRSetOS(), OSRSetPolyconic(), OSR↵SetProjCS(), OSRSetProjection(), OSRSetProjParm(), OSRSetPS(), OSRSetQSC(), OSRSetRobinson(), OSR↵SetSCH(), OSRSetSinusoidal(), OSRSetSOC(), OSRSetStatePlane(), OSRSetStatePlaneWithUnits(), OSRSet↵Stereographic(), OSRSetTargetLinearUnits(), OSRSetTM(), OSRSetTMG(), OSRSetTMSO(), OSRSetTMVariant(), OSRSetTOWGS84(), OSRSetTPED(), OSRSetUTM(), OSRSetVDG(), OSRSetVertCS(), OSRSetWagner(), OS↵RSetWellKnownGeogCS(), OSRStripCTParms(), and OSRValidate().

### 12.2.3 Typedef Documentation

#### 12.2.3.1 CPLErrorHandler

```
typedef void( * CPLErrorHandler) ( CPLErr, CPLErrorNum, const char *)
```

Callback for a custom error handler

#### 12.2.3.2 CPLErrorNum

```
typedef int CPLErrorNum
```

Error number

### 12.2.4 Enumeration Type Documentation

#### 12.2.4.1 CPLErr

```
enum CPLErr
```

Error category

### 12.2.5 Function Documentation

12.2.5.1 `_CPLAssert()`

```
void _CPLAssert (
    const char * pszExpression,
    const char * pszFile,
    int iLine )
```

Report failure of a logical assertion.

Applications would normally use the `CPLAssert()` (p. ??) 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.

12.2.5.2 `CPLDebug()`

```
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 determined by `strstr()`).

Categories are usually an identifier for the subsystem producing the error. For instance "GDAL" might be used for the GDAL core, and "TIFF" for messages from the TIFF translator.

## Parameters

<i>pszCategory</i>	name of the debugging message category.
<i>pszFormat</i>	printf() style format string for message to display. Remaining arguments are assumed to be for format.

References `CPLGetConfigOption()`, `CPLsnprintf()`, `CPLvsprintf()`, `EQUAL`, `EQUALN`, and `VSIMalloc()`.

Referenced by `CPLODBCStatement::CollectResultsInfo()`, `CPLDumpSharedList()`, `CPLHTTPFetchEx()`, `CPL↵  
LQuadTreeGetAdvisedMaxDepth()`, `CPLSetErrorHandlerEx()`, `CPLTurnFailureIntoWarning()`, `CPLUnescape↵  
String()`, `CPLvsprintf()`, `OGRGeometryFactory::createFromWkb()`, `OGRGeometryFactory::curveToLineString()`,  
`OGRSpatialReference::Dereference()`, `OGRGeometry::Distance()`, `OGRGeometry::Distance3D()`, `CPLODBC↵  
Session::EstablishSession()`, `OGRSpatialReference::exportToPanorama()`, `OGRSpatialReference::exportToPCI()`,  
`OGRSpatialReference::exportToUSGS()`, `OGRSimpleCurve::exportToWkt()`, `OGRPolygon::exportToWkt()`, `OGR↵  
MultiPoint::exportToWkt()`, `OGR_SRSNode::FixupOrdering()`, `OGRSpatialReference::GetAxis()`, `OGRLayer::Get↵  
GeomType()`, `CPLODBCStatement::GetTables()`, `OGRLayer::Identity()`, `OGRSpatialReference::importFromESRI()`,  
`OGRSpatialReference::importFromOzi()`, `OGRSpatialReference::importFromPanorama()`, `OGRSpatialReference↵  
::importFromPCI()`, `OGRSpatialReference::importFromProj4()`, `OGRTriangle::importFromWkb()`, `OGRPolyhedral↵  
Surface::importFromWkb()`, `CPLODBCDriverInstaller::InstallDriver()`, `OGRLayer::Intersection()`, `OGRLinearRing↵  
::isPointInRing()`, `OGRLinearRing::isPointOnRingBoundary()`, `OGRSpatialReference::IsSame()`, `OGRSpatial↵  
Reference::IsSameGeogCS()`, `OGRSpatialReference::morphFromESRI()`, `OGRLinearRing::OGRLinearRing()`, `O↵`

GRGeometryFactory::organizePolygons(), VSISparseFileHandle::Read(), OGRSpatialReference::SetFromUser↵  
Input(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetLocalCS(), OGRSpatialReference::Set↵  
ProjCS(), CPLWorkerThreadPool::SubmitJob(), CPLWorkerThreadPool::SubmitJobs(), OGRGeometryCollection↵  
::transform(), OGRLayer::Union(), OGRSpatialReference::Validate(), and VSIGetMemFileBuffer().

### 12.2.5.3 CPLDefaultErrorHandler()

```
void CPLDefaultErrorHandler (
    CPLErr eErrClass,
    CPLErrorNum nError,
    const char * pszErrorMsg )
```

Default error handler.

### 12.2.5.4 CPLEmergencyError()

```
void CPLEmergencyError (
    const char * pszMessage )
```

Fatal error when things are bad.

This function should be called in an emergency situation where it is unlikely that a regular error report would work. This would include in the case of heap exhaustion for even small allocations, or any failure in the process of reporting an error (such as TLS allocations).

This function should never return. After the error message has been reported as best possible, the application will abort() similarly to how **CPLError()** (p. ??) aborts on CE\_Fatal class errors.

#### Parameters

<i>pszMessage</i>	the error message to report.
-------------------	------------------------------

### 12.2.5.5 CPLError()

```
void CPLError (
    CPLErr eErrClass,
    CPLErrorNum 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 **CPLSetError↵  
Handler()** (p. ??).

The `eErrClass` argument can have the value `CE_Warning` indicating that the message is an informational warning, `CE_Failure` indicating that the action failed, but that normal recover mechanisms will be used or `CE_Fatal` meaning that a fatal error has occurred, and that **CPLError()** (p. ??) should not return.

The default behaviour of **CPLError()** (p. ??) is to report errors to `stderr`, and to `abort()` after reporting a `CE_Fatal` error. It is expected that some applications will want to suppress error reporting, and will want to install a C++ exception, or `longjmp()` approach to no local fatal error recovery.

Regardless of how application error handlers or the default error handler choose to handle an error, the error number, and message will be stored for recovery with **CPLGetLastErrorNo()** (p. ??) and **CPLGetLastErrorMsg()** (p. ??).

#### Parameters

<i>eErrClass</i>	one of <code>CE_Warning</code> , <code>CE_Failure</code> or <code>CE_Fatal</code> .
<i>err_no</i>	the error number ( <code>CPL_*</code> ) from <b>cpl_error.h</b> (p. ??).
<i>fmt</i>	a <code>printf()</code> style format string. Any additional arguments will be treated as arguments to fill in this format in a manner similar to <code>printf()</code> .

References **CPLErrorV()**.

Referenced by `OGRLayer::AlterFieldDefn()`, `OGRGeometry::Boundary()`, `OGRGeometry::Buffer()`, `OGRLineString::CastToLinearRing()`, `OGRGeometry::Centroid()`, `OGRLayer::Clip()`, `CPLODBCSession::CloseSession()`, `OGRGeometry::Contains()`, `OGRGeometry::ConvexHull()`, `CPLAtGIntBigEx()`, `CPLCloseShared()`, `CPLCopyTree()`, `CPLCorrespondingPaths()`, `CPLCreateZip()`, `CPLEscapeString()`, `CPLFGets()`, `CPLGetSymbol()`, `CPLHTTPFetchEx()`, `CPLHTTPMultiFetch()`, `CPLHTTPParseMultipartMime()`, `CPLMalloc()`, `CPLParseXMLString()`, `CPLQuadTreeInsert()`, `CPLReadLine3L()`, `CPLRealloc()`, `CPLSerializeXMLTreeToFile()`, `CPLSPrintf()`, `CPLUnescapeString()`, `CPLUnlinkTree()`, `CPLVirtualMemFileMapNew()`, `CPLVirtualMemNew()`, `OGRLayer::CreateField()`, `OGRGeometryFactory::createFromGEOS()`, `OGRLayer::CreateGeomField()`, `OGRGeometry::createFromGEOSContext()`, `OGRGeometry::Crosses()`, `CSLLoad2()`, `CSLSave()`, `OGRCurvePolygon::CurvePolyToPoly()`, `OGRGeometryFactory::curveToLineString()`, `OGRGeometry::DelaunayTriangulation()`, `OGRLayer::DeleteField()`, `OGRGeometry::Difference()`, `OGRGeometry::Disjoint()`, `OGRGeometry::Distance()`, `OGRGeometry::Distance3D()`, `OGRLayer::Erase()`, `OGRGeometry::exportToGEOS()`, `OGRSpatialReference::exportToMCoordSys()`, `OGRSpatialReference::exportToProj4()`, `OGRGeometryCollection::exportToWkb()`, `CPLODBCStatement::Fetch()`, `OGRPolyhedralSurface::get_Area()`, `OGRFeature::GetFieldAsInteger()`, `OGRFeatureDefn::GetFieldDefn()`, `OGRFeatureDefn::GetGeomFieldDefn()`, `OGRSimpleCurve::getSubLine()`, `GOA2GetRefreshToken()`, `OGRLayer::Identity()`, `OGRSpatialReference::importFromCRSURL()`, `OGRSpatialReference::ImportFromESRIStatePlaneWKT()`, `OGRSpatialReference::importFromMCoordSys()`, `OGRSpatialReference::importFromOzi()`, `OGRSpatialReference::importFromPanorama()`, `OGRSpatialReference::importFromUrl()`, `OGRSpatialReference::importFromURN()`, `OGRSpatialReference::importFromUSGS()`, `OGRSimpleCurve::importFromWkb()`, `OGRMultiSurface::importFromWkt()`, `OGRPolyhedralSurface::importFromWkt()`, `OGRSpatialReference::importFromWMAUTO()`, `CPLStringList::InsertStringDirectly()`, `OGRLayer::Intersection()`, `OGRGeometry::Intersection()`, `OGRGeometry::IsRing()`, `OGRGeometry::IsSimple()`, `OGRGeometry::IsValid()`, `CPLJSONDocument::Load()`, `CPLJSONDocument::LoadChunks()`, `CPLJSONDocument::LoadMemory()`, `CPLJSONDocument::LoadUrl()`, `OGR_DS_CreateLayer()`, `OGR_F_IsFieldNull()`, `OGR_F_IsFieldSet()`, `OGR_F_IsFieldSetAndNotNull()`, `OGR_G_AddPoint()`, `OGR_G_AddPoint_2D()`, `OGR_G_AddPointM()`, `OGR_G_AddPointZM()`, `OGR_G_Area()`, `OGR_G_Centroid()`, `OGR_G_CreateFromGML()`, `OGR_G_Equals()`, `OGR_G_ExportEnvelopeToGMLTree()`, `OGR_G_ExportToGMLEx()`, `OGR_G_GetGeometryRef()`, `OGR_G_GetPoint()`, `OGR_G_GetPoints()`, `OGR_G_GetPointsZM()`, `OGR_G_GetPointZM()`, `OGR_G_Length()`, `OGR_G_PointOnSurface()`, `OGR_G_Segmentize()`, `OGR_G_SetPoint()`, `OGR_G_SetPoint_2D()`, `OGR_G_SetPointCount()`, `OGR_G_SetPointM()`, `OGR_G_SetPoints()`, `OGR_G_SetPointsZM()`, `OGR_G_SetPointZM()`, `OGRCreateCoordinateTransformation()`, `OGRGeocode()`, `OGRGeocodeCreateSession()`, `OGRGeocodeReverse()`, `OGRTriangle::OGRTriangle()`, `OGRGeometryFactory::organizePolygons()`, `OSRCalcInvFlattening()`, `OSRCalcSemiMinorFromInvFlattening()`, `OGRGeometry::Overlaps()`, `OGRGeometry::Polygonize()`, `OGRSimpleCurve::Project()`, `OGRLayer::ReorderField()`, `OGRLayer::ReorderFields()`, `CPLJSONDocument::Save()`, `OGRSimpleCurve::segmentize()`, `OGRCurvePolygon::segmentize()`, `GOA2Manager::SetAuthFromRefreshToken()`, `GOA2Manager::SetAuthFromServiceAccount()`, `OGRSpatialReference::SetCompoundCS()`, `OGRFieldDefn::SetDefault()`, `OGRSpatialReference::SetEckert()`, `OGRFeature::SetField()`, `OGRLayer::SetSpatial`

Filter(), OGRSpatialReference::SetStatePlane(), OGRFieldDefn::SetSubType(), OGRFieldDefn::SetType(), OGRSpatialReference::SetUTM(), OGRSpatialReference::SetWagner(), OGRGeometry::Simplify(), OGRGeometry::SimplifyPreserveTopology(), OGRLayer::SymDifference(), OGRGeometry::SymDifference(), OGRGeometry::Touches(), OGRSimpleCurve::transform(), OGRGeometry::transformTo(), OGRLayer::Union(), OGRGeometry::Union(), OGRGeometry::UnionCascaded(), OGRLayer::Update(), OGRFeature::Validate(), CPLString::vPrintf(), VSICallocVerbose(), VSIIngestFile(), VSIMalloc2Verbose(), VSIMalloc3Verbose(), VSIMallocAlignedAutoVerbose(), VSIMallocVerbose(), VSIReallocVerbose(), VSIStrdupVerbose(), and OGRGeometry::Within().

#### 12.2.5.6 CPLErrorReset()

```
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.

#### 12.2.5.7 CPLErrorSetState()

```
void CPLErrorSetState (
    CPLErr eErrClass,
    CPLErrorNum 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

#### 12.2.5.8 CPLErrorV()

```
void CPLErrorV (
    CPLErr eErrClass,
    CPLErrorNum err_no,
    const char * fmt,
    va_list args )
```

Same as **CPLError()** (p. ??) but with a `va_list`

Referenced by **CPLError()**.

### 12.2.5.9 CPLGetErrorCounter()

```
GUInt32 CPLGetErrorCounter (
    void )
```

Get the error counter

Fetches the number of errors emitted in the current error context, since the last call to **CPLErrorReset()** (p. ??)

#### Returns

the error counter.

#### Since

GDAL 2.3

### 12.2.5.10 CPLGetErrorHandlerUserData()

```
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

### 12.2.5.11 CPLGetLastErrorMsg()

```
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.

#### 12.2.5.12 CPLGetLastErrorNo()

```
CPLErrorNum 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 **CPL\_ None** (0) if there are no posted errors.

#### 12.2.5.13 CPLGetLastErrorType()

```
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 **OGR\_DS\_SyncToDisk()**.

#### 12.2.5.14 CPLLoggingErrorHandler()

```
void CPLLoggingErrorHandler (
    CPLErr eErrClass,
    CPLErrorNum nError,
    const char * pszErrorMsg )
```

Error handler that logs into the file defined by the **CPL\_LOG** configuration option, or **stderr** otherwise.

#### 12.2.5.15 CPLPopErrorHandler()

```
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.

#### 12.2.5.16 CPLPushErrorHandler()

```
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

<i>pfnErrorHandlerNew</i>	new error handler function.
---------------------------	-----------------------------

## 12.2.5.17 CPLPushErrorHandlerEx()

```
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

<i>pfnErrorHandlerNew</i>	new error handler function.
<i>pUserData</i>	User data to put on the error context.

## 12.2.5.18 CPLQuietErrorHandler()

```
void CPLQuietErrorHandler (
    CPLErr eErrClass,
    CPLErrorNum nError,
    const char * pszErrorMsg )
```

Error handler that does not do anything, except for debug messages.

## 12.2.5.19 CPLSetCurrentErrorHandlerCatchDebug()

```
void CPLSetCurrentErrorHandlerCatchDebug (
    int bCatchDebug )
```

Set if the current error handler should intercept debug messages, or if they should be processed by the previous handler.

By default when installing a custom error handler, this one intercepts debug messages. In some cases, this might not be desirable and the user would prefer that the previous installed handler (or the default one if no previous installed handler exists in the stack) deal with it. In which case, this function should be called with bCatchDebug = FALSE.

## Parameters

<i>bCatchDebug</i>	FALSE if the current error handler should not intercept debug messages
--------------------	--

## Since

GDAL 2.1

## 12.2.5.20 CPLSetErrorHandler()

```
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()** (p. ??)) 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()** (p. ??) 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 CPLPushErrorHandler (and not removed with CPLPopErrorHandler) take precedence 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()** (p. ??) to suppress error reporting in a limited segment of code.

## Parameters

<i>pfnErrorHandlerNew</i>	new error handler function.
---------------------------	-----------------------------

## Returns

returns the previously installed error handler.

References CPLSetErrorHandlerEx().

## 12.2.5.21 CPLSetErrorHandlerEx()

```
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 CPLGetErrorHandlerUserData.

## Parameters

<i>pfnErrorHandlerNew</i>	new error handler function.
<i>pUserData</i>	User data to carry along with the error context.

## Returns

returns the previously installed error handler.

References CPLDebug().

Referenced by CPLSetErrorHandler().

## 12.2.5.22 CPLTurnFailureIntoWarning()

```
void CPLTurnFailureIntoWarning (
    int bOn )
```

Whether failures should be turned into warnings.

References CPLDebug().

## 12.3 cpl\_hash\_set.h File Reference

```
#include "cpl_port.h"
```

## Typedefs

- typedef struct **\_CPLHashSet** **CPLHashSet**
- typedef unsigned long(\* **CPLHashSetHashFunc**) (const void \*elt)
- typedef int(\* **CPLHashSetEqualFunc**) (const void \*elt1, const void \*elt2)
- typedef void(\* **CPLHashSetFreeEltFunc**) (void \*elt)
- typedef int(\* **CPLHashSetIterEltFunc**) (void \*elt, void \*user\_data)

## Functions

- **CPLHashSet** \* **CPLHashSetNew** ( **CPLHashSetHashFunc** fnHashFunc, **CPLHashSetEqualFunc** fnEqualFunc, **CPLHashSetFreeEltFunc** fnFreeEltFunc)
- void **CPLHashSetDestroy** ( **CPLHashSet** \*set)
- void **CPLHashSetClear** ( **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)
- int **CPLHashSetRemoveDeferRehash** ( **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)

### 12.3.1 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.

### 12.3.2 Typedef Documentation

#### 12.3.2.1 CPLHashSet

```
typedef struct _CPLHashSet CPLHashSet
```

Opaque type for a hash set

#### 12.3.2.2 CPLHashSetEqualFunc

```
typedef int(* CPLHashSetEqualFunc) (const void *elt1, const void *elt2)
```

CPLHashSetEqualFunc

#### 12.3.2.3 CPLHashSetFreeEltFunc

```
typedef void(* CPLHashSetFreeEltFunc) (void *elt)
```

CPLHashSetFreeEltFunc

## 12.3.2.4 CPLHashSetHashFunc

```
typedef unsigned long(* CPLHashSetHashFunc) (const void *elt)
```

CPLHashSetHashFunc

## 12.3.2.5 CPLHashSetIterEltFunc

```
typedef int(* CPLHashSetIterEltFunc) (void *elt, void *user_data)
```

CPLHashSetIterEltFunc

## 12.3.3 Function Documentation

## 12.3.3.1 CPLHashSetClear()

```
void CPLHashSetClear (
    CPLHashSet * set )
```

Clear all elements from a hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

## Parameters

<i>set</i>	the hash set
------------	--------------

## Since

GDAL 2.1

References CPLRealloc().

## 12.3.3.2 CPLHashSetDestroy()

```
void CPLHashSetDestroy (
    CPLHashSet * set )
```

Destroys an allocated hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

**Parameters**

<i>set</i>	the hash set
------------	--------------

References CPLFree, and CPLListDestroy().

**12.3.3.3 CPLHashSetEqualPointer()**

```
int CPLHashSetEqualPointer (
    const void * elt1,
    const void * elt2 )
```

Equality function for arbitrary pointers

**Parameters**

<i>elt1</i>	the first arbitrary pointer to compare
<i>elt2</i>	the second arbitrary pointer to compare

**Returns**

TRUE if the pointers are equal

Referenced by CPLHashSetNew().

**12.3.3.4 CPLHashSetEqualStr()**

```
int CPLHashSetEqualStr (
    const void * elt1,
    const void * elt2 )
```

Equality function for strings

**Parameters**

<i>elt1</i>	the first string to compare. May be NULL.
<i>elt2</i>	the second string to compare. May be NULL.

**Returns**

TRUE if the strings are equal

## 12.3.3.5 CPLHashSetForeach()

```
void CPLHashSetForeach (
    CPLHashSet * set,
    CPLHashSetIterEltFunc fnIterFunc,
    void * user_data )
```

Walk through the hash set and runs the provided function on all the elements

This function is provided the user\_data argument of CPLHashSetForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the hash set must *NOT* be modified during the walk.

## Parameters

<i>set</i>	the hash set.
<i>fnIterFunc</i>	the function called on each element.
<i>user_data</i>	the user data provided to the function.

References CPLAssert, \_CPLList::pData, and \_CPLList::psNext.

## 12.3.3.6 CPLHashSetHashPointer()

```
unsigned long CPLHashSetHashPointer (
    const void * elt )
```

Hash function for an arbitrary pointer

## Parameters

<i>elt</i>	the arbitrary pointer to hash
------------	-------------------------------

## Returns

the hash value of the pointer

Referenced by CPLHashSetNew().

## 12.3.3.7 CPLHashSetHashStr()

```
unsigned long CPLHashSetHashStr (
    const void * elt )
```

Hash function for a zero-terminated string

**Parameters**

<i>elt</i>	the string to hash. May be NULL.
------------	----------------------------------

**Returns**

the hash value of the string

**12.3.3.8 CPLHashSetInsert()**

```
int CPLHashSetInsert (
    CPLHashSet * set,
    void * elt )
```

Inserts an element into a hash set.

If the element was already inserted in the hash set, the previous element is replaced by the new element. If a free function was provided, it is used to free the previously inserted element

**Parameters**

<i>set</i>	the hash set
<i>elt</i>	the new element to insert in the hash set

**Returns**

TRUE if the element was not already in the hash set

References CPLAssert, \_CPLList::pData, and \_CPLList::psNext.

**12.3.3.9 CPLHashSetLookup()**

```
void* CPLHashSetLookup (
    CPLHashSet * set,
    const void * elt )
```

Returns the element found in the hash set corresponding to the element to look up The element must not be modified.

**Parameters**

<i>set</i>	the hash set
<i>elt</i>	the element to look up in the hash set



**Returns**

the element found in the hash set or NULL

References CPLAssert.

**12.3.3.10 CPLHashSetNew()**

```
CPLHashSet* CPLHashSetNew (
    CPLHashSetHashFunc fnHashFunc,
    CPLHashSetEqualFunc fnEqualFunc,
    CPLHashSetFreeEltFunc fnFreeEltFunc )
```

Creates a new hash set

The hash function must return a hash value for the elements to insert. If *fnHashFunc* is NULL, *CPLHashSetHashPointer* will be used.

The equal function must return if two elements are equal. If *fnEqualFunc* is NULL, *CPLHashSetEqualPointer* will be used.

The free function is used to free elements inserted in the hash set, when the hash set is destroyed, when elements are removed or replaced. If *fnFreeEltFunc* is NULL, elements inserted into the hash set will not be freed.

**Parameters**

<i>fnHashFunc</i>	hash function. May be NULL.
<i>fnEqualFunc</i>	equal function. May be NULL.
<i>fnFreeEltFunc</i>	element free function. May be NULL.

**Returns**

a new hash set

References CPLCalloc(), CPLHashSetEqualPointer(), CPLHashSetHashPointer(), and CPLMalloc().

**12.3.3.11 CPLHashSetRemove()**

```
int CPLHashSetRemove (
    CPLHashSet * set,
    const void * elt )
```

Removes an element from a hash set

**Parameters**

<i>set</i>	the hash set
<i>elt</i>	the new element to remove from the hash set

**Returns**

TRUE if the element was in the hash set

**12.3.3.12 CPLHashSetRemoveDeferRehash()**

```
int CPLHashSetRemoveDeferRehash (
    CPLHashSet * set,
    const void * elt )
```

Removes an element from a hash set.

This will defer potential rehashing of the set to later calls to **CPLHashSetInsert()** (p. ??) or **CPLHashSetRemove()** (p. ??).

**Parameters**

<i>set</i>	the hash set
<i>elt</i>	the new element to remove from the hash set

**Returns**

TRUE if the element was in the hash set

**Since**

GDAL 2.1

**12.3.3.13 CPLHashSetSize()**

```
int CPLHashSetSize (
    const CPLHashSet * set )
```

Returns the number of elements inserted in the hash set

Note: this is not the internal size of the hash set

**Parameters**

<i>set</i>	the hash set
------------	--------------

**Returns**

the number of elements in the hash set

References CPLAssert.

## 12.4 cpl\_http.h File Reference

```
#include "cpl_conv.h"
#include "cpl_string.h"
#include "cpl_progress.h"
#include "cpl_vsi.h"
```

### Classes

- struct **CPLMimePart**
- struct **CPLHTTPResult**
- class **GOA2Manager**

### Functions

- int **CPLHTTPEnabled** (void)  
*Return if CPLHTTP services can be useful.*
- **CPLHTTPResult** \* **CPLHTTPFetch** (const char \*pszURL, **CSLConstList** papszOptions)  
*Fetch a document from an url and return in a string.*
- **CPLHTTPResult** \* **CPLHTTPFetchEx** (const char \*pszURL, **CSLConstList** papszOptions, GDAL↔ ProgressFunc pfnProgress, void \*pProgressArg, CPLHTTPFetchWriteFunc pfnWrite, void \*pWriteArg)
- **CPLHTTPResult** \*\* **CPLHTTPMultiFetch** (const char \*const \*papszURL, int nURLCount, int nMax↔ Simultaneous, **CSLConstList** papszOptions)  
*Fetch several documents at once.*
- 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. ??)*
- void **CPLHTTPDestroyMultiResult** ( **CPLHTTPResult** \*\*papsResults, int nCount)  
*Clean the memory associated with the return value of **CPLHTTPMultiFetch()** (p. ??)*
- int **CPLHTTPParseMultipartMime** ( **CPLHTTPResult** \*psResult)  
*Parses 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)
- char \*\* **GOA2GetAccessTokenFromServiceAccount** (const char \*pszPrivateKey, const char \*pszClient↔ Email, const char \*pszScope, **CSLConstList** papszAdditionalClaims, **CSLConstList** papszOptions)
- char \*\* **GOA2GetAccessTokenFromCloudEngineVM** ( **CSLConstList** papszOptions)
- bool **CPLIsMachinePotentiallyGCEInstance** ()
- bool **CPLIsMachineForSureGCEInstance** ()

#### 12.4.1 Detailed Description

Interface for downloading HTTP, FTP documents

#### 12.4.2 Function Documentation

#### 12.4.2.1 CPLHTTPDestroyMultiResult()

```
void CPLHTTPDestroyMultiResult (
    CPLHTTPResult ** papsResults,
    int nCount )
```

Clean the memory associated with the return value of **CPLHTTPMultiFetch()** (p. ??)

##### Parameters

<i>papsResults</i>	pointer to the return value of <b>CPLHTTPMultiFetch()</b> (p. ??)
<i>nCount</i>	value of the nURLCount parameter passed to <b>CPLHTTPMultiFetch()</b> (p. ??)

Since

GDAL 2.3

References CPLFree, and CPLHTTPDestroyResult().

#### 12.4.2.2 CPLHTTPDestroyResult()

```
void CPLHTTPDestroyResult (
    CPLHTTPResult * psResult )
```

Clean the memory associated with the return value of **CPLHTTPFetch()** (p. ??)

##### Parameters

<i>psResult</i>	pointer to the return value of <b>CPLHTTPFetch()</b> (p. ??)
-----------------	--

References CPLFree, CSLDestroy(), CPLHTTPResult::nMimePartCount, CPLHTTPResult::pabyData, CPLMimePart::papszHeaders, CPLHTTPResult::papszHeaders, CPLHTTPResult::pasMimePart, CPLHTTPResult::pszContentType, and CPLHTTPResult::pszErrBuf.

Referenced by CPLHTTPDestroyMultiResult(), and CPLJSONDocument::LoadUrl().

#### 12.4.2.3 CPLHTTPEnabled()

```
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

## 12.4.2.4 CPLHTTPFetch()

```
CPLHTTPResult* CPLHTTPFetch (
    const char * pszURL,
    CSLConstList papszOptions )
```

Fetch a document from an url and return in a string.

## Parameters

<i>pszURL</i>	valid URL recognized by underlying download library (libcurl)
---------------	---

## Parameters

<i>papszOptions</i>	<p>option list as a NULL-terminated array of strings. May be NULL. The following options are handled :</p> <ul style="list-style-type: none"> <li>• CONNECTTIMEOUT=val, where val is in seconds (possibly with decimals). This is the maximum delay for the connection to be established before being aborted (GDAL &gt;= 2.2).</li> <li>• TIMEOUT=val, where val is in seconds. This is the maximum delay for the whole request to complete before being aborted.</li> <li>• LOW_SPEED_TIME=val, where val is in seconds. This is the maximum time where the transfer speed should be below the LOW_SPEED_LIMIT (if not specified 1b/s), before the transfer to be considered too slow and aborted. (GDAL &gt;= 2.1)</li> <li>• LOW_SPEED_LIMIT=val, where val is in bytes/second. See LOW_SPEED_TIME. Has only effect if LOW_SPEED_TIME is specified too. (GDAL &gt;= 2.1)</li> <li>• HEADERS=val, where val is an extra header to use when getting a web page. For example "Accept: application/x-ogcwk"</li> <li>• HEADER_FILE=filename: filename of a text file with "key: value" headers. (GDAL &gt;= 2.2)</li> <li>• HTTPAUTH=[BASIC/NTLM/GSSNEGOTIATE/ANY] to specify an authentication scheme to use.</li> <li>• USERPWD=userid:password to specify a user and password for authentication</li> <li>• POSTFIELDS=val, where val is a nul-terminated string to be passed to the server with a POST request.</li> <li>• PROXY=val, to make requests go through a proxy server, where val is of the form proxy.server.com:port_number</li> <li>• PROXYUSERPWD=val, where val is of the form username:password</li> <li>• PROXYAUTH=[BASIC/NTLM/DIGEST/ANY] to specify an proxy authentication scheme to use.</li> <li>• NETRC=[YES/NO] to enable or disable use of \$HOME/.netrc, default YES.</li> <li>• CUSTOMREQUEST=val, where val is GET, PUT, POST, DELETE, etc.. (GDAL &gt;= 1.9.0)</li> <li>• COOKIE=val, where val is formatted as COOKIE1=VALUE1; COOKIE2=VALUE2; ...</li> <li>• MAX_RETRY=val, where val is the maximum number of retry attempts if a 429, 502, 503 or 504 HTTP error occurs. Default is 0. (GDAL &gt;= 2.0)</li> <li>• RETRY_DELAY=val, where val is the number of seconds between retry attempts. Default is 30. (GDAL &gt;= 2.0)</li> <li>• MAX_FILE_SIZE=val, where val is a number of bytes (GDAL &gt;= 2.2)</li> <li>• CAINFO=/path/to/bundle.crt. This is path to Certificate Authority (CA) bundle file. By default, it will be looked in a system location. If the CAINFO options is not defined, GDAL will also look if the CURL_CA_BUNDLE environment variable is defined to use it as the CAINFO value, and as a fallback to the SSL_CERT_FILE environment variable. (GDAL &gt;= 2.1.3)</li> <li>• HTTP_VERSION=1.0/1.1/2/TLS (GDAL &gt;= 2.3). Specify HTTP version to use. Will default to 1.1 generally (except on some controlled environments, like Google Compute Engine VMs, where 2TLS will be the default). Support for HTTP/2 requires curl 7.33 or later, built against nghttp2. "2TLS" means that HTTP/2 will be attempted for HTTPS connections only. Whereas "2" means that HTTP/2 will be attempted for HTTP or HTTPS.</li> </ul>
	<p>Generated by Doxygen</p> <ul style="list-style-type: none"> <li>• SSL_VERIFYSTATUS=YES/NO (GDAL &gt;= 2.3, and curl &gt;= 7.41): determines whether the status of the server cert using the "Certificate Status Request" TLS extension (aka. OCSP stapling) should be checked. If this option is enabled but the server does not</li> </ul>

## Parameters

Alternatively, if not defined in the `papszOptions` arguments, the `CONNECTTIMEOUT`, `TIMEOUT`, `LOW_SPEED_TIME`, `LOW_SPEED_LIMIT`, `USERPWD`, `PROXY`, `PROXYUSERPWD`, `PROXYAUTH`, `NETRC`, `MAX_RETRY` and `RETRY_DELAY`, `HEADER_FILE`, `HTTP_VERSION`, `SSL_VERIFYSTATUS`, `USE_CAPI_STORE` values are searched in the configuration options respectively named `GDAL_HTTP_CONNECTTIMEOUT`, `GDAL_HTTP_TIMEOUT`, `GDAL_HTTP_LOW_SPEED_TIME`, `GDAL_HTTP_LOW_SPEED_LIMIT`, `GDAL_HTTP_USERPWD`, `GDAL_HTTP_PROXY`, `GDAL_HTTP_PROXYUSERPWD`, `GDAL_PROXY_AUTH`, `GDAL_HTTP_NETRC`, `GDAL_HTTP_MAX_RETRY`, `GDAL_HTTP_RETRY_DELAY`, `GDAL_HTTP_HEADER_FILE`, `GDAL_HTTP_VERSION`, `GDAL_HTTP_SSL_VERIFYSTATUS`, `GDAL_HTTP_USE_CAPI_STORE`

## Returns

a `CPLHTTPResult*` structure that must be freed by **`CPLHTTPDestroyResult()`** (p. ??), or `NULL` if libcurl support is disabled

References `CPLHTTPFetchEx()`.

Referenced by `GOA2GetAccessTokenFromCloudEngineVM()`, and `GOA2GetRefreshToken()`.

## 12.4.2.5 CPLHTTPFetchEx()

```
CPLHTTPResult* CPLHTTPFetchEx (
    const char * pszURL,
    CSLConstList papszOptions,
    GDALProgressFunc pfnProgress,
    void * pProgressArg,
    CPLHTTPFetchWriteFunc pfnWrite,
    void * pWriteArg )
```

Fetch a document from an url and return in a string.

## Parameters

<i>pszURL</i>	Url to fetch document from web.
<i>papszOptions</i>	Option list as a NULL-terminated array of strings. Available keys see in <code>CPLHTTPFetch</code> .
<i>pfnProgress</i>	Callback for reporting algorithm progress matching the <code>GDALProgressFunc()</code> semantics. May be <code>NULL</code> .
<i>pProgressArg</i>	Callback argument passed to <code>pfnProgress</code> .
<i>pfnWrite</i>	Write function pointer matching the <code>CPLHTTPWriteFunc()</code> semantics. May be <code>NULL</code> .
<i>pWriteArg</i>	Argument which will pass to a write function.

## Returns

A `CPLHTTPResult*` structure that must be freed by **`CPLHTTPDestroyResult()`** (p. ??), or `NULL` if libcurl support is disabled.

References `CPLCalloc()`, `CPLDebug()`, `CPLError()`, `CPLGetConfigOption()`, `CPLSprintf()`, `CPLStrdup()`, `CPLTestBool()`, `CSLFetchNameValue()`, `CPLHTTPResult::nStatus`, `CPLHTTPResult::pszErrBuf`, `STARTS_WITH`, and `VSIGetMemFileBuffer()`.

Referenced by `CPLHTTPFetch()`, and `CPLJSONDocument::LoadUrl()`.

#### 12.4.2.6 CPLHTTPMultiFetch()

```
CPLHTTPResult** CPLHTTPMultiFetch (
    const char *const * papszURL,
    int nURLCount,
    int nMaxSimultaneous,
    CSLConstList papszOptions )
```

Fetch several documents at once.

##### Parameters

<i>papszURL</i>	array of valid URLs recognized by underlying download library (libcurl)
<i>nURLCount</i>	number of URLs of <i>papszURL</i>
<i>nMaxSimultaneous</i>	maximum number of downloads to issue simultaneously. Any negative or zero value means unlimited.
<i>papszOptions</i>	option list as a NULL-terminated array of strings. May be NULL. Refer to <b>CPLHTTPFetch()</b> (p. ??) for valid options.

##### Returns

an array of `CPLHTTPResult*` structures that must be freed by **CPLHTTPDestroyMultiResult()** (p. ??) or `NULL` if libcurl support is disabled

##### Since

GDAL 2.3

References `CPLError()`.

#### 12.4.2.7 CPLHTTPParseMultipartMime()

```
int CPLHTTPParseMultipartMime (
    CPLHTTPResult * psResult )
```

Parses a MIME multipart message.

This function will iterate over each part and put it in a separate element of the `pasMimePart` array of the provided `psResult` structure.

##### Parameters

<i>psResult</i>	pointer to the return value of <b>CPLHTTPFetch()</b> (p. ??)
-----------------	--



**Returns**

TRUE if the message contains MIME multipart message.

References CPLError(), CPLHTTPResult::nMimePartCount, and CPLHTTPResult::pszContentType.

**12.4.2.8 CPLIsMachineForSureGCEInstance()**

```
bool CPLIsMachineForSureGCEInstance ( )
```

Returns whether the current machine is surely a Google Compute Engine instance.

This does a very quick check without network access. Note: only works for Linux GCE instances.

**Returns**

true if the current machine is surely a GCE instance.

**Since**

GDAL 2.3

References CPLGetConfigOption(), CPLReadLineL(), CPLTestBool(), VSIFCloseL(), and VSIFOpenL().

Referenced by CPLIsMachinePotentiallyGCEInstance().

**12.4.2.9 CPLIsMachinePotentiallyGCEInstance()**

```
bool CPLIsMachinePotentiallyGCEInstance ( )
```

Returns whether the current machine is potentially a Google Compute Engine instance.

This does a very quick check without network access. To confirm if the machine is effectively a GCE instance, metadata.google.internal must be queried.

**Returns**

true if the current machine is potentially a GCE instance.

**Since**

GDAL 2.3

References CPLGetConfigOption(), CPLIsMachineForSureGCEInstance(), and CPLTestBool().

**12.4.2.10 GOA2GetAccessToken()**

```
char* GOA2GetAccessToken (
    const char * pszRefreshToken,
    const char * pszScope )
```

Fetch access token using refresh token.

The permanent refresh token is used to fetch a temporary (usually one hour) access token using Google OAuth2 web services.

A CPLError will be reported if the request fails for some reason. Common reasons include the refresh token having been revoked by the user or http connection problems.

## Parameters

<i>pszRefreshToken</i>	the refresh token from <b>GOA2GetRefreshToken()</b> (p. ??).
<i>pszScope</i>	the scope for which it is valid. Currently unused

## Returns

access token, to be freed with **CPLFree()** (p. ??), null on failure.

References CPLStrdup(), CSLDestroy(), and CSLFetchNameValue().

## 12.4.2.11 GOA2GetAccessTokenFromCloudEngineVM()

```
char** GOA2GetAccessTokenFromCloudEngineVM (
    CSLConstList papszOptions )
```

Fetch access token using Cloud Engine internal REST API

The default service accounts bound to the current Google Cloud Engine VM is used for OAuth2 authentication

A CPLError will be reported if the request fails for some reason. Common reasons include the refresh token having been revoked by the user or http connection problems.

## Parameters

<i>papszOptions</i>	NULL terminated list of options. None currently
---------------------	---

## Returns

a list of key=value pairs, including a access\_token and expires\_in

## Since

GDAL 2.3

References CPLStringList::AddString(), CPLGetConfigOption(), CPLHTTPFetch(), and CSLFetchNameValueDef().

Referenced by GOA2Manager::GetBearer().

## 12.4.2.12 GOA2GetAccessTokenFromServiceAccount()

```
char** GOA2GetAccessTokenFromServiceAccount (
    const char * pszPrivateKey,
    const char * pszClientEmail,
    const char * pszScope,
```

```
CSLConstList papszAdditionalClaims,  
CSLConstList papszOptions )
```

Fetch access token using Service Account OAuth2

See <https://developers.google.com/identity/protocols/OAuth2ServiceAccount>

A CPLError will be reported if the request fails for some reason.

## Parameters

<i>pszPrivateKey</i>	Private key as a RSA private key
<i>pszClientEmail</i>	Client email
<i>pszScope</i>	the service being requested
<i>papszAdditionalClaims</i>	additional claims, or NULL
<i>papszOptions</i>	NULL terminated list of options. None currently

## Returns

a list of key=value pairs, including a `access_token` and `expires_in`

## Since

GDAL 2.3

See <https://developers.google.com/identity/protocols/OAuth2ServiceAccount> and <https://jwt.io/>

Referenced by `GOA2Manager::GetBearer()`.

## 12.4.2.13 GOA2GetAuthorizationURL()

```
char* GOA2GetAuthorizationURL (
    const char * pszScope )
```

Return authorization url for a given scope.

Returns the URL that a user should visit, and use for authentication in order to get an "auth token" indicating their willingness to use a service.

Note that when the user visits this url they will be asked to login (using a google/gmail/etc) account, and to authorize use of the requested scope for the application "GDAL/OGR". Once they have done so, they will be presented with a lengthy string they should "enter into their application". This is the "auth token" to be passed to **GOA2GetRefreshToken()** (p. ??). The "auth token" can only be used once.

This function should never fail.

## Parameters

<i>pszScope</i>	the service being requested, not yet URL encoded, such as "https://www.googleapis.com/auth/fusiontables".
-----------------	---

## Returns

the URL to visit - should be freed with **CPLFree()** (p. ??).

References `CPLES_URL`, `CPLEscapeString()`, `CPLGetConfigOption()`, `CPLStrdup()`, `CPLString::Printf()`, and `CPLString::Seize()`.

Referenced by GOA2GetRefreshToken().

#### 12.4.2.14 GOA2GetRefreshToken()

```
char* GOA2GetRefreshToken (
    const char * pszAuthToken,
    const char * pszScope )
```

Turn Auth Token into a Refresh Token.

A one time "auth token" provided by the user is turned into a reusable "refresh token" using a google oauth2 web service.

A CPLError will be reported if the translation fails for some reason. Common reasons include the auth token already having been used before, it not being appropriate for the passed scope and configured client api or http connection problems. NULL is returned on error.

##### Parameters

<i>pszAuthToken</i>	the authorization token from the user.
<i>pszScope</i>	the scope for which it is valid.

##### Returns

refresh token, to be freed with **CPLFree()** (p. ??), null on failure.

References CPLStringList::AddString(), CPLError(), CPLGetConfigOption(), CPLHTTPFetch(), GOA2GetAuthorizationURL(), CPLHTTPResult::pabyData, CPLString::Printf(), and CPLString::Seize().

## 12.5 cpl\_json.h File Reference

```
#include "cpl_progress.h"
#include <string>
#include <vector>
```

### Classes

- class **CPLJSONObject**  
The **CPLJSONArray** (p. ??) class holds JSON object from **CPLJSONDocument** (p. ??).
- class **CPLJSONArray**  
The **JSONArray** class JSON array from **JSONDocument**.
- class **CPLJSONDocument**  
The **CPLJSONDocument** (p. ??) class Wrapper class around json-c library.

### 12.5.1 Detailed Description

Interface for read and write JSON documents

## 12.6 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** \*const psList, int nPosition)
- int **CPLListCount** (const **CPLList** \*psList)
- **CPLList \* CPLListRemove** ( **CPLList** \*psList, int nPosition)
- void **CPLListDestroy** ( **CPLList** \*psList)
- **CPLList \* CPLListGetNext** (const **CPLList** \*psElement)
- void \* **CPLListGetData** (const **CPLList** \*psElement)

### 12.6.1 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.

### 12.6.2 Typedef Documentation

#### 12.6.2.1 CPLList

```
typedef struct _CPLList CPLList
```

List element structure.

### 12.6.3 Function Documentation

#### 12.6.3.1 CPLListAppend()

```
CPLList* CPLListAppend (
    CPLList * psList,
    void * pData )
```

Append an object list and return a pointer to the modified list. If the input list is NULL, then a new list is created.

**Parameters**

<i>psList</i>	pointer to list head.
<i>pData</i>	pointer to inserted data object. May be NULL.

**Returns**

pointer to the head of modified list.

References CPLListGetLast(), CPLMalloc(), \_CPLList::pData, and \_CPLList::pNext.

Referenced by CPLListInsert().

**12.6.3.2 CPLListCount()**

```
int CPLListCount (
    const CPLList * psList )
```

Return the number of elements in a list.

**Parameters**

<i>psList</i>	pointer to list head.
---------------	-----------------------

**Returns**

number of elements in a list.

References \_CPLList::pNext.

Referenced by CPLListInsert().

**12.6.3.3 CPLListDestroy()**

```
void CPLListDestroy (
    CPLList * psList )
```

Destroy a list. Caller responsible for freeing data objects contained in list elements.

**Parameters**

<i>psList</i>	pointer to list head.
---------------	-----------------------

References CPLFree, and \_CPLList::pNext.



Referenced by CPLHashSetDestroy(), and CPLWorkerThreadPool::~CPLWorkerThreadPool().

#### 12.6.3.4 CPLListGet()

```
CPLList* CPLListGet (
    CPLList * psList,
    int nPosition )
```

Return the pointer to the specified element in a list.

##### Parameters

<i>psList</i>	pointer to list head.
<i>nPosition</i>	the index of the element in the list, 0 being the first element.

##### Returns

pointer to the specified element in a list.

References `_CPLList::psNext`.

#### 12.6.3.5 CPLListGetData()

```
void* CPLListGetData (
    const CPLList * psElement )
```

Return pointer to the data object contained in given list element.

##### Parameters

<i>psElement</i>	pointer to list element.
------------------	--------------------------

##### Returns

pointer to the data object contained in given list element.

References `_CPLList::pData`.

#### 12.6.3.6 CPLListGetLast()

```
CPLList* CPLListGetLast (
    CPLList *const psList )
```

Return the pointer to last element in a list.

## Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

## Returns

pointer to last element in a list.

References `_CPLList::psNext`.

Referenced by `CPLListAppend()`, and `CPLListInsert()`.

12.6.3.7 `CPLListGetNext()`

```
CPLList* CPLListGetNext (
    const CPLList * psElement )
```

Return the pointer to next element in a list.

## Parameters

<i>psElement</i>	pointer to list element.
------------------	--------------------------

## Returns

pointer to the list element preceded by the given element.

References `_CPLList::psNext`.

12.6.3.8 `CPLListInsert()`

```
CPLList* CPLListInsert (
    CPLList * psList,
    void * pData,
    int nPosition )
```

Insert an object into list at specified position (zero based). If the input list is NULL, then a new list is created.

## Parameters

<i>psList</i>	pointer to list head.
<i>pData</i>	pointer to inserted data object. May be NULL.
<i>nPosition</i>	position number to insert an object.

**Returns**

pointer to the head of modified list.

References `CPLListAppend()`, `CPLListCount()`, `CPLListGetLast()`, `CPLMalloc()`, `_CPLList::pData`, and `_CPLList::psNext`.

**12.6.3.9 CPLListRemove()**

```
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**

<i>psList</i>	pointer to list head.
<i>nPosition</i>	position number to delete an element.

**Returns**

pointer to the head of modified list.

References `CPLFree`, and `_CPLList::psNext`.

**12.7 cpl\_minixml.h File Reference**

```
#include "cpl_port.h"
```

**Classes**

- struct **CPLXMLNode**
- class **CPLXMLTreeCloser**

**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** (const CPLXMLNode \*poRoot, const char \*pszPath, const char \*psz↵  
Default)  
*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** (const 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.*

### 12.7.1 Detailed Description

Definitions for CPL mini XML Parser/Serializer.

### 12.7.2 Typedef Documentation

### 12.7.2.1 CPLXMLNode

```
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 hierarchical tree structure for a document can be represented as a tree of **CPLXMLNode** (p. ??) structures.

## 12.7.3 Enumeration Type Documentation

### 12.7.3.1 CPLXMLNodeType

```
enum CPLXMLNodeType
```

XML node type

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

## 12.7.4 Function Documentation

### 12.7.4.1 CPLAddXMLAttributeAndValue()

```
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:

```
CPLXMLNode *psAttributeNode;

psAttributeNode = CPLCreateXMLNode( psParent, CXT_Attribute, pszName );
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

<i>psParent</i>	the parent node to which the resulting node should be attached. Must not be NULL.
<i>pszName</i>	the attribute name to create.
<i>pszValue</i>	the text to attach to the attribute. Must not be NULL.

## Since

GDAL 2.0

References CPLAssert, CPLCreateXMLNode(), CXT\_Attribute, and CXT\_Text.

## 12.7.4.2 CPLAddXMLChild()

```
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 (its psNext is not NULL) they will be trimmed, but if the child has children they are carried with it.

## Parameters

<i>psParent</i>	the node to attach the child to. May not be NULL.
<i>psChild</i>	the child to add to the parent. May not be NULL. Should not be a child of any other parent.

References CXT\_Attribute, CPLXMLNode::eType, CPLXMLNode::psChild, and CPLXMLNode::psNext.

## 12.7.4.3 CPLAddXMLSibling()

```
void CPLAddXMLSibling (
    CPLXMLNode * psOlderSibling,
    CPLXMLNode * psNewSibling )
```

Add new sibling.

The passed psNewSibling is added to the end of siblings of the psOlderSibling node. That is, it is added to the end of the psNext chain. There is no special handling if psNewSibling is an attribute. If this is required, use **CPLAddXMLChild()** (p. ??).

## Parameters

<i>psOlderSibling</i>	the node to attach the sibling after.
<i>psNewSibling</i>	the node to add at the end of psOlderSiblings psNext chain.

References CPLXMLNode::psNext.

## 12.7.4.4 CPLCleanXMLElementName()

```
void CPLCleanXMLElementName (
    char * pszTarget )
```

Make string into safe XML token.

Modifies a string in place to try and make it into a legal XML token that can be used as an element name. This is accomplished by changing any characters not legal in a token into an underscore.

NOTE: This function should implement the rules in section 2.3 of <http://www.w3.org/TR/xml11/> but it doesn't yet do that properly. We only do a rough approximation of that.

## Parameters

<i>pszTarget</i>	the string to be adjusted. It is altered in place.
------------------	--

## 12.7.4.5 CPLCloneXMLTree()

```
CPLXMLNode* CPLCloneXMLTree (
    const CPLXMLNode * psTree )
```

Copy tree.

Creates a deep copy of a **CPLXMLNode** (p. ??) tree.

## Parameters

<i>psTree</i>	the tree to duplicate.
---------------	------------------------

## Returns

a copy of the whole tree.

References CPLCreateXMLNode(), CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

#### 12.7.4.6 CPLCreateXMLElementAndValue()

```
CPLXMLNode* CPLCreateXMLElementAndValue (
    CPLXMLNode * psParent,
    const char * pszName,
    const char * pszValue )
```

Create an element and text value.

This is function is a convenient short form for:

```
CPLXMLNode *psTextNode;
CPLXMLNode *psElementNode;

psElementNode = CPLCreateXMLNode( psParent, CXT_Element, pszName );
psTextNode = CPLCreateXMLNode( psElementNode, CXT_Text, pszValue );

return psElementNode;
```

It creates a CXT\_Element node, with a CXT\_Text child, and attaches the element to the passed parent.

##### Parameters

<i>psParent</i>	the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.
<i>pszName</i>	the element name to create.
<i>pszValue</i>	the text to attach to the element. Must not be NULL.

##### Returns

the pointer to the new element node.

References CPLCreateXMLNode(), CXT\_Element, and CXT\_Text.

#### 12.7.4.7 CPLCreateXMLNode()

```
CPLXMLNode* CPLCreateXMLNode (
    CPLXMLNode * poParent,
    CPLXMLNodeType eType,
    const char * pszText )
```

Create an document tree item.

Create a single **CPLXMLNode** (p. ??) object with the desired value and type, and attach it as a child of the indicated parent.

##### Parameters

<i>poParent</i>	the parent to which this node should be attached as a child. May be NULL to keep as free standing.
<i>eType</i>	the type of the newly created node
<i>pszText</i>	the value of the newly created node



**Returns**

the newly created node, now owned by the caller (or parent node).

References CPLCalloc(), CPLStrdup(), CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by CPLAddXMLAttributeAndValue(), CPLCloneXMLTree(), CPLCreateXMLElementAndValue(), CPL↔SetXMLValue(), and OGR\_G\_ExportEnvelopeToGMLTree().

**12.7.4.8 CPLDestroyXMLNode()**

```
void CPLDestroyXMLNode (
    CPLXMLNode * psNode )
```

Destroy a tree.

This function frees resources associated with a **CPLXMLNode** (p. ??) and all its children nodes.

**Parameters**

<i>psNode</i>	the tree to free.
---------------	-------------------

References CPLFree, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by OGRSpatialReference::exportToXML(), and OGRSpatialReference::importFromXML().

**12.7.4.9 CPLGetXMLNode()**

```
CPLXMLNode* CPLGetXMLNode (
    CPLXMLNode * psRoot,
    const char * pszPath )
```

Find node by path.

Searches the document or subdocument indicated by psRoot for an element (or attribute) with the given path. The path should consist of a set of element names separated by dots, not including the name of the root element (psRoot). If the requested element is not found NULL is returned.

Attribute names may only appear as the last item in the path.

The search is done from the root nodes children, but all intermediate nodes in the path must be specified. Searching for "name" would only find a name element or attribute if it is a direct child of the root, not at any level in the subdocument.

If the pszPath is prefixed by "=" then the search will begin with the root node, and its siblings, instead of the root nodes children. This is particularly useful when searching within a whole document which is often prefixed by one or more "junk" nodes like the <?xml> declaration.

**Parameters**

<i>psRoot</i>	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszPath</i>	the list of element names in the path (dot separated).

**Returns**

the requested element node, or NULL if not found.

References CSLDestroy(), CSLTokenizeStringComplex(), CXT\_Text, EQUAL, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by CPLGetXMLValue().

**12.7.4.10 CPLGetXMLValue()**

```
const char* CPLGetXMLValue (
    const   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 no value then the passed default value is returned.

The returned value points to memory within the document tree, and should not be altered or freed.

**Parameters**

<i>psRoot</i>	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszPath</i>	the list of element names in the path (dot separated). An empty path means get the value of the psRoot node.
<i>pszDefault</i>	the value to return if a corresponding value is not found, may be NULL.

**Returns**

the requested value or pszDefault if not found.

References CPLAssert, CPLGetXMLNode(), CXT\_Attribute, CXT\_Element, CXT\_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

#### 12.7.4.11 CPLParseXMLFile()

```
CPLXMLNode* CPLParseXMLFile (
    const char * pszFilename )
```

Parse XML file into tree.

The named file is opened, loaded into memory as a big string, and parsed with **CPLParseXMLString()** (p. ??). Errors in reading the file or parsing the XML will be reported by **CPLError()** (p. ??).

The "large file" API is used, so XML files can come from virtualized files.

##### Parameters

<i>pszFilename</i>	the file to open.
--------------------	-------------------

##### Returns

NULL on failure, or the document tree on success.

References CPLFree, CPLParseXMLString(), and VSIIngestFile().

#### 12.7.4.12 CPLParseXMLString()

```
CPLXMLNode* CPLParseXMLString (
    const char * pszString )
```

Parse an XML string into tree form.

The passed document is parsed into a **CPLXMLNode** (p. ??) tree representation. If the document is not well formed XML then NULL is returned, and errors are reported via **CPLError()** (p. ??). No validation beyond wellformedness is done. The **CPLParseXMLFile()** (p. ??) convenience function can be used to parse from a file.

The returned document tree is owned by the caller and should be freed with **CPLDestroyXMLNode()** (p. ??) when no longer needed.

If the document has more than one "root level" element then those after the first will be attached to the first as siblings (via the psNext pointers) even though there is no common parent. A document with no XML structure (no angle brackets for instance) would be considered well formed, and returned as a single CXT\_Text node.

##### Parameters

<i>pszString</i>	the document to parse.
------------------	------------------------

##### Returns

parsed tree or NULL on error.

References CPLError().

Referenced by CPLParseXMLFile(), OGRSpatialReference::importFromXML(), and OGR\_G\_ExportToGMLTree().

#### 12.7.4.13 CPLRemoveXMLChild()

```
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 its own children, but is cleanly removed from the child list of the parent.

##### Parameters

<i>psParent</i>	the node to the child is attached to.
<i>psChild</i>	the child to remove.

##### Returns

TRUE on success or FALSE if the child was not found.

References CPLXMLNode::psChild, and CPLXMLNode::psNext.

#### 12.7.4.14 CPLSearchXMLNode()

```
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 its first child, that child's first child, and so on.

Use **CPLGetXMLNode()** (p. ??) to find a specific child, or along a specific node path.

##### Parameters

<i>psRoot</i>	the subtree to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszElement</i>	the name of the element or attribute to search for.

**Returns**

The matching node or NULL on failure.

References CXT\_Attribute, CXT\_Element, EQUAL, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

**12.7.4.15 CPLSerializeXMLTree()**

```
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()** (p. ??) when no longer needed.

**Parameters**

<i>psNode</i>	the node to serialize.
---------------	------------------------

**Returns**

the document on success or NULL on failure.

References CPLCalloc(), CPLXMLNode::psNext, and VSIFree().

Referenced by CPLSerializeXMLTreeToFile(), and OGRSpatialReference::exportToXML().

**12.7.4.16 CPLSerializeXMLTreeToFile()**

```
int CPLSerializeXMLTreeToFile (
    const CPLXMLNode * psTree,
    const char * pszFilename )
```

Write document tree to a file.

The passed document tree is converted into one big string (with **CPLSerializeXMLTree()** (p. ??)) and then written to the named file. Errors writing the file will be reported by **CPLError()** (p. ??). The source document tree is not altered. If the output file already exists it will be overwritten.

**Parameters**

<i>psTree</i>	the document tree to write.
<i>pszFilename</i>	the name of the file to write to.

**Returns**

TRUE on success, FALSE otherwise.

References CPLError(), CPLSerializeXMLTree(), and VSIFOpenL().

**12.7.4.17 CPLSetXMLValue()**

```
int CPLSetXMLValue (
    CPLXMLNode * psRoot,
    const char * pszPath,
    const char * pszValue )
```

Set element value by path.

Find (or create) the target element or attribute specified in the path, and assign it the indicated value.

Any path elements that do not already exist will be created. The target nodes value (the first CXT\_Text child) will be replaced with the provided value.

If the target node is an attribute instead of an element, the name should be prefixed with a #.

Example: CPLSetXMLValue( "Citation.Id.Description", "DOQ dataset" ); CPLSetXMLValue( "Citation.Id.↵  
Description.#name", "doq" );

**Parameters**

<i>psRoot</i>	the subdocument to be updated.
<i>pszPath</i>	the dot separated path to the target element/attribute.
<i>pszValue</i>	the text value to assign.

**Returns**

TRUE on success.

References CPLCreateXMLNode(), CPLFree, CPLStrdup(), CSLDestroy(), CSLTokenizeStringComplex(), CXT\_↵  
Attribute, CXT\_Element, CXT\_Text, EQUAL, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::ps↵  
Next, and CPLXMLNode::pszValue.

**12.7.4.18 CPLStripXMLNamespace()**

```
void CPLStripXMLNamespace (
    CPLXMLNode * psRoot,
    const char * pszNamespace,
    int bRecurse )
```

Strip indicated namespaces.

The subdocument (psRoot) is recursively examined, and any elements with the indicated namespace prefix will have the namespace prefix stripped from the element names. If the passed namespace is NULL, then all namespace prefixes will be stripped.

Nodes other than elements should remain unaffected. The changes are made "in place", and should not alter any node locations, only the pszValue field of affected nodes.

## Parameters

<i>psRoot</i>	the document to operate on.
<i>pszNamespace</i>	the name space prefix (not including colon), or NULL.
<i>bRecurse</i>	TRUE to recurse over whole document, or FALSE to only operate on the passed node.

References CXT\_Attribute, CXT\_Element, EQUALN, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by OGRSpatialReference::importFromXML().

## 12.8 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**

### 12.8.1 Detailed Description

ODBC Abstraction Layer (C++).

## 12.9 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 **GINTBIG\_MIN** (CPL\_STATIC\_CAST( **GIntBig**, 0x80000000) << 32)
- #define **GINTBIG\_MAX** (((CPL\_STATIC\_CAST( **GIntBig**, 0x7FFFFFFF) << 32) | 0xFFFFFFFFFU)
- #define **GUINBIG\_MAX** ((CPL\_STATIC\_CAST( **GUIntBig**, 0xFFFFFFFFFU) << 32) | 0xFFFFFFFFFU)
- #define **GINT64\_MIN** **GINTBIG\_MIN**
- #define **GINT64\_MAX** **GINTBIG\_MAX**
- #define **GUIN64\_MAX** **GUINBIG\_MAX**
- #define **CPL\_FRMT\_GB\_WITHOUT\_PREFIX** "l"
- #define **CPL\_FRMT\_GIB** "%" **CPL\_FRMT\_GB\_WITHOUT\_PREFIX** "d"
- #define **CPL\_FRMT\_GUIB** "%" **CPL\_FRMT\_GB\_WITHOUT\_PREFIX** "u"
- #define **CPL\_C\_START** extern "C" {
- #define **CPL\_C\_END** }
- #define **MIN**(a, b) (((a)<(b)) ? (a) : (b))
- #define **MAX**(a, b) (((a)>(b)) ? (a) : (b))
- #define **ABS**(x) (((x)<0) ? (-1\*(x)) : (x))
- #define **M\_PI** 3.14159265358979323846
- #define **STRCASECMP**(a, b) (strcasecmp(a,b))
- #define **STRNCASECMP**(a, b, n) (strncasecmp(a,b,n))
- #define **EQUALN**(a, b, n) ( **STRNCASECMP**(a,b,n)==0)
- #define **EQUAL**(a, b) ( **STRCASECMP**(a,b)==0)
- #define **STARTS\_WITH**(a, b) (strncmp(a,b,strlen(b)) == 0)
- #define **STARTS\_WITH\_CI**(a, b) **EQUALN**(a,b,strlen(b))
- #define **CPL\_SWAP16**(x) CPL\_STATIC\_CAST( **GUInt16**, (CPL\_STATIC\_CAST( **GUInt16**, x) << 8) | (CPL\_STATIC\_CAST( **GUInt16**, x) >> 8) )
- #define **CPL\_SWAP32**(x)
- #define **CPL\_SWAP64**(x)
- #define **CPL\_SWAP16PTR**(x)
- #define **CPL\_SWAP32PTR**(x)
- #define **CPL\_SWAP64PTR**(x)
- #define **CPL\_SWAPDOUBLE**(p) **CPL\_SWAP64PTR**(p)
- #define **CPL\_LSBWORD16**(x) (x)
- #define **CPL\_MSBWORD16**(x) **CPL\_SWAP16**(x)
- #define **CPL\_LSBWORD32**(x) (x)
- #define **CPL\_MSBWORD32**(x) **CPL\_SWAP32**(x)
- #define **CPL\_LSBPTR16**(x) CPL\_STATIC\_ASSERT\_IF\_AVAILABLE(sizeof(\*(x)) == 1 || sizeof(\*(x)) == 2)
- #define **CPL\_MSBPTR16**(x) **CPL\_SWAP16PTR**(x)
- #define **CPL\_LSBPTR32**(x) CPL\_STATIC\_ASSERT\_IF\_AVAILABLE(sizeof(\*(x)) == 1 || sizeof(\*(x)) == 4)
- #define **CPL\_MSBPTR32**(x) **CPL\_SWAP32PTR**(x)
- #define **CPL\_LSBPTR64**(x) CPL\_STATIC\_ASSERT\_IF\_AVAILABLE(sizeof(\*(x)) == 1 || sizeof(\*(x)) == 8)
- #define **CPL\_MSBPTR64**(x) **CPL\_SWAP64PTR**(x)
- #define **CPL\_LSBINT16PTR**(x) ((\*CPL\_REINTERPRET\_CAST(const **GByte\***, x)) | ((\*CPL\_REINTERPRET\_CAST(const **GByte\***, x))+1) << 8))
- #define **CPL\_LSBINT32PTR**(x)
- #define **CPL\_LSBSINT16PTR**(x) CPL\_STATIC\_CAST( **GInt16**, **CPL\_LSBINT16PTR**(x))
- #define **CPL\_LSBUINT16PTR**(x) CPL\_STATIC\_CAST( **GUInt16**, **CPL\_LSBINT16PTR**(x))
- #define **CPL\_LSBSINT32PTR**(x) CPL\_STATIC\_CAST( **GInt32**, **CPL\_LSBINT32PTR**(x))
- #define **CPL\_LSBUINT32PTR**(x) CPL\_STATIC\_CAST( **GUInt32**, **CPL\_LSBINT32PTR**(x))
- #define **CPL\_NULL\_TERMINATED**
- #define **CPL\_PRINT\_FUNC\_FORMAT**(format\_idx, arg\_idx)
- #define **CPL\_SCAN\_FUNC\_FORMAT**(format\_idx, arg\_idx)
- #define **CPL\_FORMAT\_STRING**(arg) arg
- #define **CPL\_SCANF\_FORMAT\_STRING**(arg) arg
- #define **CPL\_WARN\_UNUSED\_RESULT**
- #define **CPL\_UNUSED**

- `#define CPL_NO_RETURN`
- `#define CPL_RETURNS_NONNULL`
- `#define CPL_RESTRICT`
- `#define CPL_OVERRIDE override`
- `#define CPL_FINAL final`
- `#define CPL_DISALLOW_COPY_ASSIGN(className)`
- `#define CPL_ARRAYSIZE(array)`
- `#define CPL_FALLTHROUGH`

## Typedefs

- `typedef int GInt32`
- `typedef unsigned int GUInt32`
- `typedef short GInt16`
- `typedef unsigned short GUInt16`
- `typedef unsigned char GByte`
- `typedef int GBool`
- `typedef long long GIntBig`
- `typedef unsigned long long GUIntBig`
- `typedef GIntBig GInt64`
- `typedef GUIntBig GUInt64`
- `typedef GIntBig GPtrDiff_t`
- `typedef char ** CSLConstList`

### 12.9.1 Detailed Description

Core portability definitions for CPL.

### 12.9.2 Macro Definition Documentation

#### 12.9.2.1 ABS

```
#define ABS(  
    x ) ((x)<0) ? (-1*(x)) : (x)
```

Macro to compute the absolute value

#### 12.9.2.2 CPL\_ARRAYSIZE

```
#define CPL_ARRAYSIZE(  
    array )
```

**Value:**

```
((sizeof(array) / sizeof(*(array))) / \  
 static_cast<size_t>(!(sizeof(array) % sizeof(*(array)))))
```

Returns the size of C style arrays.

Referenced by `OGRSpatialReference::morphFromESRI()`, and `VSI curlClearCache()`.

### 12.9.2.3 CPL\_C\_END

```
#define CPL_C_END }
```

Macro to end a block of C symbols

### 12.9.2.4 CPL\_C\_START

```
#define CPL_C_START extern "C" {
```

Macro to start a block of C symbols

### 12.9.2.5 CPL\_DISALLOW\_COPY\_ASSIGN

```
#define CPL_DISALLOW_COPY_ASSIGN(  
    ClassName )
```

**Value:**

```
ClassName( const ClassName & ) = delete; \  
    ClassName &operator=( const ClassName & ) = delete;
```

Helper to remove the copy and assignment constructors so that the compiler will not generate the default versions.

Must be placed in the private section of a class and should be at the end.

### 12.9.2.6 CPL\_FALLTHROUGH

```
#define CPL_FALLTHROUGH
```

Macro for fallthrough in a switch case construct

### 12.9.2.7 CPL\_FINAL

```
#define CPL_FINAL final
```

C++11 final qualifier

### 12.9.2.8 CPL\_FORMAT\_STRING

```
#define CPL_FORMAT_STRING(  
    arg ) arg
```

Macro into which to wrap the format argument of a printf-like function

### 12.9.2.9 CPL\_FRMT\_GB\_WITHOUT\_PREFIX

```
#define CPL_FRMT_GB_WITHOUT_PREFIX "ll"
```

Printf formatting suffix for GIntBig

### 12.9.2.10 CPL\_FRMT\_GIB

```
#define CPL_FRMT_GIB "%" CPL_FRMT_GB_WITHOUT_PREFIX "d"
```

Printf formatting for GIntBig

Referenced by OGRFeature::DumpReadable(), OGRFeature::GetFieldAsString(), and VSIRealloc().

### 12.9.2.11 CPL\_FRMT\_GUIB

```
#define CPL_FRMT_GUIB "%" CPL_FRMT_GB_WITHOUT_PREFIX "u"
```

Printf formatting for GUIntBig

Referenced by VSIRealloc().

### 12.9.2.12 CPL\_LSBINT16PTR

```
#define CPL_LSBINT16PTR(  
    x ) ((CPL_REINTERPRET_CAST(const GByte*, x)) | ((*CPL_REINTERPRET_CAST(const  
GByte*, x))+1) << 8))
```

Return a Int16 from the 2 bytes ordered in LSB order at address x.

**Deprecated** Use rather CPL\_LSBSINT16PTR or CPL\_LSBUINT16PTR for explicit signedness.

### 12.9.2.13 CPL\_LSBINT32PTR

```
#define CPL_LSBINT32PTR(  
    x )
```

**Value:**

```
((CPL_REINTERPRET_CAST(const GByte*, x)) | ((*CPL_REINTERPRET_CAST(const GByte*, x))+1) << 8) | \  
    ((*CPL_REINTERPRET_CAST(const GByte*, x))+2) << 16) | ((*CPL_REINTERPRET_CAST(const GByte*, x))+3) << 24))
```

Return a Int32 from the 4 bytes ordered in LSB order at address x.

**Deprecated** Use rather CPL\_LSBSINT32PTR or CPL\_LSBUINT32PTR for explicit signedness.

#### 12.9.2.14 CPL\_LSBPTR16

```
#define CPL_LSBPTR16(  
    x ) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 2)
```

Byte-swap if necessary a 16bit word at the location pointed from a originally LSB ordered pointer

#### 12.9.2.15 CPL\_LSBPTR32

```
#define CPL_LSBPTR32(  
    x ) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 4)
```

Byte-swap if necessary a 32bit word at the location pointed from a originally LSB ordered pointer

Referenced by `OGRPoint::exportToWkb()`, and `OGRSimpleCurve::exportToWkb()`.

#### 12.9.2.16 CPL\_LSBPTR64

```
#define CPL_LSBPTR64(  
    x ) CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*(x)) == 1 || sizeof(*(x)) == 8)
```

Byte-swap if necessary a 64bit word at the location pointed from a originally LSB ordered pointer

#### 12.9.2.17 CPL\_LSBSINT16PTR

```
#define CPL_LSBSINT16PTR(  
    x ) CPL_STATIC_CAST( GInt16, CPL_LSBINT16PTR(x) )
```

Return a signed Int16 from the 2 bytes ordered in LSB order at address x

#### 12.9.2.18 CPL\_LSBSINT32PTR

```
#define CPL_LSBSINT32PTR(  
    x ) CPL_STATIC_CAST( GInt32, CPL_LSBINT32PTR(x) )
```

Return a signed Int32 from the 4 bytes ordered in LSB order at address x

#### 12.9.2.19 CPL\_LSBUINT16PTR

```
#define CPL_LSBUINT16PTR(  
    x ) CPL_STATIC_CAST( GUInt16, CPL_LSBINT16PTR(x) )
```

Return a unsigned Int16 from the 2 bytes ordered in LSB order at address x

**12.9.2.20 CPL\_LSBUINT32PTR**

```
#define CPL_LSBUINT32PTR(  
    x ) CPL_STATIC_CAST( GUInt32,  CPL_LSBINT32PTR(x) )
```

Return a unsigned Int32 from the 4 bytes ordered in LSB order at address x

**12.9.2.21 CPL\_LSBWORD16**

```
#define CPL_LSBWORD16(  
    x ) (x)
```

Return a 16bit word from a originally LSB ordered word

**12.9.2.22 CPL\_LSBWORD32**

```
#define CPL_LSBWORD32(  
    x ) (x)
```

Return a 32bit word from a originally LSB ordered word

**12.9.2.23 CPL\_MSBPTR16**

```
#define CPL_MSBPTR16(  
    x )  CPL_SWAP16PTR(x)
```

Byte-swap if necessary a 16bit word at the location pointed from a originally MSB ordered pointer

**12.9.2.24 CPL\_MSBPTR32**

```
#define CPL_MSBPTR32(  
    x )  CPL_SWAP32PTR(x)
```

Byte-swap if necessary a 32bit word at the location pointed from a originally MSB ordered pointer

Referenced by OGRPoint::exportToWkb(), and OGRSimpleCurve::exportToWkb().

**12.9.2.25 CPL\_MSBPTR64**

```
#define CPL_MSBPTR64(  
    x )  CPL_SWAP64PTR(x)
```

Byte-swap if necessary a 64bit word at the location pointed from a originally MSB ordered pointer

#### 12.9.2.26 CPL\_MSBWORD16

```
#define CPL_MSBWORD16(  
    x )    CPL_SWAP16(x)
```

Return a 16bit word from a originally MSB ordered word

#### 12.9.2.27 CPL\_MSBWORD32

```
#define CPL_MSBWORD32(  
    x )    CPL_SWAP32(x)
```

Return a 32bit word from a originally MSB ordered word

#### 12.9.2.28 CPL\_NO\_RETURN

```
#define CPL_NO_RETURN
```

Qualifier for a function that does not return at all (terminates the process)

#### 12.9.2.29 CPL\_NULL\_TERMINATED

```
#define CPL_NULL_TERMINATED
```

Null terminated variadic

#### 12.9.2.30 CPL\_OVERRIDE

```
#define CPL_OVERRIDE override
```

To be used in public headers only. For non-public headers or .cpp files, use override directly.

#### 12.9.2.31 CPL\_PRINT\_FUNC\_FORMAT

```
#define CPL_PRINT_FUNC_FORMAT(  
    format_idx,  
    arg_idx )
```

Tag a function to have printf() formatting

#### 12.9.2.32 CPL\_RESTRICT

```
#define CPL_RESTRICT
```

restrict keyword to declare that pointers do not alias

### 12.9.2.33 CPL\_RETURNS\_NONNULL

```
#define CPL_RETURNS_NONNULL
```

Qualifier for a function that does not return NULL

### 12.9.2.34 CPL\_SCAN\_FUNC\_FORMAT

```
#define CPL_SCAN_FUNC_FORMAT(  
    format_idx,  
    arg_idx )
```

Tag a function to have scanf() formatting

### 12.9.2.35 CPL\_SCANF\_FORMAT\_STRING

```
#define CPL_SCANF_FORMAT_STRING(  
    arg ) arg
```

Macro into which to wrap the format argument of a sscanf-like function.

### 12.9.2.36 CPL\_SWAP16

```
#define CPL_SWAP16(  
    x ) CPL_STATIC_CAST( GUInt16, (CPL_STATIC_CAST( GUInt16, x) << 8) | (CPL_STATIC_CAST( GUInt16, x) >> 8) )
```

Byte-swap a 16bit unsigned integer

### 12.9.2.37 CPL\_SWAP16PTR

```
#define CPL_SWAP16PTR(  
    x )
```

**Value:**

```
{  
    GByte      byTemp, *_pabyDataT = CPL_REINTERPRET_CAST(GByte*, x);  
    CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*x) == 1 || sizeof(*x) == 2);  
  
    byTemp = *_pabyDataT;  
    *_pabyDataT = *_pabyDataT + 1;  
    *_pabyDataT = byTemp;  
}
```

Byte-swap a 16 bit pointer



## 12.9.2.38 CPL\_SWAP32

```
#define CPL_SWAP32(
    x )
```

**Value:**

```
CPL_STATIC_CAST(GUInt32, \
    ((CPL_STATIC_CAST(GUInt32, x) & 0x000000ffU) << 24) | \
    ((CPL_STATIC_CAST(GUInt32, x) & 0x0000ff00U) << 8) | \
    ((CPL_STATIC_CAST(GUInt32, x) & 0x00ff0000U) >> 8) | \
    ((CPL_STATIC_CAST(GUInt32, x) & 0xff000000U) >> 24) )
```

Byte-swap a 32bit unsigned integer

Referenced by `OGRSimpleCurve::exportToWkb()`, `OGRPolygon::exportToWkb()`, `OGRGeometryCollection::exportToWkb()`, and `OGRPolyhedralSurface::exportToWkb()`.

## 12.9.2.39 CPL\_SWAP32PTR

```
#define CPL_SWAP32PTR(
    x )
```

**Value:**

```
{
    GByte      byTemp, *_pabyDataT = CPL_REINTERPRET_CAST(GByte*, x); \
    CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*x) == 1 || sizeof(*x) == 4); \
    \
    byTemp = _pabyDataT[0]; \
    _pabyDataT[0] = _pabyDataT[3]; \
    _pabyDataT[3] = byTemp; \
    byTemp = _pabyDataT[1]; \
    _pabyDataT[1] = _pabyDataT[2]; \
    _pabyDataT[2] = byTemp; \
}
```

Byte-swap a 32 bit pointer

## 12.9.2.40 CPL\_SWAP64

```
#define CPL_SWAP64(
    x )
```

**Value:**

```
((CPL_STATIC_CAST(GUInt64, CPL_SWAP32(CPL_STATIC_CAST(GUInt32, x))) << 32) | \
    (CPL_STATIC_CAST(GUInt64, CPL_SWAP32(CPL_STATIC_CAST(GUInt32, CPL_STATIC_CAST(
    GUInt64, x) >> 32)))))
```

Byte-swap a 64bit unsigned integer

#### 12.9.2.41 CPL\_SWAP64PTR

```
#define CPL_SWAP64PTR(
    x )
```

**Value:**

```
{
    GByte      byTemp, *_pabyDataT = CPL_REINTERPRET_CAST(GByte*, x);
    CPL_STATIC_ASSERT_IF_AVAILABLE(sizeof(*x) == 1 || sizeof(*x) == 8); \
    byTemp = _pabyDataT[0];
    _pabyDataT[0] = _pabyDataT[7];
    _pabyDataT[7] = byTemp;
    byTemp = _pabyDataT[1];
    _pabyDataT[1] = _pabyDataT[6];
    _pabyDataT[6] = byTemp;
    byTemp = _pabyDataT[2];
    _pabyDataT[2] = _pabyDataT[5];
    _pabyDataT[5] = byTemp;
    byTemp = _pabyDataT[3];
    _pabyDataT[3] = _pabyDataT[4];
    _pabyDataT[4] = byTemp;
}
```

Byte-swap a 64 bit pointer

Referenced by `OGRSimpleCurve::exportToWkb()`.

#### 12.9.2.42 CPL\_SWAPDOUBLE

```
#define CPL_SWAPDOUBLE(
    p )    CPL_SWAP64PTR(p)
```

Byte-swap a 64 bit pointer

Referenced by `OGRPoint::exportToWkb()`, and `OGRPoint::importFromWkb()`.

#### 12.9.2.43 CPL\_UNUSED

```
#define CPL_UNUSED
```

Qualifier for an argument that is unused

Referenced by `CPLODBCDriverInstaller::InstallDriver()`.

#### 12.9.2.44 CPL\_WARN\_UNUSED\_RESULT

```
#define CPL_WARN_UNUSED_RESULT
```

Qualifier to warn when the return value of a function is not used

## 12.9.2.45 EQUAL

```
#define EQUAL(
    a,
    b ) ( STRCASECMP (a,b) ==0)
```

Alias for strcasecmp() == 0

Referenced by OGRTriangulatedSurface::addGeometry(), OGR\_SRSNode::applyRemapper(), OGRSpatialReference::AutoidentifyEPSG(), OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::CopyGeogCSFrom(), CPLCheckForFile(), CPLCorrespondingPaths(), CPLDebug(), CPLEncodingCharSize(), CPLExtractRelativePath(), CPLGetXMLNode(), CPLPrintTime(), CPLRecode(), CPLRecodeFromWChar(), CPLRecodeToWChar(), CPLSearchXMLNode(), CPLSetXMLValue(), CPLTestBool(), CSLFindString(), OGRGeometryFactory::curveToLineString(), OGRGeometry::dumpReadable(), OGRFeature::DumpReadable(), OGRSpatialReference::EPSGTreatsAsLatLong(), OGRSpatialReference::EPSGTreatsAsNorthingEasting(), OGRSpatialReference::exportToERM(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGR\_SRSNode::FindChild(), OGRSpatialReference::FindProjParm(), OGR\_SRSNode::FixupOrdering(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAxis(), CPLDBCStatement::GetColId(), OGRSpatialReference::GetEPSGGeogCS(), OGRSpatialReference::GetExtension(), OGRFeatureDefn::GetFieldIndex(), OGRFeatureDefn::GetGeomFieldIndex(), OGR\_SRSNode::GetNode(), OGRStyleTable::GetStyleName(), OGRSpatialReference::GetTargetLinearUnits(), OGRSpatialReference::GetUTMZone(), OGRSpatialReference::importFromERM(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::ImportFromESRIStatePlaneWKT(), OGRSpatialReference::ImportFromESRIWisconsinWKT(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRMultiSurface::importFromWkt(), OGRPolyhedralSurface::importFromWkt(), OGRMultiPoint::importFromWkt(), OGRSpatialReference::importFromXML(), OGRSpatialReference::IsAngularParameter(), OGRSpatialReference::IsCompound(), OGRFieldDefn::IsDefaultDriverSpecific(), OGRSpatialReference::IsGeocentric(), OGRSpatialReference::IsGeographic(), OGRSpatialReference::IsLinearParameter(), OGRSpatialReference::IsLocal(), OGRSpatialReference::IsLongitudeParameter(), OGRSpatialReference::IsProjected(), OGRSpatialReference::IsSame(), OGRSpatialReference::IsSameGeogCS(), OGRSpatialReference::IsSameVertCS(), OGRSpatialReference::IsVertical(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGR\_G\_ExportToGMLEx(), OGRGeocodeCreateSession(), OPTGetParameterList(), OPTGetProjectionMethods(), OGRGeometryFactory::organizePolygons(), OGRCurvePolygon::segmentize(), GOA2Manager::SetAuthFromServiceAccount(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetGeogCS(), OGRLayer::SetIgnoredFields(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetVertCS(), OGRSpatialReference::SetWellKnownGeogCS(), OGRSpatialReference::StripCTParms(), OGRSpatialReference::StripVertical(), and VSIRReadDirRecursive().

## 12.9.2.46 EQUALN

```
#define EQUALN(
    a,
    b,
    n ) ( STRNCASECMP (a,b,n) ==0)
```

Alias for strncasecmp() == 0

Referenced by CPLCorrespondingPaths(), CPLDebug(), CPLExtractRelativePath(), CPLStripXMLNamespace(), CSLFetchNameValue(), CSLFetchNameValueMultiple(), CSLFindName(), CSLSetNameValue(), CPLStringList::FindName(), CPLString::ifind(), OGRSpatialReference::importFromDict(), and OGRSpatialReference::importFromPCI().

**12.9.2.47 GINT64\_MAX**

```
#define GINT64_MAX GINTBIG_MAX
```

Maximum GInt64 value

**12.9.2.48 GINT64\_MIN**

```
#define GINT64_MIN GINTBIG_MIN
```

Minimum GInt64 value

**12.9.2.49 GINTBIG\_MAX**

```
#define GINTBIG_MAX ((CPL_STATIC_CAST( GIntBig, 0x7FFFFFFF) << 32) | 0xFFFFFFFFU)
```

Maximum GIntBig value

**12.9.2.50 GINTBIG\_MIN**

```
#define GINTBIG_MIN (CPL_STATIC_CAST( GIntBig, 0x80000000) << 32)
```

Minimum GIntBig value

**12.9.2.51 GUINT64\_MAX**

```
#define GUINT64_MAX GUINTBIG_MAX
```

Minimum GUInt64 value

**12.9.2.52 GUINTBIG\_MAX**

```
#define GUINTBIG_MAX ((CPL_STATIC_CAST( GUIntBig, 0xFFFFFFFFU) << 32) | 0xFFFFFFFFU)
```

Maximum GUIntBig value

**12.9.2.53 M\_PI**

```
#define M_PI 3.14159265358979323846
```

PI definition

Referenced by `OGRGeometryFactory::approximateArcAngles()`, `OGRSpatialReference::convertToOtherProjection()`, `OGRGeometryFactory::curveToLineString()`, and `OGRCircularString::get_Area()`.

**12.9.2.54 MAX**

```
#define MAX(
    a,
    b ) ((a)>(b)) ? (a) : (b)
```

Macro to compute the maximum of 2 values

Referenced by OGRFieldDefn::SetWidth().

**12.9.2.55 MIN**

```
#define MIN(
    a,
    b ) ((a)<(b)) ? (a) : (b)
```

Macro to compute the minimum of 2 values

**12.9.2.56 STARTS\_WITH**

```
#define STARTS_WITH(
    a,
    b ) (strcmp(a,b,strlen(b)) == 0)
```

Returns whether a starts with b

Referenced by CPLHTTPFetchEx(), CPLIsFilenameRelative(), and CPLStrtodDelim().

**12.9.2.57 STARTS\_WITH\_CI**

```
#define STARTS_WITH_CI(
    a,
    b ) EQUALN(a,b,strlen(b))
```

Returns whether a starts with b (case insensitive comparison)

Referenced by CPLEncodingCharSize(), CPLEExpandTilde(), CPLPrintPointer(), CPLScanPointer(), CPLStrtodDelim(), CPLUnescapeString(), OGRGeometryFactory::createFromWkt(), OGRSpatialReference::exportToPCI(), OGRPolygon::exportToWkt(), OGRFeature::FillUnsetWithDefault(), OGRSpatialReference::importFromCRSURL(), OGRSpatialReference::importFromDict(), OGRSpatialReference::importFromERM(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::ImportFromESRIWisconsinWKT(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromUrl(), OGRSpatialReference::importFromURN(), OGRMultiSurface::importFromWkt(), OGRSpatialReference::importFromWMSAUTO(), OGRSpatialReference::IsAngularParameter(), OGRSpatialReference::IsLinearParameter(), OGRSpatialReference::IsLongitudeParameter(), OGRSpatialReference::IsSame(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRFromOGCGeomType(), OGRGeocodeCreateSession(), OGRSpatialReference::SetFromUserInput(), and OGRSpatialReference::SetWellKnownGeogCS().

### 12.9.2.58 STRCASECMP

```
#define STRCASECMP (
    a,
    b ) (strcasecmp(a,b))
```

Alias for strcasecmp()

### 12.9.2.59 STRNCASECMP

```
#define STRNCASECMP (
    a,
    b,
    n ) (strncasecmp(a,b,n))
```

Alias for strncasecmp()

## 12.9.3 Typedef Documentation

### 12.9.3.1 CSLConstList

```
typedef char** CSLConstList
```

Type of a constant null-terminated list of nul terminated strings. Seen as char\*\* from C and const char\* const\* from C++

### 12.9.3.2 GBool

```
typedef int GBool
```

Type for boolean values (alias to int)

### 12.9.3.3 GByte

```
typedef unsigned char GByte
```

Unsigned byte type

### 12.9.3.4 GInt16

```
typedef short GInt16
```

Int16 type

#### 12.9.3.5 GInt32

```
typedef int    GInt32
```

Int32 type

#### 12.9.3.6 GInt64

```
typedef GIntBig GInt64
```

Signed 64 bit integer type

#### 12.9.3.7 GIntBig

```
typedef long long GIntBig
```

Large signed integer type (generally 64-bit integer type). Use GInt64 when exactly 64 bit is needed

#### 12.9.3.8 GPtrDiff\_t

```
typedef GIntBig GPtrDiff_t
```

Integer type large enough to hold the difference between 2 addresses

#### 12.9.3.9 GUInt16

```
typedef unsigned short GUInt16
```

Unsigned int16 type

#### 12.9.3.10 GUInt32

```
typedef unsigned int    GUInt32
```

Unsigned int32 type

#### 12.9.3.11 GUInt64

```
typedef GUIntBig GUInt64
```

Unsigned 64 bit integer type

#### 12.9.3.12 GUIntBig

```
typedef unsigned long long GUIntBig
```

Large unsigned integer type (generally 64-bit unsigned integer type). Use GUInt64 when exactly 64 bit is needed

## 12.10 cpl\_quad\_tree.h File Reference

```
#include "cpl_port.h"
```

### Classes

- struct **CPLRectObj**

### Typedefs

- typedef struct **\_CPLQuadTree** **CPLQuadTree**
- typedef void(\* **CPLQuadTreeGetBoundsFunc**) (const void \*hFeature, **CPLRectObj** \*pBounds)
- typedef int(\* **CPLQuadTreeForeachFunc**) (void \*pElt, void \*pUserData)
- typedef void(\* **CPLQuadTreeDumpFeatureFunc**) (const void \*hFeature, int nIndentLevel, void \*pUserData)

### Functions

- **CPLQuadTree** \* **CPLQuadTreeCreate** (const **CPLRectObj** \*pGlobalBounds, **CPLQuadTreeGet↵**  
**BoundsFunc** 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)
- void **CPLQuadTreeDump** (const **CPLQuadTree** \*hQuadtree, **CPLQuadTreeDumpFeatureFunc** pfn↵  
DumpFeatureFunc, void \*pUserData)
- void **CPLQuadTreeGetStats** (const **CPLQuadTree** \*hQuadtree, int \*pnFeatureCount, int \*pnNodeCount,  
int \*pnMaxDepth, int \*pnMaxBucketCapacity)

#### 12.10.1 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

#### 12.10.2 Typedef Documentation



## 12.10.2.1 CPLQuadTree

```
typedef struct _CPLQuadTree CPLQuadTree
```

Opaque type for a quad tree

## 12.10.2.2 CPLQuadTreeDumpFeatureFunc

```
typedef void(* CPLQuadTreeDumpFeatureFunc) (const void *hFeature, int nIndentLevel, void *p←
UserData)
```

CPLQuadTreeDumpFeatureFunc

## 12.10.2.3 CPLQuadTreeForeachFunc

```
typedef int(* CPLQuadTreeForeachFunc) (void *pElt, void *pUserData)
```

CPLQuadTreeForeachFunc

## 12.10.2.4 CPLQuadTreeGetBoundsFunc

```
typedef void(* CPLQuadTreeGetBoundsFunc) (const void *hFeature, CPLRectObj *pBounds)
```

CPLQuadTreeGetBoundsFunc

## 12.10.3 Function Documentation

## 12.10.3.1 CPLQuadTreeCreate()

```
CPLQuadTree* CPLQuadTreeCreate (
    const CPLRectObj * pGlobalBounds,
    CPLQuadTreeGetBoundsFunc pfnGetBounds )
```

Create a new quadtree

## Parameters

<i>pGlobalBounds</i>	a pointer to the global extent of all the elements that will be inserted
<i>pfnGetBounds</i>	a user provided function to get the bounding box of the inserted elements. If it is set to NULL, then <b>CPLQuadTreeInsertWithBounds()</b> (p. ??) must be used, and extra memory will be used to keep features bounds in the quad tree.

**Returns**

a newly allocated quadtree

References CPLAssert, and CPLMalloc().

**12.10.3.2 CPLQuadTreeDestroy()**

```
void CPLQuadTreeDestroy (
    CPLQuadTree * hQuadTree )
```

Destroy a quadtree

**Parameters**

<i>hQuadTree</i>	the quad tree to destroy
------------------	--------------------------

References CPLAssert, and CPLFree.

**12.10.3.3 CPLQuadTreeDump()**

```
void CPLQuadTreeDump (
    const CPLQuadTree * hQuadTree,
    CPLQuadTreeDumpFeatureFunc pfnDumpFeatureFunc,
    void * pUserData )
```

Dump quad tree

**12.10.3.4 CPLQuadTreeForeach()**

```
void CPLQuadTreeForeach (
    const CPLQuadTree * hQuadTree,
    CPLQuadTreeForeachFunc pfnForeach,
    void * pUserData )
```

Walk through the quadtree and runs the provided function on all the elements

This function is provided with the user\_data argument of pfnForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the quadtree must *NOT* be modified during the walk.

**Parameters**

<i>hQuadTree</i>	the quad tree
<i>pfnForeach</i>	the function called on each element.
<i>pUserData</i>	the user data provided to the function.

References CPLAssert.

### 12.10.3.5 CPLQuadTreeGetAdvisedMaxDepth()

```
int CPLQuadTreeGetAdvisedMaxDepth (
    int nExpectedFeatures )
```

Returns the optimal depth of a quadtree to hold *nExpectedFeatures*

#### Parameters

<i>nExpectedFeatures</i>	the expected maximum number of elements to be inserted.
--------------------------	---

#### Returns

the optimal depth of a quadtree to hold *nExpectedFeatures*

References CPLDebug().

### 12.10.3.6 CPLQuadTreeGetStats()

```
void CPLQuadTreeGetStats (
    const CPLQuadTree * hQuadTree,
    int * pnFeatureCount,
    int * pnNodeCount,
    int * pnMaxDepth,
    int * pnMaxBucketCapacity )
```

Get stats

References CPLAssert.

### 12.10.3.7 CPLQuadTreeInsert()

```
void CPLQuadTreeInsert (
    CPLQuadTree * hQuadTree,
    void * hFeature )
```

Insert a feature into a quadtree

#### Parameters

<i>hQuadTree</i>	the quad tree
<i>hFeature</i>	the feature to insert

References CPLError().

#### 12.10.3.8 CPLQuadTreeInsertWithBounds()

```
void CPLQuadTreeInsertWithBounds (
    CPLQuadTree * hQuadTree,
    void * hFeature,
    const CPLRectObj * psBounds )
```

Insert a feature into a quadtree

##### Parameters

<i>hQuadTree</i>	the quad tree
<i>hFeature</i>	the feature to insert
<i>psBounds</i>	bounds of the feature

#### 12.10.3.9 CPLQuadTreeSearch()

```
void** CPLQuadTreeSearch (
    const CPLQuadTree * hQuadTree,
    const CPLRectObj * pAoi,
    int * pnFeatureCount )
```

Returns all the elements inserted whose bounding box intersects the provided area of interest

##### Parameters

<i>hQuadTree</i>	the quad tree
<i>pAoi</i>	the pointer to the area of interest
<i>pnFeatureCount</i>	the user data provided to the function.

##### Returns

an array of features that must be freed with CPLFree

References CPLAssert.

#### 12.10.3.10 CPLQuadTreeSetBucketCapacity()

```
void CPLQuadTreeSetBucketCapacity (
    CPLQuadTree * hQuadTree,
    int nBucketCapacity )
```

Set the maximum capacity of a node of a quadtree. The default value is 8. Note that the maximum capacity will only be honoured if the features inserted have a point geometry. Otherwise it may be exceeded.

## Parameters

<i>hQuadTree</i>	the quad tree
<i>nBucketCapacity</i>	the maximum capacity of a node of a quadtree

## 12.10.3.11 CPLQuadTreeSetMaxDepth()

```
void CPLQuadTreeSetMaxDepth (
    CPLQuadTree * hQuadTree,
    int nMaxDepth )
```

Set the maximum depth of a quadtree. By default, quad trees have no maximum depth, but a maximum bucket capacity.

## Parameters

<i>hQuadTree</i>	the quad tree
<i>nMaxDepth</i>	the maximum depth allowed

## 12.11 cpl\_string.h File Reference

```
#include "cpl_error.h"
#include "cpl_conv.h"
#include "cpl_vsi.h"
```

## Classes

- class **CPLString**  
*Convenient string class based on std::string.*
- class **CPLStringList**  
*String list class designed around our use of C "char\*" string lists.*

## Macros

- #define **CSLT\_HONOURSTRINGS** 0x0001
- #define **CSLT\_ALLOWEMPTYTOKENS** 0x0002
- #define **CSLT\_PRESERVEQUOTES** 0x0004
- #define **CSLT\_PRESERVEESCAPES** 0x0008
- #define **CSLT\_STRIPLEADSPACES** 0x0010
- #define **CSLT\_STRIPENDSPACES** 0x0020
- #define **CPLES\_BackslashQuotable** 0
- #define **CPLES\_XML** 1
- #define **CPLES\_URL** 2

- `#define CPLES_SQL 3`
- `#define CPLES_CSV 4`
- `#define CPLES_XML_BUT_QUOTES 5`
- `#define CPLES_CSV_FORCE_QUOTING 6`
- `#define CPL_ENC_LOCALE ""`
- `#define CPL_ENC_UTF8 "UTF-8"`
- `#define CPL_ENC_UTF16 "UTF-16"`
- `#define CPL_ENC_UCS2 "UCS-2"`
- `#define CPL_ENC_UCS4 "UCS-4"`
- `#define CPL_ENC_ASCII "ASCII"`
- `#define CPL_ENC_ISO8859_1 "ISO-8859-1"`

## Enumerations

- `enum CPLValueType { CPL_VALUE_STRING, CPL_VALUE_REAL, CPL_VALUE_INTEGER }`

## Functions

- `char ** CSLAddString (char **papszStrList, const char *pszNewString)`
- `char ** CSLAddStringMayFail (char **papszStrList, const char *pszNewString)`
- `int CSLCount ( CSLConstList papszStrList)`
- `const char * CSLGetField ( CSLConstList, int)`
- `void CSLDestroy (char **papszStrList)`
- `char ** CSLDuplicate ( CSLConstList papszStrList)`
- `char ** CSLMerge (char **papszOrig, CSLConstList papszOverride)`  
*Merge two lists.*
- `char ** CSLTokenizeString (const char *pszString)`
- `char ** CSLTokenizeStringComplex (const char *pszString, const char *pszDelimiter, int bHonourStrings, int bAllowEmptyTokens)`
- `char ** CSLTokenizeString2 (const char *pszString, const char *pszDelimiter, int nCSLTFlags)`
- `int CSLPrint ( CSLConstList papszStrList, FILE *fpOut)`
- `char ** CSLLoad (const char *pszFname)`
- `char ** CSLLoad2 (const char *pszFname, int nMaxLines, int nMaxCols, CSLConstList papszOptions)`
- `int CSLSave ( CSLConstList papszStrList, const char *pszFname)`
- `char ** CSLInsertStrings (char **papszStrList, int nInsertAtLineNo, CSLConstList papszNewLines)`
- `char ** CSLInsertString (char **papszStrList, int nInsertAtLineNo, const char *pszNewLine)`
- `char ** CSLRemoveStrings (char **papszStrList, int nFirstLineToDelete, int nNumToRemove, char ***ppapszRetStrings)`
- `int CSLFindString ( CSLConstList papszList, const char *pszTarget)`
- `int CSLFindStringCaseSensitive ( CSLConstList papszList, const char *pszTarget)`
- `int CSLPartialFindString ( CSLConstList papszHaystack, const char *pszNeedle)`
- `int CSLFindName ( CSLConstList papszStrList, const char *pszName)`
- `int CSLFetchBoolean ( CSLConstList papszStrList, const char *pszKey, int bDefault)`
- `int CSLTestBoolean (const char *pszValue)`
- `int CPLTestBoolean (const char *pszValue)`
- `bool CPLTestBool (const char *pszValue)`
- `bool CPLFetchBool ( CSLConstList papszStrList, const char *pszKey, bool bDefault)`
- `const char * CPLParseNameValue (const char *pszNameValue, char **ppszKey)`
- `const char * CSLFetchNameValue ( CSLConstList papszStrList, const char *pszName)`
- `const char * CSLFetchNameValueDef ( CSLConstList papszStrList, const char *pszName, const char *pszDefault)`
- `char ** CSLFetchNameValueMultiple ( CSLConstList papszStrList, const char *pszName)`
- `char ** CSLAddNameValue (char **papszStrList, const char *pszName, const char *pszValue)`

- char \*\* **CSLSetNameValue** (char \*\*papszStrList, const char \*pszName, const char \*pszValue)
- void **CSLSetNameValueSeparator** (char \*\*papszStrList, const char \*pszSeparator)
- char \*\* **CSLParseCommandLine** (const char \*pszCommandLine)
- char \* **CPEscapeString** (const char \*pszString, int nLength, int nScheme)
- char \* **CPLUnescapeString** (const char \*pszString, int \*pnLength, int nScheme)
- char \* **CPLBinaryToHex** (int nBytes, const **GByte** \*pabyData)
- **GByte** \* **CPLHexToBinary** (const char \*pszHex, int \*pnBytes)
- char \* **CPLBase64Encode** (int nBytes, const **GByte** \*pabyData)
- int **CPLBase64DecodeInPlace** (**GByte** \*pszBase64)
- **CPLValueType** **CPLGetValueType** (const char \*pszValue)
- size\_t **CPLStrncpy** (char \*pszDest, const char \*pszSrc, size\_t nDestSize)
- size\_t **CPLStrcat** (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 **CPLprintf** (const char \*fmt,...)
- const char \* **CPLSPrintf** (const char \*fmt,...)
- char \*\* **CSLAppendPrintf** (char \*\*papszStrList, const char \*fmt,...)
- int **CPLVASPrintf** (char \*\*buf, const char \*fmt, va\_list args)
- int **CPLEncodingCharSize** (const char \*pszEncoding)
- char \* **CPLRecode** (const char \*pszSource, const char \*pszSrcEncoding, const char \*pszDstEncoding)
- char \* **CPLRecodeFromWChar** (const wchar\_t \*pwszSource, const char \*pszSrcEncoding, const char \*pszDstEncoding)
- wchar\_t \* **CPLRecodeToWChar** (const char \*pszSource, const char \*pszSrcEncoding, const char \*pszDstEncoding)
- int **CPLIsUTF8** (const char \*pabyData, int nLen)
- char \* **CPLForceToASCII** (const char \*pabyData, int nLen, char chReplacementChar)
- int **CPLStrlenUTF8** (const char \*pszUTF8Str)
- **CPLString** **CPLOPrintf** (const char \*pszFormat,...)
- **CPLString** **CPLovPrintf** (const char \*pszFormat, va\_list args)
- **CPLString** **CPLURLGetValue** (const char \*pszURL, const char \*pszKey)
- **CPLString** **CPLURLAddKVP** (const char \*pszURL, const char \*pszKey, const char \*pszValue)

### 12.11.1 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()** (p. ??)).

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.

### 12.11.2 Macro Definition Documentation

### 12.11.2.1 CPL\_ENC\_ASCII

```
#define CPL_ENC_ASCII "ASCII"
```

ASCII encoding

Referenced by `CPLEncodingCharSize()`, `CPLRecode()`, `CPLRecodeFromWChar()`, and `CPLRecodeToWChar()`.

### 12.11.2.2 CPL\_ENC\_ISO8859\_1

```
#define CPL_ENC_ISO8859_1 "ISO-8859-1"
```

ISO-8859-1 (LATIN1) encoding

Referenced by `CPLRecode()`, `CPLRecodeFromWChar()`, and `CPLRecodeToWChar()`.

### 12.11.2.3 CPL\_ENC\_LOCALE

```
#define CPL_ENC_LOCALE ""
```

Encoding of the current locale

### 12.11.2.4 CPL\_ENC\_UCS2

```
#define CPL_ENC_UCS2 "UCS-2"
```

UCS-2 encoding

Referenced by `CPLEncodingCharSize()`, `CPLRecodeFromWChar()`, and `CPLRecodeToWChar()`.

### 12.11.2.5 CPL\_ENC\_UCS4

```
#define CPL_ENC_UCS4 "UCS-4"
```

UCS-4 encoding

Referenced by `CPLEncodingCharSize()`.



### 12.11.2.6 CPL\_ENC\_UTF16

```
#define CPL_ENC_UTF16 "UTF-16"
```

UTF-16 encoding

Referenced by `CPLEncodingCharSize()`.

### 12.11.2.7 CPL\_ENC\_UTF8

```
#define CPL_ENC_UTF8 "UTF-8"
```

UTF-8 encoding

Referenced by `CPLEncodingCharSize()`, `CPLRecode()`, `CPLRecodeFromWChar()`, `CPLRecodeToWChar()`, `CPLUnescapeString()`, and `CPLString::Recode()`.

### 12.11.2.8 CPLES\_BackslashQuotable

```
#define CPLES_BackslashQuotable 0
```

Scheme for **CPLEscapeString()** (p. ??)/`CPLUnescapeString()` for backlash quoting

Referenced by `CPLEscapeString()`, and `CPLUnescapeString()`.

### 12.11.2.9 CPLES\_CSV

```
#define CPLES_CSV 4
```

Scheme for **CPLEscapeString()** (p. ??)/`CPLUnescapeString()` for CSV

Referenced by `CPLEscapeString()`, and `CPLUnescapeString()`.

### 12.11.2.10 CPLES\_CSV\_FORCE\_QUOTING

```
#define CPLES_CSV_FORCE_QUOTING 6
```

Scheme for **CPLEscapeString()** (p. ??)/`CPLUnescapeString()` for CSV (forced quoting)

Referenced by `CPLEscapeString()`.

#### 12.11.2.11 CPLES\_SQL

```
#define CPLES_SQL 3
```

Scheme for **CPLEscapeString()** (p. ??)/CPLUnescapeString() for SQL

Referenced by CPLEscapeString(), CPLUnescapeString(), and OGRFeature::FillUnsetWithDefault().

#### 12.11.2.12 CPLES\_URL

```
#define CPLES_URL 2
```

Scheme for **CPLEscapeString()** (p. ??)/CPLUnescapeString() for URL

Referenced by CPLEscapeString(), CPLUnescapeString(), and GOA2GetAuthorizationURL().

#### 12.11.2.13 CPLES\_XML

```
#define CPLES_XML 1
```

Scheme for **CPLEscapeString()** (p. ??)/CPLUnescapeString() for XML

Referenced by CPLEscapeString(), and CPLUnescapeString().

#### 12.11.2.14 CPLES\_XML\_BUT\_QUOTES

```
#define CPLES_XML_BUT_QUOTES 5
```

Scheme for **CPLEscapeString()** (p. ??)/CPLUnescapeString() for XML (preserves quotes)

Referenced by CPLEscapeString(), and CPLUnescapeString().

#### 12.11.2.15 CSLT\_ALLOWEMPTYTOKENS

```
#define CSLT_ALLOWEMPTYTOKENS 0x0002
```

Flag for **CSLTokenizeString2()** (p. ??) to allow empty tokens

Referenced by CSLTokenizeString2(), CSLTokenizeStringComplex(), and OGRSpatialReference::importFromOzi().

#### 12.11.2.16 CSLT\_HONOURSTRINGS

```
#define CSLT_HONOURSTRINGS 0x0001
```

Flag for **CSLTokenizeString2()** (p. ??) to honour strings

Referenced by **CSLTokenizeString()**, **CSLTokenizeString2()**, **CSLTokenizeStringComplex()**, and **OGRStyleMgr::GetPart()**.

#### 12.11.2.17 CSLT\_PRESERVEESCAPES

```
#define CSLT_PRESERVEESCAPES 0x0008
```

Flag for **CSLTokenizeString2()** (p. ??) to preserve escape characters

Referenced by **CSLTokenizeString2()**, and **OGRStyleMgr::GetPart()**.

#### 12.11.2.18 CSLT\_PRESERVEQUOTES

```
#define CSLT_PRESERVEQUOTES 0x0004
```

Flag for **CSLTokenizeString2()** (p. ??) to preserve quotes

Referenced by **CSLTokenizeString2()**, and **OGRStyleMgr::GetPart()**.

#### 12.11.2.19 CSLT\_STRIPENDSPACES

```
#define CSLT_STRIPENDSPACES 0x0020
```

Flag for **CSLTokenizeString2()** (p. ??) to strip trailing spaces

Referenced by **CSLTokenizeString2()**, and **OGRSpatialReference::importFromOzi()**.

#### 12.11.2.20 CSLT\_STRIPLEADSPACES

```
#define CSLT_STRIPLEADSPACES 0x0010
```

Flag for **CSLTokenizeString2()** (p. ??) to strip leading spaces

Referenced by **CSLTokenizeString2()**, and **OGRSpatialReference::importFromOzi()**.

### 12.11.3 Enumeration Type Documentation

#### 12.11.3.1 CPLValueType

```
enum CPLValueType
```

Type of value

## Enumerator

CPL_VALUE_STRING	String
CPL_VALUE_REAL	Real number
CPL_VALUE_INTEGER	Integer

## 12.11.4 Function Documentation

## 12.11.4.1 CPLBase64DecodeInPlace()

```
int CPLBase64DecodeInPlace (
    GByte * pszBase64 )
```

Decode base64 string "pszBase64" (null terminated) in place.

Returns length of decoded array or 0 on failure.

## 12.11.4.2 CPLBase64Encode()

```
char* CPLBase64Encode (
    int nDataLen,
    const GByte * pabyBytesToEncode )
```

Base64 encode a buffer.

References CPLStrdup().

## 12.11.4.3 CPLBinaryToHex()

```
char* CPLBinaryToHex (
    int nBytes,
    const GByte * pabyData )
```

Binary to hexadecimal translation.

## Parameters

<i>nBytes</i>	number of bytes of binary data in pabyData.
<i>pabyData</i>	array of data bytes to translate.

## Returns

hexadecimal translation, zero terminated. Free with **CPLFree()** (p. ??).

References CPLMalloc().

#### 12.11.4.4 CPLEncodingCharSize()

```
int CPLEncodingCharSize (
    const char * pszEncoding )
```

Return bytes per character for encoding.

This function returns the size in bytes of the smallest character in this encoding. For fixed width encodings (ASCII, UCS-2, UCS-4) this is straight forward. For encodings like UTF8 and UTF16 which represent some characters as a sequence of atomic character sizes the function still returns the atomic character size (1 for UTF8, 2 for UTF16).

This function will return the correct value for well known encodings with corresponding CPL\_ENC\_ values. It may not return the correct value for other encodings even if they are supported by the underlying iconv or windows transliteration services. Hopefully it will improve over time.

##### Parameters

<i>pszEncoding</i>	the name of the encoding.
--------------------	---------------------------

##### Returns

the size of a minimal character in bytes or -1 if the size is unknown.

References CPL\_ENC\_ASCII, CPL\_ENC\_UCS2, CPL\_ENC\_UCS4, CPL\_ENC\_UTF16, CPL\_ENC\_UTF8, EQU↵AL, and STARTS\_WITH\_CI.

#### 12.11.4.5 CPLEscapeString()

```
char* CPLEscapeString (
    const char * pszInput,
    int nLength,
    int nScheme )
```

Apply escaping to string to preserve special characters.

This function will "escape" a variety of special characters to make the string suitable to embed within a string constant or to write within a text stream but in a form that can be reconstituted to its original form. The escaping will even preserve zero bytes allowing preservation of raw binary data.

**CPLES\_BackslashQuotable(0)** (p. ??): 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)** (p. ??): This scheme converts the '<', '>', '"' and '&' characters into their XML/HTML equivalent (<, >, " and &) making a string safe to embed as CDATA within an XML element. The '\0' is not escaped and should not be included in the input.

**CPLES\_URL(2)** (p. ??): Everything except alphanumerics and the characters '\$', '-', '\_', ':', '+', '!', '\*', '"', '(', ')', and ';' (see RFC1738) 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)** (p. ??): 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)** (p. ??): If the values contains commas, semicolons, tabs, double quotes, or newlines it placed in double quotes, and double quotes in the value are doubled. Suitable for use when constructing field values for .csv files. Note that **CPLUnescapeString()** (p. ??) currently does not support this format, only **CPLEscapeString()** (p. ??). See cpl\_csv.cpp for CSV parsing support.

#### Parameters

<i>pszInput</i>	the string to escape.
<i>nLength</i>	The number of bytes of data to preserve. If this is -1 the strlen(pszString) function will be used to compute the length.
<i>nScheme</i>	the encoding scheme to use.

#### Returns

an escaped, zero terminated string that should be freed with **CPLFree()** (p. ??) when no longer needed.

References CPLError(), CPLES\_BackslashQuotable, CPLES\_CSV, CPLES\_CSV\_FORCE\_QUOTING, CPLES\_↵SQL, CPLES\_URL, CPLES\_XML, CPLES\_XML\_BUT\_QUOTES, and CPLMalloc().

Referenced by GOA2GetAuthorizationURL().

#### 12.11.4.6 CPLFetchBool()

```
bool CPLFetchBool (
    CSLConstList papszStrList,
    const char * pszKey,
    bool bDefault )
```

Check for boolean key value.

In a StringList of "Name=Value" pairs, look to see if there is a key with the given name, and if it can be interpreted as being TRUE. If the key appears without any "=Value" portion it will be considered true. If the value is NO, FALSE or 0 it will be considered FALSE otherwise if the key appears in the list it will be considered TRUE. If the key doesn't appear at all, the indicated default value will be returned.

#### Parameters

<i>papszStrList</i>	the string list to search.
<i>pszKey</i>	the key value to look for (case insensitive).
<i>bDefault</i>	the value to return if the key isn't found at all.

**Returns**

true or false

References CPLTestBool(), CSLFetchNameValue(), and CSLFindString().

Referenced by CSLFetchBoolean(), and CSLLoad2().

**12.11.4.7 CPLForceToASCII()**

```
char* CPLForceToASCII (
    const char * pabyData,
    int nLen,
    char chReplacementChar )
```

Return a new string that is made only of ASCII characters. If non-ASCII characters are found in the input string, they will be replaced by the provided replacement character.

**Parameters**

<i>pabyData</i>	input string to test
<i>nLen</i>	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.
<i>chReplacementChar</i>	character which will be used when the input stream contains a non ASCII character. Must be valid ASCII!

**Returns**

a new string that must be freed with **CPLFree()** (p. ??).

**Since**

GDAL 1.7.0

References CPLMalloc().

**12.11.4.8 CPLGetValueType()**

```
CPLValueType CPLGetValueType (
    const char * pszValue )
```

Detect the type of the value contained in a string, whether it is a real, an integer or a string. Leading and trailing spaces are skipped in the analysis.

Note: in the context of this function, integer must be understood in a broad sense. It does not mean that the value can fit into a 32 bit integer for example. It might be larger.

## Parameters

<i>pszValue</i>	the string to analyze
-----------------	-----------------------

## Returns

returns the type of the value contained in the string.

References CPL\_VALUE\_STRING, and CPLAtof().

## 12.11.4.9 CPLHexToBinary()

```
GByte* CPLHexToBinary (
    const char * pszHex,
    int * pnBytes )
```

Hexadecimal to binary translation

## Parameters

<i>pszHex</i>	the input hex encoded string.
<i>pnBytes</i>	the returned count of decoded bytes placed here.

## Returns

returns binary buffer of data - free with **CPLFree()** (p. ??).

References CPLMalloc().

## 12.11.4.10 CPLIsUTF8()

```
int CPLIsUTF8 (
    const char * pabyData,
    int nLen )
```

Test if a string is encoded as UTF-8.

## Parameters

<i>pabyData</i>	input string to test
<i>nLen</i>	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.



**Returns**

TRUE if the string is encoded as UTF-8. FALSE otherwise

**Since**

GDAL 1.7.0

**12.11.4.11 CPLPrintf()**

```
CPLString CPLPrintf (
    const char * pszFormat,
    ... )
```

Return a **CPLString** (p. ??) with the content of `sprintf()`

References `CPLString::vPrintf()`.

**12.11.4.12 CPLOvPrintf()**

```
CPLString CPLOvPrintf (
    const char * pszFormat,
    va_list args )
```

Return a **CPLString** (p. ??) with the content of `vsprintf()`

References `CPLString::vPrintf()`.

**12.11.4.13 CPLParseNameValue()**

```
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 **VSIAlloc()** (p. ??), and returned in that pointer. It is the applications responsibility to free this string, but the application should not modify or free the returned value portion.

This function also support "NAME:VALUE" strings and will strip white space from around the delimiter when forming name and value strings.

Eventually **CSLFetchNameValue()** (p. ??) and friends may be modified to use **CPLParseNameValue()** (p. ??).

**Parameters**

<i>pszNameValue</i>	string in "NAME=VALUE" format.
<i>ppszKey</i>	optional pointer through which to return the name portion.

**Returns**

the value portion (pointing into original string).

References CPLMalloc().

Referenced by CSLMerge(), CSLSetNameValueSeparator(), and OGRGeometryFactory::curveToLineString().

**12.11.4.14 CPLprintf()**

```
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**

<i>fmt</i>	formatting string
...	arguments

**Returns**

the number of characters (excluding terminating nul) written in output buffer.

**Since**

GDAL 2.0

References CPLvsprintf().

**12.11.4.15 CPLRecode()**

```
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 **CPL\_Error()** (p. ??).

#### Parameters

<i>pszSource</i>	a NULL terminated string.
<i>pszSrcEncoding</i>	the source encoding.
<i>pszDstEncoding</i>	the destination encoding.

#### Returns

a NULL terminated string which should be freed with **CPL\_Free()** (p. ??).

#### Since

GDAL 1.6.0

References CPL\_ENC\_ASCII, CPL\_ENC\_ISO8859\_1, CPL\_ENC\_UTF8, CPLStrdup(), and EQUAL.

Referenced by CPLString::Recode().

#### 12.11.4.16 CPLRecodeFromWChar()

```
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 (i.e. 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 **CPL\_Error()** (p. ??).

#### Parameters

<i>pwszSource</i>	the source wchar_t string, terminated with a 0 wchar_t.
<i>pszSrcEncoding</i>	the source encoding, typically CPL_ENC_UCS2.
<i>pszDstEncoding</i>	the destination encoding, typically CPL_ENC_UTF8.

**Returns**

a zero terminated multi-byte string which should be freed with **CPLFree()** (p. ??), or NULL if an error occurs.

**Since**

GDAL 1.6.0

References CPL\_ENC\_ASCII, CPL\_ENC\_ISO8859\_1, CPL\_ENC\_UCS2, CPL\_ENC\_UTF8, and EQUAL.

Referenced by CPLUnescapeString().

**12.11.4.17 CPLRecodeToWChar()**

```
wchar_t* CPLRecodeToWChar (
    const char * pszSource,
    const char * pszSrcEncoding,
    const char * pszDstEncoding )
```

Convert UTF-8 string to a wchar\_t string.

Convert a 8bit, multi-byte per character input string into a wide character (wchar\_t) string. The only guaranteed supported source encodings are CPL\_ENC\_UTF8, CPL\_ENC\_ASCII and CPL\_ENC\_ISO8869\_1 (LATIN1). The only guaranteed supported destination encoding is CPL\_ENC\_UCS2. Other source and destination encodings may be supported depending on the underlying implementation.

Note that the wchar\_t type varies in size on different systems. On win32 it is normally 2 bytes, and on UNIX 4 bytes.

If an error occurs an error may, or may not be posted with **CPLError()** (p. ??).

**Parameters**

<i>pszSource</i>	input multi-byte character string.
<i>pszSrcEncoding</i>	source encoding, typically CPL_ENC_UTF8.
<i>pszDstEncoding</i>	destination encoding, typically CPL_ENC_UCS2.

**Returns**

the zero terminated wchar\_t string (to be freed with **CPLFree()** (p. ??)) or NULL on error.

**Since**

GDAL 1.6.0

References CPL\_ENC\_ASCII, CPL\_ENC\_ISO8859\_1, CPL\_ENC\_UCS2, CPL\_ENC\_UTF8, and EQUAL.

## 12.11.4.18 CPLsnprintf()

```
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

<i>str</i>	output buffer
<i>size</i>	size of the output buffer (including space for terminating nul)
<i>fmt</i>	formatting string
...	arguments

## Returns

the number of characters (excluding terminating nul) that would be written if size is big enough. Or potentially -1 with Microsoft C runtime for Visual Studio < 2015.

## Since

GDAL 2.0

References CPLvsprintf().

Referenced by CPLDebug(), CPLPrintDouble(), OGRFeature::DumpReadable(), CPLString::FormatC(), OGRFeature::GetFieldAsString(), and OGRFeature::SetField().

## 12.11.4.19 CPLSPrintf()

```
const char* CPLSPrintf (
    const char * fmt,
    ... )
```

**CPLSPrintf()** (p. ??) that works with 10 static buffer.

It returns a ref. to a static buffer that should not be freed and is valid only until the next call to **CPLSPrintf()** (p. ??).

References CPLCalloc(), CPLError(), and CPLvsprintf().

Referenced by CPLHTTPFetchEx(), CPLsetlocale(), OGRFeature::DumpReadable(), OGRSpatialReference::importFromProj4(), OGRStyleTable::IsExist(), OGRGeometryTypeToName(), CPLJSONArray::operator[](), OGRFeature::SetField(), VSIIInstallCryptFileHandler(), and VSIReadDirRecursive().

### 12.11.4.20 CPLStrlcat()

```
size_t CPLStrlcat (
    char * pszDest,
    const char * pszSrc,
    size_t nDestSize )
```

Appends a source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1 and that there is at least one byte free in pszDest, that is to say `strlen(pszDest_before) < nDestSize`)

This function is designed to be a safer, more consistent, and less error prone replacement for `strncat`. Its contract is identical to `libbsd's strlcat`.

Truncation can be detected by testing if the return value of `CPLStrlcat` is greater or equal to `nDestSize`.

```
char szDest[5] = {};
CPLStrlcpy(szDest, "ab", sizeof(szDest));
if( CPLStrlcat(szDest, "cde", sizeof(szDest)) >= sizeof(szDest) )
    fprintf(stderr, "truncation occurred !\n");
```

#### Parameters

<i>pszDest</i>	destination buffer. Must be NUL terminated before running <code>CPLStrlcat</code>
<i>pszSrc</i>	source string. Must be NUL terminated
<i>nDestSize</i>	size of destination buffer (including space for the NUL terminator character)

#### Returns

the theoretical 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()`.

### 12.11.4.21 CPLStrlcpy()

```
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( CPLStrncpy(szDest, "abcde", sizeof(szDest)) >= sizeof(szDest) )
    fprintf(stderr, "truncation occurred !\n");
```

**Parameters**

<i>pszDest</i>	destination buffer
<i>pszSrc</i>	source string. Must be NUL terminated
<i>nDestSize</i>	size of destination buffer (including space for the NUL terminator character)

**Returns**

the length of the source string (=strlen(pszSrc))

**Since**

GDAL 1.7.0

Referenced by CPLCleanTrailingSlash(), CPLFormFilename(), CPLGetBasename(), CPLGetDirname(), CPLGet↔Extension(), CPLGetPath(), CPLProjectRelativeFilename(), CPLResetExtension(), and CPLStrcat().

**12.11.4.22 CPLStrlenUTF8()**

```
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**

<i>pszUTF8Str</i>	a nul-terminated UTF-8 string
-------------------	-------------------------------

**Returns**

the number of UTF-8 characters.

**12.11.4.23 CPLStrnlen()**

```
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

<i>pszStr</i>	a NUL terminated string
<i>nMaxLen</i>	maximum number of bytes to read in pszStr

## Returns

strlen(pszStr) if the length is lesser than nMaxLen, otherwise nMaxLen if the NUL character has not been found in the first nMaxLen bytes.

## Since

GDAL 1.7.0

Referenced by CPLScanLong(), CPLScanUIntBig(), CPLScanULong(), OGRSpatialReference::importFromPCI(), and VSIFOpenExL().

## 12.11.4.24 CPLTestBool()

```
bool CPLTestBool (
    const char * pszValue )
```

Test what boolean value contained in the string.

If pszValue is "NO", "FALSE", "OFF" or "0" will be returned false. Otherwise, true will be returned.

## Parameters

<i>pszValue</i>	the string should be tested.
-----------------	------------------------------

## Returns

true or false.

## References EQUAL.

Referenced by OGRLayer::Clip(), CPLFetchBool(), CPLHTTPFetchEx(), CPLIsMachineForSureGCEInstance(), CPLIsMachinePotentiallyGCEInstance(), CPLTestBoolean(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), CSLTestBoolean(), OGRGeometry::dumpReadable(), OGRFeature::DumpReadable(), OGRLayer::Erase(), CPLStringList::FetchBool(), OGRFeatureDefn::GetGeomType(), OGRLayer::Identity(), OGRLayer::Intersection(), OGR\_G\_ExportToGMLEx(), OGRGeometryFactory::organizePolygons(), OGRFeature::SetField(), OGRLayer::SymDifference(), OGRGeometryFactory::transformWithOptions(), OGRLayer::Union(), OGRLayer::Update(), and OGRSpatialReference::Validate().

#### 12.11.4.25 CPLTestBoolean()

```
int CPLTestBoolean (
    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.

Use this only in C code. In C++, prefer **CPLTestBool()** (p. ??).

##### Parameters

<i>pszValue</i>	the string should be tested.
-----------------	------------------------------

##### Returns

TRUE or FALSE.

References CPLTestBool().

#### 12.11.4.26 CPLUnescapeString()

```
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 its original form.

##### Parameters

<i>pszInput</i>	the input string. This is a zero terminated string.
<i>pnLength</i>	location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.
<i>nScheme</i>	the escaped scheme to undo (see <b>CPLEscapeString()</b> (p. ??) for a list). Does not yet support CSV.

##### Returns

a copy of the unescaped string that should be freed by the application using **CPLFree()** (p. ??) when no longer needed.

References CPL\_ENC\_UTF8, CPLDebug(), CPLError(), CPLES\_BackslashQuotable, CPLES\_CSV, CPLES\_SQL, CPLES\_URL, CPLES\_XML, CPLES\_XML\_BUT\_QUOTES, CPLFree, CPLMalloc(), CPLRecodeFromWChar(), and STARTS\_WITH\_CI.

Referenced by OGRFeature::FillUnsetWithDefault().

#### 12.11.4.27 CPLURLAddKVP()

```
CPLString CPLURLAddKVP (
    const char * pszURL,
    const char * pszKey,
    const char * pszValue )
```

Return a new URL with a new key=value pair.

##### Parameters

<i>pszURL</i>	the URL.
<i>pszKey</i>	the key to find.
<i>pszValue</i>	the value of the key (may be NULL to unset an existing KVP).

##### Returns

the modified URL.

##### Since

GDAL 1.9.0

References CPLString::ifind().

#### 12.11.4.28 CPLURLGetValue()

```
CPLString CPLURLGetValue (
    const char * pszURL,
    const char * pszKey )
```

Return the value matching a key from a key=value pair in a URL.

##### Parameters

<i>pszURL</i>	the URL.
<i>pszKey</i>	the key to find.

##### Returns

the value of empty string if not found.

Since

GDAL 1.9.0

References `CPLString::ifind()`.

#### 12.11.4.29 CPLVASPrintf()

```
int CPLVASPrintf (
    char ** buf,
    const char * fmt,
    va_list ap )
```

This is intended to serve as an easy to use C callable `vasprintf()` alternative. Used in the GeoJSON library for instance

References `CPLStrdup()`, and `CPLString::vPrintf()`.

#### 12.11.4.30 CPLvsnprintf()

```
int CPLvsnprintf (
    char * str,
    size_t size,
    const char * fmt,
    va_list args )
```

`vsprintf()` wrapper that is not sensitive to `LC_NUMERIC` settings.

This function has the same contract as standard `vsprintf()`, except that formatting of floating-point numbers will use decimal point, whatever the current locale is set.

##### Parameters

<i>str</i>	output buffer
<i>size</i>	size of the output buffer (including space for terminating nul)
<i>fmt</i>	formatting string
<i>args</i>	arguments

##### Returns

the number of characters (excluding terminating nul) that would be written if size is big enough. Or potentially -1 with Microsoft C runtime for Visual Studio < 2015.

Since

GDAL 2.0

References CPLAssert, CPLDebug(), and end().

Referenced by CPLDebug(), CPLprintf(), CPLsnprintf(), CPLSPrintf(), and CPLString::vPrintf().

#### 12.11.4.31 CSLAddNameValue()

```
char** CSLAddNameValue (
    char ** papszStrList,
    const char * pszName,
    const char * pszValue )
```

Add a new entry to a StringList of "Name=Value" pairs, ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

This function does not check if a "Name=Value" pair already exists for that name and can generate multiple entries for the same name. Use **CSLSetNameValue()** (p. ??) if you want each name to have only one value.

Returns the modified StringList.

References CPLFree, CPLMalloc(), and CSLAddString().

Referenced by CSLSetNameValue(), and OGRSpatialReference::importFromProj4().

#### 12.11.4.32 CSLAddString()

```
char** CSLAddString (
    char ** papszStrList,
    const char * pszNewString )
```

Append a string to a StringList and return a pointer to the modified StringList.

If the input StringList is NULL, then a new StringList is created. Note that CSLAddString performance when building a list is in  $O(n^2)$  which can cause noticeable slow down when  $n > 10000$ .

References CSLAddStringMayFail().

Referenced by OGRStyleTable::AddStyle(), CPLCorrespondingPaths(), CSLAddNameValue(), CSLAppendPrintf(), CSLFetchNameValueMultiple(), OPTGetParameterList(), and OPTGetProjectionMethods().

#### 12.11.4.33 CSLAddStringMayFail()

```
char** CSLAddStringMayFail (
    char ** papszStrList,
    const char * pszNewString )
```

Same as **CSLAddString()** (p. ??) but may return NULL in case of (memory) failure

References CSLCount(), VSI\_CALLOC\_VERBOSE, VSI\_REALLOC\_VERBOSE, VSI\_STRDUP\_VERBOSE, and VSIFree().

Referenced by CPLPushFinderLocation(), and CSLAddString().

#### 12.11.4.34 CSLAppendPrintf()

```
char** CSLAppendPrintf (
    char ** papszStrList,
    const char * fmt,
    ... )
```

Use **CPLSPrintf()** (p. ??) to append a new line at the end of a StringList. Returns the modified StringList.

References CSLAddString(), and CPLString::vPrintf().

#### 12.11.4.35 CSLCount()

```
int CSLCount (
    CSLConstList papszStrList )
```

Return number of items in a string list.

Returns the number of items in a string list, not counting the terminating NULL. Passing in NULL is safe, and will result in a count of zero.

Lists are counted by iterating through them so long lists will take more time than short lists. Care should be taken to avoid using **CSLCount()** (p. ??) as an end condition for loops as it will result in  $O(n^2)$  behavior.

##### Parameters

<i>papszStrList</i>	the string list to count.
---------------------	---------------------------

##### Returns

the number of entries.

Referenced by CPLStringList::Count(), CPLCorrespondingPaths(), CPLDefaultFindFile(), CSLAddStringMayFail(), CSLDuplicate(), CSLInsertStrings(), CSLRemoveStrings(), CSLSetNameValueSeparator(), OGRFeature::Equal(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::GetAttrNode(), OGRStyleTable::GetNextStyle(), OGRStyleTable::GetStyleName(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromWMSAUTO(), OGRStyleTable::IsExist(), OGRFeature::SetField(), OGRSpatialReference::SetNode(), and VSIRReadDirRecursive().

#### 12.11.4.36 CSLDestroy()

```
void CSLDestroy (
    char ** papszStrList )
```

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

## Parameters

<i>papszStrList</i>	the list to free.
---------------------	-------------------

Referenced by OGRStyleTable::Clear(), CPLODBCStatement::Clear(), CPLStringList::Clear(), CPLGetXMLNode(), CPLHTTPDestroyResult(), CPLSetConfigOptions(), CPLSetThreadLocalConfigOptions(), CPLSetXMLValue(), C↵SLRemoveStrings(), OGRSpatialReference::exportToPCI(), OGR\_SRSNode::exportToPrettyWkt(), OGR\_SRS↵Node::exportToWkt(), OGRSpatialReference::GetAttrNode(), GOA2Manager::GetBearer(), OGRStyleMgr::Get↵Part(), GOA2GetAccessToken(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFrom↵PCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromWMSAUTO(), OGRStyle↵Table::LoadStyleTable(), OGRFeature::SetField(), OGRFeature::SetFieldNull(), OGRSpatialReference::SetNode(), OGRFeature::UnsetField(), VSIRReadDirRecursive(), and VSIRmdirRecursive().

## 12.11.4.37 CSLDuplicate()

```
char** CSLDuplicate (
    CSLConstList 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 **CSL↵Destroy()** (p. ??).

## Parameters

<i>papszStrList</i>	the input string list.
---------------------	------------------------

## Returns

newly allocated copy.

References CPLMalloc(), CPLStrdup(), and CSLCount().

Referenced by OGRStyleTable::Clone(), CPLGetConfigOptions(), CPLGetThreadLocalConfigOptions(), CPLSet↵ConfigOptions(), CPLSetThreadLocalConfigOptions(), CPLStringList::CPLStringList(), and CSLMerge().

## 12.11.4.38 CSLFetchBoolean()

```
int CSLFetchBoolean (
    CSLConstList papszStrList,
    const char * pszKey,
    int bDefault )
```

DEPRECATED. Check for boolean key value.

In a StringList of "Name=Value" pairs, look to see if there is a key with the given name, and if it can be interpreted as being TRUE. If the key appears without any "=Value" portion it will be considered true. If the value is NO, FALSE or 0 it will be considered FALSE otherwise if the key appears in the list it will be considered TRUE. If the key doesn't appear at all, the indicated default value will be returned.

**Parameters**

<i>papszStrList</i>	the string list to search.
<i>pszKey</i>	the key value to look for (case insensitive).
<i>bDefault</i>	the value to return if the key isn't found at all.

**Returns**

TRUE or FALSE

References CPLFetchBool().

**12.11.4.39 CSLFetchNameValue()**

```
const char* CSLFetchNameValue (
    CSLConstList papszStrList,
    const char * pszName )
```

In a StringList of "Name=Value" pairs, look for the first value associated with the specified name. The search is not case sensitive. ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

Returns a reference to the value in the StringList that the caller should not attempt to free.

Returns NULL if the name is not found.

References EQUALN.

Referenced by CPLFetchBool(), CPLGetConfigOption(), CPLGetThreadLocalConfigOption(), CPLHTTPFetchEx(), CSLFetchNameValueDef(), OGRGeometry::dumpReadable(), OGRFeature::DumpReadable(), GOA2Manager::GetBearer(), GOA2GetAccessToken(), OGRSpatialReference::importFromProj4(), OGR\_G\_ExportToGMLEx(), and OGRGeometryFactory::organizePolygons().

**12.11.4.40 CSLFetchNameValueDef()**

```
const char* CSLFetchNameValueDef (
    CSLConstList papszStrList,
    const char * pszName,
    const char * pszDefault )
```

Same as **CSLFetchNameValue()** (p. ??) but return pszDefault in case of no match

References CSLFetchNameValue().

Referenced by OGRLayer::Clip(), OGRLayer::Erase(), GOA2GetAccessTokenFromCloudEngineVM(), OGRLayer::Identity(), OGRLayer::Intersection(), OGRSpatialReference::IsSameGeogCS(), CPLJSONDocument::LoadUrl(), OGR\_G\_ExportToGMLEx(), OGR\_G\_ExportToJsonEx(), OGRLayer::SymDifference(), OGRGeometryFactory::transformWithOptions(), OGRLayer::Union(), and OGRLayer::Update().



#### 12.11.4.41 CSLFetchNameValueMultiple()

```
char** CSLFetchNameValueMultiple (
    CSLConstList papszStrList,
    const char * pszName )
```

In a StringList of "Name=Value" pairs, look for all the values with the specified name. The search is not case sensitive. ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

Returns StringList with one entry for each occurrence of the specified name. The StringList should eventually be destroyed by calling **CSLDestroy()** (p. ??).

Returns NULL if the name is not found.

References CSLAddString(), and EQUALN.

#### 12.11.4.42 CSLFindName()

```
int CSLFindName (
    CSLConstList papszStrList,
    const char * pszName )
```

Find StringList entry with given key name.

##### Parameters

<i>papszStrList</i>	the string list to search.
<i>pszName</i>	the key value to look for (case insensitive).

##### Returns

-1 on failure or the list index of the first occurrence matching the given key.

References EQUALN.

Referenced by CPLStringList::FindName().

#### 12.11.4.43 CSLFindString()

```
int CSLFindString (
    CSLConstList 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**

<i>papszList</i>	the string list to be searched.
<i>pszTarget</i>	the string to be searched for.

**Returns**

the index of the string within the list or -1 on failure.

References EQUAL.

Referenced by CPLFetchBool(), CPLStringList::FindString(), and OGR\_SRSNode::FixupOrdering().

**12.11.4.44 CSLFindStringCaseSensitive()**

```
int CSLFindStringCaseSensitive (
    CSLConstList 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**

<i>papszList</i>	the string list to be searched.
<i>pszTarget</i>	the string to be searched for.

**Returns**

the index of the string within the list or -1 on failure.

**Since**

GDAL 2.0

Referenced by CPLPushFinderLocation().

**12.11.4.45 CSLGetField()**

```
const char* CSLGetField (
    CSLConstList papszStrList,
    int iField )
```

Fetches the indicated field, being careful not to crash if the field doesn't exist within this string list.

The returned pointer should not be freed, and doesn't necessarily last long.

Referenced by OGRStyleTable::Find(), OGRStyleTable::GetNextStyle(), OGRStyleMgr::GetPart(), and OGR\_SpatialReference::morphFromESRI().

#### 12.11.4.46 CSLInsertString()

```
char** CSLInsertString (
    char ** papszStrList,
    int nInsertAtLineNo,
    const char * pszNewLine )
```

Insert a string at a given line number inside a StringList

nInsertAtLineNo is a 0-based line index before which the new string should be inserted. If this value is -1 or is larger than the actual number of strings in the list then the string is added at the end of the source StringList.

Returns the modified StringList.

References CSLInsertStrings().

#### 12.11.4.47 CSLInsertStrings()

```
char** CSLInsertStrings (
    char ** papszStrList,
    int nInsertAtLineNo,
    CSLConstList papszNewLines )
```

Copies the contents of a StringList inside another StringList before the specified line.

nInsertAtLineNo is a 0-based line index before which the new strings should be inserted. If this value is -1 or is larger than the actual number of strings in the list then the strings are added at the end of the source StringList.

Returns the modified StringList.

References CPLRealloc(), CPLStrdup(), and CSLCount().

Referenced by CSLInsertString().

#### 12.11.4.48 CSLLoad()

```
char** CSLLoad (
    const char * pszFname )
```

Load a text file into a string list.

The VSI\*L API is used, so **VSI\*OpenL()** (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 **CPL\*Error()** (p. ??) will be issued and NULL returned.

**Parameters**

<i>pszFname</i>	the name of the file to read.
-----------------	-------------------------------

**Returns**

a string list with the files lines, now owned by caller. To be freed with **CSLDestroy()** (p. ??)

References CSLLoad2().

Referenced by OGRStyleTable::LoadStyleTable().

**12.11.4.49 CSLLoad2()**

```
char** CSLLoad2 (
    const char * pszFname,
    int nMaxLines,
    int nMaxCols,
    CSLConstList papszOptions )
```

Load a text file into a string list.

The VSI\*L API is used, so **VSIFOpenL()** (p. ??) supported objects that aren't physical files can also be accessed. Files are returned as a string list, with one item in the string list per line. End of line markers are stripped (by **CPLReadLineL()** (p. ??)).

If reading the file fails a **CPLError()** (p. ??) will be issued and NULL returned.

**Parameters**

<i>pszFname</i>	the name of the file to read.
<i>nMaxLines</i>	maximum number of lines to read before stopping, or -1 for no limit.
<i>nMaxCols</i>	maximum number of characters in a line before stopping, or -1 for no limit.
<i>papszOptions</i>	NULL-terminated array of options. Unused for now.

**Returns**

a string list with the files lines, now owned by caller. To be freed with **CSLDestroy()** (p. ??)

**Since**

GDAL 1.7.0

References CPLError(), CPLFetchBool(), and VSIFOpenL().

Referenced by CSLLoad().

#### 12.11.4.50 CSLMerge()

```
char** CSLMerge (
    char ** papszOrig,
    CSLConstList papszOverride )
```

Merge two lists.

The two lists are merged, ensuring that if any keys appear in both that the value from the second (*papszOverride*) list take precedence.

##### Parameters

<i>papszOrig</i>	the original list, being modified.
<i>papszOverride</i>	the list of items being merged in. This list is unaltered and remains owned by the caller.

##### Returns

updated list.

References CPLFree, CPLParseNameValue(), CSLDuplicate(), and CSLSetNameValue().

#### 12.11.4.51 CSLParseCommandLine()

```
char** CSLParseCommandLine (
    const char * pszCommandLine )
```

Tokenize command line arguments in a list of strings.

##### Parameters

<i>pszCommandLine</i>	command line
-----------------------	--------------

##### Returns

NULL terminated list of strings to free with **CSLDestroy()** (p. ??)

##### Since

GDAL 2.1

References CSLTokenizeString().

#### 12.11.4.52 CSLPartialFindString()

```
int CSLPartialFindString (
    CSLConstList papszHaystack,
    const char * pszNeedle )
```

Find a substring within a string list.

Returns the index of the entry in the string list that contains the target string as a substring. The search is case sensitive (unlike **CSLFindString()** (p. ??)).

##### Parameters

<i>papszHaystack</i>	the string list to be searched.
<i>pszNeedle</i>	the substring to be searched for.

##### Returns

the index of the string within the list or -1 on failure.

Referenced by CPLStringList::PartialFindString().

#### 12.11.4.53 CSLPrint()

```
int CSLPrint (
    CSLConstList papszStrList,
    FILE * fpOut )
```

Print a StringList to fpOut. If fpOut==NULL, then output is sent to stdout.

Returns the number of lines printed.

#### 12.11.4.54 CSLRemoveStrings()

```
char** CSLRemoveStrings (
    char ** papszStrList,
    int nFirstLineToDelete,
    int nNumToRemove,
    char *** ppapszRetStrings )
```

Remove strings inside a StringList

nFirstLineToDelete is the 0-based line index of the first line to remove. If this value is -1 or is larger than the actual number of strings in list then the nNumToRemove last strings are removed.

If ppapszRetStrings != NULL then the deleted strings won't be free'd, they will be stored in a new StringList and the pointer to this new list will be returned in \*ppapszRetStrings.

Returns the modified StringList.

References CPLAlloc(), CPLFree, CSLCount(), and CSLDestroy().

Referenced by OGRStyleTable::RemoveStyle().

## 12.11.4.55 CSLSave()

```
int CSLSave (
    CSLConstList papszStrList,
    const char * pszFname )
```

Write a StringList to a text file.

Returns the number of lines written, or 0 if the file could not be written.

References CPLError(), and VSIFOpenL().

Referenced by OGRStyleTable::SaveStyleTable().

## 12.11.4.56 CSLSetNameValue()

```
char** CSLSetNameValue (
    char ** papszList,
    const char * pszName,
    const char * pszValue )
```

Assign value to name in StringList.

Set the value for a given name in a StringList of "Name=Value" pairs ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

If there is already a value for that name in the list then the value is changed, otherwise a new "Name=Value" pair is added.

## Parameters

<i>papszList</i>	the original list, the modified version is returned.
<i>pszName</i>	the name to be assigned a value. This should be a well formed token (no spaces or very special characters).
<i>pszValue</i>	the value to assign to the name. This should not contain any newlines (CR or LF) but is otherwise pretty much unconstrained. If NULL any corresponding value will be removed.

## Returns

modified StringList.

References CPLFree, CPLMalloc(), CSLAddNameValue(), and EQUALN.

Referenced by CPLSetConfigOption(), CPLSetThreadLocalConfigOption(), and CSLMerge().

#### 12.11.4.57 CSLSetNameValueSeparator()

```
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 manipulable by the CSL name/value functions any more.

The **CPLParseNameValue()** (p. ??) function is used to break the existing lines, and it also strips white space from around the existing delimiter, thus the old separator, and any white space will be replaced by the new separator. For formatting purposes it may be desirable to include some white space in the new separator. e.g. ": " or "= ".

##### Parameters

<i>papszList</i>	the list to update. Component strings may be freed but the list array will remain at the same location.
<i>pszSeparator</i>	the new separator string to insert.

References CPLFree, CPLMalloc(), CPLParseNameValue(), and CSLCount().

#### 12.11.4.58 CSLTestBoolean()

```
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.

Deprecated. Removed in GDAL 3.x.

Use **CPLTestBoolean()** (p. ??) for C and **CPLTestBool()** (p. ??) for C++.

##### Parameters

<i>pszValue</i>	the string should be tested.
-----------------	------------------------------

##### Returns

TRUE or FALSE.

References CPLTestBool().



## 12.11.4.59 CSLTokenizeString()

```
char** CSLTokenizeString (
    const char * pszString )
```

Tokenizes a string and returns a StringList with one string for each token.

References CSLT\_HONOURSTRINGS, and CSLTokenizeString2().

Referenced by CSLParseCommandLine().

## 12.11.4.60 CSLTokenizeString2()

```
char** CSLTokenizeString2 (
    const char * pszString,
    const char * pszDelimiters,
    int nCSLTFlags )
```

Tokenize a string.

This function will split a string into tokens based on specified' delimiter(s) with a variety of options. The returned result is a string list that should be freed with **CSLDestroy()** (p. ??) when no longer needed.

The available parsing options are:

- CSLT\_ALLOWEMPTYTOKENS: allow the return of empty tokens when two delimiters in a row occur with no other text between them. If not set, empty tokens will be discarded;
- CSLT\_STRIPLEADSPACES: strip leading space characters from the token (as reported by isspace());
- CSLT\_STRIPENDSPACES: strip ending space characters from the token (as reported by isspace());
- CSLT\_HONOURSTRINGS: double quotes can be used to hold values that should not be broken into multiple tokens;
- CSLT\_PRESERVEQUOTES: string quotes are carried into the tokens when this is set, otherwise they are removed;
- CSLT\_PRESERVEESCAPES: if set backslash escapes (for backslash itself, and for literal double quotes) will be preserved in the tokens, otherwise the backslashes will be removed in processing.

**Example:**

Parse a string into tokens based on various white space (space, newline, tab) and then print out results and cleanup. Quotes may be used to hold white space in tokens.

```
char **papszTokens =
    CSLTokenizeString2( pszCommand, " \t\n",
        CSLT_HONOURSTRINGS | CSLT_ALLOWEMPTYTOKENS );

for( int i = 0; papszTokens != NULL && papszTokens[i] != NULL; ++i )
    printf( "arg %d: '%s'", papszTokens[i] ); // ok

CSLDestroy( papszTokens );
```

**Parameters**

<i>pszString</i>	the string to be split into tokens.
<i>pszDelimiters</i>	one or more characters to be used as token delimiters.
<i>nCSLTFlags</i>	an ORing of one or more of the CSLT_ flag values.

**Returns**

a string list of tokens owned by the caller.

References CPLStringList::AddString(), CPLStringList::Assign(), CPLStringList::Count(), CPLCalloc(), CPLFree, CPLRealloc(), CSLT\_ALLOWEMPTYTOKENS, CSLT\_HONOURSTRINGS, CSLT\_PRESERVEESCAPES, CSLT\_PRESERVEQUOTES, CSLT\_STRIPENDSPACES, CSLT\_STRIPLEADSPACES, CPLStringList::List(), and CPLStringList::StealList().

Referenced by CSLTokenizeString(), CSLTokenizeStringComplex(), OGRStyleMgr::GetPart(), and OGRSpatialReference::importFromOzi().

**12.11.4.61 CSLTokenizeStringComplex()**

```
char** CSLTokenizeStringComplex (
    const char * pszString,
    const char * pszDelimiters,
    int bHonourStrings,
    int bAllowEmptyTokens )
```

Obsolete tokenizing api. Use **CSLTokenizeString2()** (p. ??)

References CSLT\_ALLOWEMPTYTOKENS, CSLT\_HONOURSTRINGS, and CSLTokenizeString2().

Referenced by CPLGetXMLNode(), CPLSetXMLValue(), OGRSpatialReference::GetAttrNode(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromWMSAUTO(), and OGRSpatialReference::SetNode().

**12.12 cpl\_virtualmem.h File Reference**

```
#include <stddef.h>
#include "cpl_port.h"
#include "cpl_vsi.h"
```

**Typedefs**

- typedef struct **CPLVirtualMem** **CPLVirtualMem**
- typedef void(\* **CPLVirtualMemCachePageCb**) ( **CPLVirtualMem** \*ctxt, size\_t nOffset, void \*pPageToFill, size\_t nToFill, void \*pUserData)
- typedef void(\* **CPLVirtualMemUnCachePageCb**) ( **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, **CPLVirtualMemCachePageCb** pfnCachePage, **CPLVirtualMemUnCachePageCb** 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)

### 12.12.1 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.

### 12.12.2 Typedef Documentation

### 12.12.2.1 CPLVirtualMem

```
typedef struct CPLVirtualMem CPLVirtualMem
```

Opaque type that represents a virtual memory mapping.

### 12.12.2.2 CPLVirtualMemCachePageCbK

```
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

<i>ctxt</i>	virtual memory handle.
<i>nOffset</i>	offset of the page in the memory mapping.
<i>pPageToFill</i>	address of the page to fill. Note that the address might be a temporary location, and not at <b>CPLVirtualMemGetAddr()</b> (p. ??) + nOffset.
<i>nToFill</i>	number of bytes of the page.
<i>pUserData</i>	user data that was passed to <b>CPLVirtualMemNew()</b> (p. ??).

### 12.12.2.3 CPLVirtualMemFreeUserData

```
typedef void(* CPLVirtualMemFreeUserData) (void *pUserData)
```

Callback triggered when a virtual memory mapping is destroyed.

#### Parameters

<i>pUserData</i>	user data that was passed to <b>CPLVirtualMemNew()</b> (p. ??).
------------------	---

### 12.12.2.4 CPLVirtualMemUnCachePageCbK

```
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

<i>ctxt</i>	virtual memory handle.
-------------	------------------------

## Parameters

<i>nOffset</i>	offset of the page in the memory mapping.
<i>pPageToBeEvicted</i>	address of the page that will be flushed. Note that the address might be a temporary location, and not at <b>CPLVirtualMemGetAddr()</b> (p. ??) + nOffset.
<i>nToBeEvicted</i>	number of bytes of the page.
<i>pUserData</i>	user data that was passed to <b>CPLVirtualMemNew()</b> (p. ??).

## 12.12.3 Enumeration Type Documentation

## 12.12.3.1 CPLVirtualMemAccessMode

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

## 12.12.4 Function Documentation

## 12.12.4.1 CPLGetPageSize()

```
size_t CPLGetPageSize (
    void )
```

Return the size of a page of virtual memory.

## Returns

the page size.

## Since

GDAL 1.11

#### 12.12.4.2 CPLIsVirtualMemFileMapAvailable()

```
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

#### 12.12.4.3 CPLVirtualMemDeclareThread()

```
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

<code>ctxt</code>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------------	--

##### Since

GDAL 1.11

#### 12.12.4.4 CPLVirtualMemDerivedNew()

```
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

<i>pVMemBase</i>	Base virtual memory mapping
<i>nOffset</i>	Offset in the base virtual memory mapping from which to start the new mapping.
<i>nSize</i>	Size of the base virtual memory mapping to expose in the the new mapping.
<i>pfnFreeUserData</i>	callback that is called when the object is destroyed.
<i>pCbkUserData</i>	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 VSI\_CALLOC\_VERBOSE.

## 12.12.4.5 CPLVirtualMemFileMapNew()

```
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.

Supported on Linux only in GDAL <= 2.0, and all POSIX systems supporting mmap() in GDAL >= 2.1

## Parameters

<i>fp</i>	Virtual file handle.
<i>nOffset</i>	Offset in the file to start the mapping from.
<i>nLength</i>	Length of the portion of the file to map into memory.
<i>eAccessMode</i>	Permission to use for the virtual memory mapping. This must be consistent with how the file has been opened.
<i>pfnFreeUserData</i>	callback that is called when the object is destroyed.
<i>pCbkUserData</i>	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().

**12.12.4.6 CPLVirtualMemFree()**

```
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**

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------	--

**Since**

GDAL 1.11

References CPLFree.

**12.12.4.7 CPLVirtualMemGetAccessMode()**

```
CPLVirtualMemAccessMode CPLVirtualMemGetAccessMode (
    CPLVirtualMem * ctxt )
```

Return the access mode of the virtual memory mapping.

**Parameters**

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------	--

**Returns**

the access mode of the virtual memory mapping.



Since

GDAL 1.11

#### 12.12.4.8 CPLVirtualMemGetAddr()

```
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

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------	--

##### Returns

the pointer to the start of a virtual memory mapping.

Since

GDAL 1.11

#### 12.12.4.9 CPLVirtualMemGetPageSize()

```
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

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------	--

**Returns**

the page size

**Since**

GDAL 1.11

**12.12.4.10 CPLVirtualMemGetSize()**

```
size_t CPLVirtualMemGetSize (
    CPLVirtualMem * ctxt )
```

Return the size of the virtual memory mapping.

**Parameters**

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------	--

**Returns**

the size of the virtual memory mapping.

**Since**

GDAL 1.11

**12.12.4.11 CPLVirtualMemIsAccessThreadSafe()**

```
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 or not in **CPLVirtualMemNew()** (p. ??) and/or the implementation.

On Linux, this will always return TRUE if `bSingleThreadUsage` = FALSE.

**Parameters**

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------	--

**Returns**

TRUE if this memory mapping can be accessed safely from concurrent threads.

**Since**

GDAL 1.11

**12.12.4.12 CPLVirtualMemIsFileMapping()**

```
int CPLVirtualMemIsFileMapping (
    CPLVirtualMem * ctxt )
```

Return if the virtual memory mapping is a direct file mapping.

**Parameters**

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------	--

**Returns**

TRUE if the virtual memory mapping is a direct file mapping.

**Since**

GDAL 1.11

**12.12.4.13 CPLVirtualMemManagerTerminate()**

```
void CPLVirtualMemManagerTerminate (
    void )
```

Cleanup any resource and handlers related to virtual memory.

This function must be called after the last **CPLVirtualMem** (p. ??) object has been freed.

**Since**

GDAL 2.0

## 12.12.4.14 CPLVirtualMemNew()

```

CPLVirtualMem* CPLVirtualMemNew (
    size_t nSize,
    size_t nCacheSize,
    size_t nPageSizeHint,
    int bSingleThreadUsage,
    CPLVirtualMemAccessMode eAccessMode,
    CPLVirtualMemCachePageCbk pfnCachePage,
    CPLVirtualMemUnCachePageCbk pfnUnCachePage,
    CPLVirtualMemFreeUserData pfnFreeUserData,
    void * pCbkJUserData )

```

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 **CPLVirtualMemManagerTerminate()** (p. ??).

## Parameters

<i>nSize</i>	size in bytes of the virtual memory mapping.
<i>nCacheSize</i>	size in bytes of the maximum memory that will be really allocated (must ideally fit into RAM).
<i>nPageSizeHint</i>	hint for the page size. Must be a multiple of the system page size, returned by <b>CPLGetPageSize()</b> (p. ??). Minimum value is generally 4096. Might be set to 0 to let the function determine a default page size.
<i>bSingleThreadUsage</i>	set to TRUE if there will be no concurrent threads that will access the virtual memory mapping. This can optimize performance a bit.
<i>eAccessMode</i>	permission to use for the virtual memory mapping.
<i>pfnCachePage</i>	callback triggered when a still unmapped page of virtual memory is accessed. The callback has the responsibility of filling the page with relevant values.
<i>pfnUnCachePage</i>	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.
<i>pfnFreeUserData</i>	callback that can be used to free <code>pCbkJUserData</code> . Might be NULL
<i>pCbkJUserData</i>	user data passed to <code>pfnCachePage</code> and <code>pfnUnCachePage</code> .

## Returns

a virtual memory object that must be freed by **CPLVirtualMemFree()** (p. ??), or NULL in case of failure.

## Since

GDAL 1.11

References **CPLError()**.

## 12.12.4.15 CPLVirtualMemPin()

```
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

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
<i>pAddr</i>	the memory region to pin.
<i>nSize</i>	the size of the memory region.
<i>bWriteOp</i>	set to TRUE if the memory are will be accessed in write mode.

Since

GDAL 1.11

## 12.12.4.16 CPLVirtualMemUnDeclareThread()

```
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

<i>ctxt</i>	context returned by <b>CPLVirtualMemNew()</b> (p. ??).
-------------	--

Since

GDAL 1.11

## 12.13 cpl\_vsi.h File Reference

```
#include "cpl_port.h"
#include <unistd.h>
#include <sys/stat.h>
```

### Macros

- **#define VSI\_ISLNK(x) S\_ISLNK(x)**
- **#define VSI\_ISREG(x) S\_ISREG(x)**
- **#define VSI\_ISDIR(x) S\_ISDIR(x)**
- **#define VSI\_L\_OFFSET\_MAX GUINTBIG\_MAX**
- **#define VSI\_STAT\_EXISTS\_FLAG 0x1**
- **#define VSI\_STAT\_NATURE\_FLAG 0x2**
- **#define VSI\_STAT\_SIZE\_FLAG 0x4**
- **#define VSI\_STAT\_SET\_ERROR\_FLAG 0x8**
- **#define VSI\_MALLOC\_ALIGNED\_AUTO\_VERBOSE(size) VSIMallocAlignedAutoVerbose(size, \_\_FILE\_\_, \_\_LINE\_\_)**
- **#define VSI\_MALLOC\_VERBOSE(size) VSIMallocVerbose(size, \_\_FILE\_\_, \_\_LINE\_\_)**
- **#define VSI\_MALLOC2\_VERBOSE(nSize1, nSize2) VSIMalloc2Verbose(nSize1, nSize2, \_\_FILE\_\_, \_\_LINE\_\_)**
- **#define VSI\_MALLOC3\_VERBOSE(nSize1, nSize2, nSize3) VSIMalloc3Verbose(nSize1, nSize2, nSize3, \_\_FILE\_\_, \_\_LINE\_\_)**
- **#define VSI\_CALLOC\_VERBOSE(nCount, nSize) VSICallocVerbose(nCount, nSize, \_\_FILE\_\_, \_\_LINE\_\_)**
- **#define VSI\_REALLOC\_VERBOSE(pOldPtr, nNewSize) VSIReallocVerbose(pOldPtr, nNewSize, \_\_FILE\_\_, \_\_LINE\_\_)**
- **#define VSI\_STRDUP\_VERBOSE(pszStr) VSIStrdupVerbose(pszStr, \_\_FILE\_\_, \_\_LINE\_\_)**
- **#define CPLReadDir VSIReadDir**

### Typedefs

- **typedef GUIntBig vsi\_l\_offset**
- **typedef FILE VSILFILE**
- **typedef struct stat64 VSISStatBufL**
- **typedef size\_t(\* VSIWriteFunction) (const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream)**

### Enumerations

- **enum VSIRangeStatus { VSI\_RANGE\_STATUS\_UNKNOWN, VSI\_RANGE\_STATUS\_DATA, VSI\_RANGE\_STATUS\_HOLE }**

## Functions

- **VSILFILE \* VSIFOpenL** (const char \*, const char \*)  
*Open file.*
- **VSILFILE \* VSIFOpenExL** (const char \*, const char \*, int)  
*Open file.*
- int **VSIFCloseL** ( **VSILFILE \***) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Close file.*
- int **VSIFSeekL** ( **VSILFILE \***, **vsi\_l\_offset**, int) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Seek to requested offset.*
- **vsi\_l\_offset VSIFTellL** ( **VSILFILE \***)  
*Tell current file offset.*
- void **VSIRewindL** ( **VSILFILE \***)  
*Rewind the file pointer to the beginning of the file.*
- size\_t **VSIFReadL** (void \*, size\_t, size\_t, **VSILFILE \***) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Read bytes from file.*
- int **VSIFReadMultiRangeL** (int nRanges, void \*\*ppData, const **vsi\_l\_offset** \*panOffsets, const size\_t \*panSizes, **VSILFILE \***) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Read several ranges of bytes from file.*
- size\_t **VSIFWriteL** (const void \*, size\_t, size\_t, **VSILFILE \***) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Write bytes to file.*
- int **VSIFEOF** ( **VSILFILE \***) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Test for end of file.*
- int **VSIFTruncateL** ( **VSILFILE \***, **vsi\_l\_offset**) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Truncate/expand the file to the specified size.*
- int **VSIFFlushL** ( **VSILFILE \***) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Flush pending writes to disk.*
- int **VSIFPrintfL** ( **VSILFILE \***, const char \*,...) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Formatted write to file.*
- int **VSIFPutcL** (int, **VSILFILE \***) EXPERIMENTAL\_CPL\_WARN\_UNUSED\_RESULT  
*Write a single byte to the file.*
- **VSIRangeStatus VSIFGetRangeStatusL** ( **VSILFILE \***fp, **vsi\_l\_offset** nStart, **vsi\_l\_offset** nLength)  
*Return if a given file range contains data or holes filled with zeroes.*
- int **VSIIngestFile** ( **VSILFILE \***fp, const char \*pszFilename, **GByte** \*\*ppabyRet, **vsi\_l\_offset** \*pnSize, **GIntBig** nMaxSize)  
*Ingest a file into memory.*
- int **VSISatL** (const char \*, **VSISatBufL** \*)  
*Get filesystem object info.*
- int **VSISatExL** (const char \*pszFilename, **VSISatBufL** \*psStatBuf, int nFlags)  
*Get filesystem object info.*
- int **VSIIIsCaseSensitiveFS** (const char \*pszFilename)  
*Returns if the filenames of the filesystem are case sensitive.*
- int **VSISupportsSparseFiles** (const char \*pszPath)  
*Returns if the filesystem supports sparse files.*
- int **VSIIHasOptimizedReadMultiRange** (const char \*pszPath)  
*Returns if the filesystem supports efficient multi-range reading.*
- const char \* **VSIGetActualURL** (const char \*pszFilename)  
*Returns the actual URL of a supplied filename.*
- char \* **VSIGetSignedURL** (const char \*pszFilename, **CSLConstList** papszOptions)  
*Returns a signed URL of a supplied filename.*

- **const char \* VSIGetFileSystemOptions** (const char \*pszFilename)  
*Return the list of options associated with a virtual file system handler as a serialized XML string.*
- **char \*\* VSIGetFileSystemsPrefixes** (void)  
*Return the list of prefixes for virtual file system handlers currently registered.*
- **void \* VSIFGetNativeFileDescriptorL ( VSILFILE \*)**  
*Returns the "native" file descriptor for the virtual handle.*
- **void \* VSICalloc** (size\_t, size\_t)
- **void \* VSIMalloc** (size\_t)
- **void VSIFree** (void \*)
- **void \* VSIRealloc** (void \*, size\_t)
- **char \* VSIStrdup** (const char \*)
- **void \* VSIMallocAligned** (size\_t nAlignment, size\_t nSize)
- **void \* VSIMallocAlignedAuto** (size\_t nSize)
- **void VSIFreeAligned** (void \*ptr)
- **void \* VSIMallocAlignedAutoVerbose** (size\_t nSize, const char \*pszFile, int nLine)
- **void \* VSIMalloc2** (size\_t nSize1, size\_t nSize2)
- **void \* VSIMalloc3** (size\_t nSize1, size\_t nSize2, size\_t nSize3)
- **void \* VSIMallocVerbose** (size\_t nSize, const char \*pszFile, int nLine)
- **void \* VSIMalloc2Verbose** (size\_t nSize1, size\_t nSize2, const char \*pszFile, int nLine)
- **void \* VSIMalloc3Verbose** (size\_t nSize1, size\_t nSize2, size\_t nSize3, const char \*pszFile, int nLine)
- **void \* VSICallocVerbose** (size\_t nCount, size\_t nSize, const char \*pszFile, int nLine)
- **void \* VSIReallocVerbose** (void \*pOldPtr, size\_t nNewSize, const char \*pszFile, int nLine)
- **char \* VSIStrdupVerbose** (const char \*pszStr, const char \*pszFile, int nLine)
- **GIntBig CPLGetPhysicalIRAM** (void)
- **GIntBig CPLGetUsablePhysicalIRAM** (void)
- **char \*\* VSIReadDir** (const char \*)  
*Read names in a directory.*
- **char \*\* VSIReadDirRecursive** (const char \*pszPath)  
*Read names in a directory recursively.*
- **char \*\* VSIReadDirEx** (const char \*pszPath, int nMaxFiles)  
*Read names in a directory.*
- **int VSIMkdir** (const char \*pszPathname, long mode)  
*Create a directory.*
- **int VSIMkdirRecursive** (const char \*pszPathname, long mode)  
*Create a directory and all its ancestors.*
- **int VSIRmdir** (const char \*pszDirname)  
*Delete a directory.*
- **int VSIRmdirRecursive** (const char \*pszDirname)  
*Delete a directory recursively.*
- **int VSIUnlink** (const char \*pszFilename)  
*Delete a file.*
- **int VSIRename** (const char \*oldpath, const char \*newpath)  
*Rename a file.*
- **char \* VSIStrerror** (int)
- **GIntBig VSIGetDiskFreeSpace** (const char \*pszDirname)  
*Return free disk space available on the filesystem.*
- **void VSIInstallMemFileHandler** (void)  
*Install "memory" file system handler.*
- **void VSIInstallSubFileHandler** (void)
- **void VSIInstallCurlFileHandler** (void)  
*Install /vsicurl/ HTTP/FTP file system handler (requires libcurl)*
- **void VSICurlClearCache** (void)



*Clean local cache associated with /vsicurl/ (and related file systems)*

- void **VSIInstallCurlStreamingFileHandler** (void)  
*Install /vsicurl\_streaming/ HTTP/FTP file system handler (requires libcurl).*
- void **VSIInstallS3FileHandler** (void)  
*Install /vsi3/ Amazon S3 file system handler (requires libcurl)*
- void **VSIInstallS3StreamingFileHandler** (void)  
*Install /vsi3\_streaming/ Amazon S3 file system handler (requires libcurl).*
- void **VSIInstallGSFileHandler** (void)  
*Install /vsigs/ Google Cloud Storage file system handler (requires libcurl)*
- void **VSIInstallGSStreamingFileHandler** (void)  
*Install /vsigs\_streaming/ Google Cloud Storage file system handler (requires libcurl)*
- void **VSIInstallAzureFileHandler** (void)  
*Install /vsiaz/ Microsoft Azure Blob file system handler (requires libcurl)*
- void **VSIInstallAzureStreamingFileHandler** (void)  
*Install /vsiaz\_streaming/ Microsoft Azure Blob file system handler (requires libcurl)*
- void **VSIInstallOSSFileHandler** (void)  
*Install /vsioss/ Alibaba Cloud Object Storage Service (OSS) file system handler (requires libcurl)*
- void **VSIInstallOSSStreamingFileHandler** (void)  
*Install /vsioss\_streaming/ Alibaba Cloud Object Storage Service (OSS) file system handler (requires libcurl)*
- void **VSIInstallSwiftFileHandler** (void)  
*Install /vsiswift/ OpenStack Swif Object Storage (Swift) file system handler (requires libcurl)*
- void **VSIInstallSwiftStreamingFileHandler** (void)  
*Install /vsiswift\_streamin/ OpenStack Swif Object Storage (Swift) 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.*
- void **VSIInstallCryptFileHandler** (void)  
*Install /vsicrypt/ encrypted file system handler (requires libcrypto++)*
- void **VSISetCryptKey** (const **GByte** \*pabyKey, int nKeySize)
- **VSIFILE** \* **VSIFileFromMemBuffer** (const char \*pszFilename, **GByte** \*pabyData, **vsi\_I\_offset** nData↵  
Length, int bTakeOwnership)  
*Create memory "file" from a buffer.*
- **GByte** \* **VSIGetMemFileBuffer** (const char \*pszFilename, **vsi\_I\_offset** \*pnDataLength, int bUnlinkAnd↵  
Seize)  
*Fetch buffer underlying memory file.*
- void **VSIStdoutSetRedirection** ( **VSIWriteFunction** pFct, FILE \*stream)

### 12.13.1 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.

It 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.

### 12.13.2 Macro Definition Documentation

#### 12.13.2.1 CPLReadDir

```
#define CPLReadDir  VSIReadDir
```

Alias of **VSIReadDir()** (p. ??)

#### 12.13.2.2 VSI\_CALLOC\_VERBOSE

```
#define VSI_CALLOC_VERBOSE(
    nCount,
    nSize )  VSICallocVerbose(nCount,nSize, __FILE__, __LINE__)
```

VSI\_CALLOC\_VERBOSE

Referenced by CPLVirtualMemDerivedNew(), CSLAddStringMayFail(), OGRPolyhedralSurface::importFrom↵ Wkb(), OGRFeature::OGRFeature(), OGRGeometryCollection::OGRGeometryCollection(), and OGRSimple↵ Curve::transform().

#### 12.13.2.3 VSI\_ISDIR

```
#define VSI_ISDIR(
    x )  S_ISDIR(x)
```

Test if the file is a directory

Referenced by VSIMkdirRecursive(), VSIReadDirRecursive(), and VSIRmdirRecursive().

#### 12.13.2.4 VSI\_ISLNK

```
#define VSI_ISLNK(  
    x ) S_ISLNK(x)
```

Test if the file is a symbolic link

#### 12.13.2.5 VSI\_ISREG

```
#define VSI_ISREG(  
    x ) S_ISREG(x)
```

Test if the file is a regular file

Referenced by VSIReadDirRecursive().

#### 12.13.2.6 VSI\_L\_OFFSET\_MAX

```
#define VSI_L_OFFSET_MAX  GUINTBIG_MAX
```

Maximum value for a file offset

#### 12.13.2.7 VSI\_MALLOC2\_VERBOSE

```
#define VSI_MALLOC2_VERBOSE(  
    nSize1,  
    nSize2 ) VSIMalloc2Verbose(nSize1,nSize2, __FILE__, __LINE__)
```

VSI\_MALLOC2\_VERBOSE

#### 12.13.2.8 VSI\_MALLOC3\_VERBOSE

```
#define VSI_MALLOC3_VERBOSE(  
    nSize1,  
    nSize2,  
    nSize3 ) VSIMalloc3Verbose(nSize1,nSize2,nSize3, __FILE__, __LINE__)
```

VSI\_MALLOC3\_VERBOSE

#### 12.13.2.9 VSI\_MALLOC\_ALIGNED\_AUTO\_VERBOSE

```
#define VSI_MALLOC_ALIGNED_AUTO_VERBOSE(  
    size ) VSIMallocAlignedAutoVerbose(size, __FILE__, __LINE__)
```

**VSIMallocAlignedAutoVerbose()** (p. ??) with FILE and LINE reporting

**12.13.2.10 VSI\_MALLOC\_VERBOSE**

```
#define VSI_MALLOC_VERBOSE(
    size ) VSIMallocVerbose(size, __FILE__, __LINE__)
```

**VSI\_MALLOC\_VERBOSE**

Referenced by CPLFormCIFilename(), CPLGetCurrentDir(), OGRSimpleCurve::exportToWkt(), OGRPolygon↔::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRFeature::OGRFeature(), OGRFeature::SetField(), CPL↔WorkerThreadPool::SubmitJob(), CPLWorkerThreadPool::SubmitJobs(), and OGRSimpleCurve::transform().

**12.13.2.11 VSI\_REALLOC\_VERBOSE**

```
#define VSI_REALLOC_VERBOSE(
    pOldPtr,
    nNewSize ) VSIReallocVerbose(pOldPtr, nNewSize, __FILE__, __LINE__)
```

**VSI\_REALLOC\_VERBOSE**

Referenced by OGRGeometryCollection::addGeometryDirectly(), OGRPolyhedralSurface::addGeometryDirectly(), CSLAddStringMayFail(), and OGRSimpleCurve::setNumPoints().

**12.13.2.12 VSI\_STAT\_EXISTS\_FLAG**

```
#define VSI_STAT_EXISTS_FLAG 0x1
```

Flag provided to **VSIStatExL()** (p. ??) to test if the file exists

Referenced by CPLFormCIFilename(), and VSIStatExL().

**12.13.2.13 VSI\_STAT\_NATURE\_FLAG**

```
#define VSI_STAT_NATURE_FLAG 0x2
```

Flag provided to **VSIStatExL()** (p. ??) to query the nature (file/dir) of the file

Referenced by VSIStatExL().

**12.13.2.14 VSI\_STAT\_SET\_ERROR\_FLAG**

```
#define VSI_STAT_SET_ERROR_FLAG 0x8
```

Flag provided to **VSIStatExL()** (p. ??) to issue a VSLError in case of failure

## 12.13.2.15 VSI\_STAT\_SIZE\_FLAG

```
#define VSI_STAT_SIZE_FLAG 0x4
```

Flag provided to **VSIStatExL()** (p. ??) to query the file size

Referenced by VSIStatExL().

## 12.13.2.16 VSI\_STRDUP\_VERBOSE

```
#define VSI_STRDUP_VERBOSE(  
    pszStr )  VSIStrdupVerbose(pszStr, __FILE__, __LINE__)
```

VSI\_STRDUP\_VERBOSE

Referenced by OGRFeature::CopySelfTo(), CSLAddStringMayFail(), OGRFeature::GetFieldAsString(), OGRFeature::SetField(), OGRFeature::SetNativeData(), OGRFeature::SetNativeMediaType(), and OGRFeature::SetStyleString().

## 12.13.3 Typedef Documentation

## 12.13.3.1 vsi\_l\_offset

```
typedef GUIntBig  vsi_l_offset
```

Type for a file offset

## 12.13.3.2 VSILFILE

```
typedef FILE  VSILFILE
```

Opaque type for a FILE that implements the **VSIVirtualHandle** (p. ??) API

## 12.13.3.3 VSIStatBufL

```
typedef struct stat64  VSIStatBufL
```

Type for **VSIStatL()** (p. ??)

## 12.13.3.4 VSIWriteFunction

```
typedef size_t(* VSIWriteFunction) (const void *ptr, size_t size, size_t nmemb, FILE *stream)
```

Callback used by **VSIStdoutSetRedirection()** (p. ??)

## 12.13.4 Enumeration Type Documentation

## 12.13.4.1 VSIRangeStatus

```
enum  VSIRangeStatus
```

Range status

## Enumerator

VSI_RANGE_STATUS_UNKNOWN	Unknown
VSI_RANGE_STATUS_DATA	Data present
VSI_RANGE_STATUS_HOLE	Hole

## 12.13.5 Function Documentation

### 12.13.5.1 CPLGetPhysicalRAM()

```
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().

### 12.13.5.2 CPLGetUsablePhysicalRAM()

```
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().

### 12.13.5.3 VSICalloc()

```
void* VSICalloc (
    size_t nCount,
    size_t nSize )
```

Analog of calloc(). Use **VSIFree()** (p. ??) to free

Referenced by VSICallocVerbose().

### 12.13.5.4 VSICallocVerbose()

```
void* VSICallocVerbose (
    size_t nCount,
    size_t nSize,
    const char * pszFile,
    int nLine )
```

VSICallocVerbose

References CPLError(), and VSICalloc().

### 12.13.5.5 VSICurlClearCache()

```
void VSICurlClearCache (
    void )
```

Clean local cache associated with /vsicurl/ (and related file systems)

/vsicurl (and related file systems like /vsi3/, /vsigs/, /vsiaz/, /vsioss/, /vsiswift/) cache a number of metadata and data for faster execution in read-only scenarios. But when the content on the server-side may change during the same process, those mechanisms can prevent opening new files, or give an outdated version of them.

Since

GDAL 2.2.1

References CPL\_ARRAYSIZE.

### 12.13.5.6 VSIFCloseL()

```
int VSIFCloseL (
    VSIFILE * fp )
```

Close file.

This function closes the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fclose() function.

**Parameters**

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??). Passing a nullptr produces undefined behavior.
-----------	--

**Returns**

0 on success or -1 on failure.

References VSIVirtualHandle::Close().

Referenced by VSISubFileHandle::Close(), CPLIsMachineForSureGCEInstance(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::importFromDict(), OGRSpatialReference::importFromPCI(), and OGRSpatialReference::SetFromUserInput().

**12.13.5.7 VSIFEOF()**

```
int VSIFEOF (
    VSILFILE * fp )
```

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful **VSIFSeekL()** (p. ??) call.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

**Parameters**

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

**Returns**

TRUE if at EOF else FALSE.

References VSIVirtualHandle::Eof().

**12.13.5.8 VSIFFLUSH()**

```
int VSIFFLUSH (
    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 VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX `fflush()` call.

#### Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

#### Returns

0 on success or -1 on error.

References `VSIVirtualHandle::Flush()`.

#### 12.13.5.9 VSIFGetNativeFileDescriptorL()

```
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

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

#### Returns

the native file descriptor, or NULL.

References `VSIVirtualHandle::GetNativeFileDescriptor()`.

#### 12.13.5.10 VSIFGetRangeStatusL()

```
VSIRangeStatus VSIFGetRangeStatusL (
    VSILFILE * fp,
    vsi_l_offset nOffset,
    vsi_l_offset nLength )
```

Return if a given file range contains data or holes filled with zeroes.

This uses the filesystem capabilities of querying which regions of a sparse file are allocated or not. This is currently only implemented for Linux (and no other Unix derivatives) and Windows.

Note: A return of VSI\_RANGE\_STATUS\_DATA doesn't exclude that the extent is filled with zeroes! It must be interpreted as "may contain non-zero data".

#### Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
<i>nOffset</i>	offset of the start of the extent.
<i>nLength</i>	extent length.

#### Returns

extent status: VSI\_RANGE\_STATUS\_UNKNOWN, VSI\_RANGE\_STATUS\_DATA or VSI\_RANGE\_STATUS\_HOLE

#### Since

GDAL 2.2

References VSIVirtualHandle::GetRangeStatus().

#### 12.13.5.11 VSIFileFromMemBuffer()

```
VSILFILE* VSIFileFromMemBuffer (
    const char * pszFilename,
    GByte * pabyData,
    vsi_l_offset nDataLength,
    int bTakeOwnership )
```

Create memory "file" from a buffer.

A virtual memory file is created from the passed buffer with the indicated filename. Under normal conditions the filename would need to be absolute and within the /vsimem/ portion of the filesystem.

If *bTakeOwnership* is TRUE, then the memory file system handler will take ownership of the buffer, freeing it when the file is deleted. Otherwise it remains the responsibility of the caller, but should not be freed as long as it might be accessed as a file. In no circumstances does this function take a copy of the *pabyData* contents.

#### Parameters

<i>pszFilename</i>	the filename to be created.
<i>pabyData</i>	the data buffer for the file.
<i>nDataLength</i>	the length of buffer in bytes.
<i>bTakeOwnership</i>	TRUE to transfer "ownership" of buffer or FALSE.

**Returns**

open file handle on created file (see **VSIFOpenL()** (p. ??)).

References **VSIInstallMemFileHandler()**.

**12.13.5.12 VSIFOpenExL()**

```
VSILFILE* VSIFOpenExL (
    const char * pszFilename,
    const char * pszAccess,
    int bSetError )
```

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 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, restoring 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**

<i>pszFilename</i>	the file to open. UTF-8 encoded.
<i>pszAccess</i>	access requested (i.e. "r", "r+", "w")
<i>bSetError</i>	flag determining whether or not this open call should set <b>VSIErrors</b> on failure.

**Returns**

NULL on failure, or the file handle.

**Since**

GDAL 2.1

References **CPLStrnlen()**.

Referenced by **VSIFOpenL()**.

### 12.13.5.13 VSIFOpenL()

```
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, restoring the pre-1.8.0 behavior of **VSIFOpenL()** (p. ??).

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

#### Parameters

<i>pszFilename</i>	the file to open. UTF-8 encoded.
<i>pszAccess</i>	access requested (i.e. "r", "r+", "w")

#### Returns

NULL on failure, or the file handle.

References VSIFOpenExL().

Referenced by CPLCopyFile(), CPLIsMachineForSureGCEInstance(), CPLSerializeXMLTreeToFile(), CSL↵Load2(), CSLSave(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::importFromDict(), OGR↵SpatialReference::importFromPCI(), VSISparseFileHandle::Read(), CPLJSONDocument::Save(), OGRSpatial↵Reference::SetFromUserInput(), and VSIIngestFile().

### 12.13.5.14 VSIFPrintfL()

```
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 **VSI↵FWriteL()** (p. ??).

Analog of the POSIX fprintf() call.

## Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
<i>pszFormat</i>	the printf() style format string.

## Returns

the number of bytes written or -1 on an error.

References CPLString::vPrintf(), and VSIFWriteL().

## 12.13.5.15 VSIFPutcL()

```
int VSIFPutcL (
    int nChar,
    VSILFILE * fp )
```

Write a single byte to the file.

Writes the character nChar, cast to an unsigned char, to file.

Almost an analog of the POSIX fputc() call, except that it returns the number of character written (1 or 0), and not the (cast) character itself or EOF.

## Parameters

<i>nChar</i>	character to write.
<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).

## Returns

1 in case of success, 0 on error.

References VSIFWriteL().

## 12.13.5.16 VSIFReadL()

```
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 VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

**Parameters**

<i>pBuffer</i>	the buffer into which the data should be read (at least <code>nCount * nSize</code> bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.
<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).

**Returns**

number of objects successfully read.

References `VSIVirtualHandle::Read()`.

Referenced by `VSISubFileHandle::Read()`, `VSISparseFileHandle::Read()`, and `OGRSpatialReference::SetFromUserInput()`.

**12.13.5.17 VSIFReadMultiRangeL()**

```
int VSIFReadMultiRangeL (
    int nRanges,
    void ** ppData,
    const vsi_l_offset * panOffsets,
    const size_t * panSizes,
    VSILFILE * fp )
```

Read several ranges of bytes from file.

Reads `nRanges` objects of `panSizes[i]` bytes from the indicated file at the offset `panOffsets[i]` into the buffer `ppData[i]`.

Ranges must be sorted in ascending start offset, and must not overlap each other.

This method goes through the `VSIFFileHandler` virtualization and may work on unusual filesystems such as in memory or `/vsicurl/`.

**Parameters**

<i>nRanges</i>	number of ranges to read.
<i>ppData</i>	array of <code>nRanges</code> buffer into which the data should be read ( <code>ppData[i]</code> must be at list <code>panSizes[i]</code> bytes).
<i>panOffsets</i>	array of <code>nRanges</code> offsets at which the data should be read.
<i>panSizes</i>	array of <code>nRanges</code> sizes of objects to read (in bytes).
<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).

**Returns**

0 in case of success, -1 otherwise.

Since

GDAL 1.9.0

References VSIVirtualHandle::ReadMultiRange().

#### 12.13.5.18 VSIFree()

```
void VSIFree (
    void * pData )
```

Analog of free() for data allocated with **VSIMalloc()** (p. ??), **VSICalloc()** (p. ??), **VSIRealloc()** (p. ??)

Referenced by CPLRealloc(), CPLSerializeXMLTree(), CSLAddStringMayFail(), OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), CPLWorkerThreadPool::SubmitJob(), CPLWorkerThreadPool::SubmitJobs(), OGRSimpleCurve::transform(), and VSIFreeAligned().

#### 12.13.5.19 VSIFreeAligned()

```
void VSIFreeAligned (
    void * ptr )
```

Free a buffer allocated with **VSIMallocAligned()** (p. ??).

Parameters

<i>ptr</i>	Buffer to free.
------------	-----------------

Since

GDAL 2.2

References VSIFree().

#### 12.13.5.20 VSIFseekL()

```
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.

Caution: `vsi_l_offset` is a unsigned type, so `SEEK_CUR` can only be used for positive seek. If negative seek is needed, use `VSIFSeekL( fp, VSIFTellL(fp) + negative_offset, SEEK_SET )`.

#### Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
<i>nOffset</i>	offset in bytes.
<i>nWhence</i>	one of <code>SEEK_SET</code> , <code>SEEK_CUR</code> or <code>SEEK_END</code> .

#### Returns

0 on success or -1 on failure.

References `VSIVirtualHandle::Seek()`.

Referenced by `VSISparseFileHandle::Read()`, and `VSISubFileHandle::Seek()`.

#### 12.13.5.21 VSIFTellL()

```
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

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

#### Returns

file offset in bytes.

References `VSIVirtualHandle::Tell()`.

Referenced by `VSISubFileHandle::Read()`, `VSISubFileHandle::Tell()`, and `VSISubFileHandle::Write()`.



## 12.13.5.22 VSIFTruncateL()

```
int VSIFTruncateL (
    VSILFILE * fp,
    vsi_l_offset nNewSize )
```

Truncate/expand the file to the specified size.

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftruncate() call.

## Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
<i>nNewSize</i>	new size in bytes.

## Returns

0 on success

## Since

GDAL 1.9.0

References VSIVirtualHandle::Truncate().

## 12.13.5.23 VSIFWriteL()

```
size_t VSIFWriteL (
    const void * pBuffer,
    size_t nSize,
    size_t nCount,
    VSILFILE * fp )
```

Write bytes to file.

Writes nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

## Parameters

<i>pBuffer</i>	the buffer from which the data should be written (at least nCount * nSize bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.
<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).

**Returns**

number of objects successfully written.

References VSIVirtualHandle::Write().

Referenced by VSIFPrintFL(), VSIFPutcL(), and VSISubFileHandle::Write().

**12.13.5.24 VSIGetActualURL()**

```
const char* VSIGetActualURL (
    const char * pszFilename )
```

Returns the actual URL of a supplied filename.

Currently only returns a non-NULL value for network-based virtual file systems. For example "/vsi3/bucket/filename" will be expanded as "https://bucket.s3.amazon.com/filename"

Note that the lifetime of the returned string, is short, and may be invalidated by any following GDAL functions.

**Parameters**

<i>pszFilename</i>	the path of the filesystem object. UTF-8 encoded.
--------------------	---

**Returns**

the actual URL corresponding to the supplied filename, or NULL. Should not be freed.

**Since**

GDAL 2.3

**12.13.5.25 VSIGetDiskFreeSpace()**

```
GIntBig VSIGetDiskFreeSpace (
    const char * pszDirname )
```

Return free disk space available on the filesystem.

This function returns the free disk space available on the filesystem.

**Parameters**

<i>pszDirname</i>	a directory of the filesystem to query.
-------------------	---

**Returns**

The free space in bytes. Or -1 in case of error.

**Since**

GDAL 2.1

**12.13.5.26 VSIGetFileSystemOptions()**

```
const char* VSIGetFileSystemOptions (
    const char * pszFilename )
```

Return the list of options associated with a virtual file system handler as a serialized XML string.

Those options may be set as configuration options with **CPLSetConfigOption()** (p. ??).

**Parameters**

<i>pszFilename</i>	a filename, or prefix of a virtual file system handler.
--------------------	---

**Returns**

a string, which must not be freed, or NULL if no options is declared.

**Since**

GDAL 2.3

**12.13.5.27 VSIGetFileSystemsPrefixes()**

```
char** VSIGetFileSystemsPrefixes (
    void )
```

Return the list of prefixes for virtual file system handlers currently registered.

Typically: "", "/vsimem/", "/vsicurl/", etc

**Returns**

a NULL terminated list of prefixes. Must be freed with **CSLDestroy()** (p. ??)

**Since**

GDAL 2.3

## 12.13.5.28 VSIGetMemFileBuffer()

```

GByte* VSIGetMemFileBuffer (
    const char * pszFilename,
    vsi_l_offset * pnDataLength,
    int bUnlinkAndSeize )

```

Fetch buffer underlying memory file.

This function returns a pointer to the memory buffer underlying a virtual "in memory" file. If bUnlinkAndSeize is TRUE the filesystem object will be deleted, and ownership of the buffer will pass to the caller otherwise the underlying file will remain in existence.

## Parameters

<i>pszFilename</i>	the name of the file to grab the buffer of.
<i>pnDataLength</i>	(file) length returned in this variable.
<i>bUnlinkAndSeize</i>	TRUE to remove the file, or FALSE to leave unaltered.

## Returns

pointer to memory buffer or NULL on failure.

References CPLDebug().

Referenced by CPLHTTPFetchEx().

## 12.13.5.29 VSIGetSignedURL()

```

char* VSIGetSignedURL (
    const char * pszFilename,
    CSLConstList papszOptions )

```

Returns a signed URL of a supplied filename.

Currently only returns a non-NULL value for /vsi3/, /vsigs/, /vsiaz/ and /vsioss/. For example "/vsi3/bucket/filename" will be expanded as "https://bucket.s3.amazonaws.com/filename?X-Amz-Algorithm=AWS4-HMAC-SHA256..." Configuration options that apply for file opening (typically to provide credentials), and are returned by **VSIGetFileSystemOptions()** (p. ??), are also valid in that context.

## Parameters

<i>pszFilename</i>	the path of the filesystem object. UTF-8 encoded.
<i>papszOptions</i>	list of options, or NULL. Depend on file system handler. For /vsi3/, /vsigs/, /vsiaz/ and /vsioss/, the following options are supported: <ul style="list-style-type: none"> <li>START_DATE=YYMMDDTHHMMSSZ: date and time in UTC following ISO 8601 standard, corresponding to the start of validity of the URL. If not specified, current date time.</li> <li>EXPIRATION_DELAY=number_of_seconds: number between 1 and 604800 (seven days) for the validity of the signed URL. Defaults to 3600 (one hour)</li> <li>VERB=GET/HEAD/DELETE/PUT/POST: HTTP VERB for which the request will be used. Default to GET.</li> </ul>

/vsiaz/ supports additional options:

- SIGNEDIDENTIFIER=value: to relate the given shared access signature to a corresponding stored access policy.
- SIGNEDPERMISSIONS=r|w: permissions associated with the shared access signature. Normally deduced from VERB.

#### Returns

a signed URL, or NULL. Should be freed with **CPLFree()** (p. ??).

#### Since

GDAL 2.3

#### 12.13.5.30 VSIHasOptimizedReadMultiRange()

```
int VSIHasOptimizedReadMultiRange (
    const char * pszPath )
```

Returns if the filesystem supports efficient multi-range reading.

Currently only returns TRUE for /vsicurl/ and derived file systems.

#### Parameters

<i>pszPath</i>	the path of the filesystem object to be tested. UTF-8 encoded.
----------------	--

#### Returns

TRUE if the file system is known to have an efficient multi-range reading.

#### Since

GDAL 2.3

#### 12.13.5.31 VSIIngestFile()

```
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

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
<i>pszFilename</i>	filename.
<i>ppabyRet</i>	pointer to the target buffer. *ppabyRet must be freed with <b>VSIFree()</b> (p. ??)
<i>pnSize</i>	pointer to variable to store the file size. May be NULL.
<i>nMaxSize</i>	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()`, and `VSIFOpenL()`.

Referenced by `CPLParseXMLFile()`, and `CPLJSONDocument::Load()`.

#### 12.13.5.32 VSInstallAzureFileHandler()

```
void VSInstallAzureFileHandler (
    void )
```

Install /vsiaz/ Microsoft Azure Blob file system handler (requires libcurl)

#### See also

/vsiaz/ documentation

#### Since

GDAL 2.3

**12.13.5.33 VSInstallAzureStreamingFileHandler()**

```
void VSInstallAzureStreamingFileHandler (
    void )
```

Install /vsiaz\_streaming/ Microsoft Azure Blob file system handler (requires libcurl)

**See also**

/vsiaz\_streaming/ documentation1

**Since**

GDAL 2.3

**12.13.5.34 VSInstallCryptFileHandler()**

```
void VSInstallCryptFileHandler (
    void )
```

Install /vsicrypt/ encrypted file system handler (requires libcrypto++)

A special file handler is installed that allows reading/creating/update encrypted files on the fly, with random access capabilities.

The cryptographic algorithms used are `block ciphers`, with symmetric key.

In their simplest form, recognized filenames are of the form /vsicrypt/absolute\_path/to/file, /vsicrypt/c:/absolute\_↔ path/to/file or /vsicrypt/relative/path/to/file.

Options can also be used with the following format : /vsicrypt/option1=val1,option2=val2,...,file=/path/to/file

They can also be passed as configuration option/environment variable, because in some use cases, the syntax with option in the filename might not properly work with some drivers.

In all modes, the encryption key must be provided. There are several ways of doing so :

- By adding a `key=` parameter to the filename, like /vsicrypt/key=my\_secret\_key,file=/path/to/file. Note that this restricts the key to be in text format, whereas at its full power, it can be binary content.
- By adding a `key_b64=` parameter to the filename, to specify a binary key expressed in Base64 encoding, like /vsicrypt/key\_b64=th1sl00kslikebase64=,file=/path/to/file.
- By setting the `VSCRYPT_KEY` configuration option. The key should be in text format.
- By setting the `VSCRYPT_KEY_B64` configuration option. The key should be encoded in Base64.
- By using the **VSISetCryptKey()** (p. ??) C function.

When creating a file, if `key=GENERATE_IT` or `VSICRYPT_KEY=GENERATE_IT` is passed, the encryption key will be generated from the pseudo-random number generator of the operating system. The key will be displayed on the standard error stream in a Base64 form (unless the `VSICRYPT_DISPLAY_GENERATED_KEY` configuration option is set to OFF), and the `VSICRYPT_KEY_B64` configuration option will also be set with the Base64 form of the key (so that `CPLGetConfigOption("VSICRYPT_KEY_B64", NULL)` can be used to get it back).

The available options are :

- `alg=AES/Blowfish/Camellia/CAST256/DES_EDE2/DES_EDE3/MARS/IDEA/RC5/RC6/Serpent/SHACAL2/SKIPJACK/Twofish/XTEA`: to specify the `block cipher` algorithm. The default is AES. Only used on creation. Ignored otherwise. Note: depending on how GDAL is build, if linked against the DLL version of `libcrypto++`, only a subset of those algorithms will be available, namely AES, DES\_EDE2, DES\_EDE3 and SKIPJACK. Also available as `VSICRYPT_ALG` configuration option.
- `mode=CBC/CFB/OFB/CTR/CBC_CTS`: to specify the `block cipher mode of operation`. The default is CBC. Only used on creation. Ignored otherwise. Also available as `VSICRYPT_MODE` configuration option.
- `key=text_key`: see above.
- `key_b64=base64_encoded_key`: see above.
- `freertext=some_text`: to specify a text content that will be written *unencrypted* in the file header, for informational purposes. Default to empty. Only used on creation. Ignored otherwise. Also available as `VSICRYPT_FREETEXT` configuration option.
- `sector_size=int_value`: to specify the size of the "sector", which is the unit chunk of information that is encrypted/decrypted. Default to 512 bytes. The valid values depend on the algorithm and block cipher mode of operation. Only used on creation. Ignored otherwise. Also available as `VSICRYPT_SECTOR_SIZE` configuration option.
- `iv=initial_vector_as_text`: to specify the Initial Vector. This is an advanced option that should generally *NOT* be used. It is only useful to get completely deterministic output given the plaintext, key and other parameters, which in general *NOT* what you want to do. By default, a random initial vector of the appropriate size will be generated for each new file created. Only used on creation. Ignored otherwise. Also available as `VSICRYPT_IV` configuration option.
- `add_key_check=YES/NO`: whether a special value should be encrypted in the header, so as to be quickly able to determine if the decryption key is correct. Defaults to NO. Only used on creation. Ignored otherwise. Also available as `VSICRYPT_ADD_KEY_CHECK` configuration option.
- `file=filename`. To specify the filename. This must be the last option put in the option list (so as to make it possible to use filenames with comma in them. )

This special file handler can be combined with other virtual filesystems handlers, such as `/vszip`. For example, `/vsicrypt/vsicurl/path/to/remote/encrypted/file.tif`

Implementation details:

The structure of encrypted files is the following: a header, immediately followed by the encrypted payload (by sectors, i.e. chunks of `sector_size` bytes).

The header structure is the following :

1. 8 bytes. Signature. Fixed value: VSICRYPT.
2. `UINT16_LE`. Header size (including previous signature bytes).
3. `UINT8`. Format major version. Current value: 1.
4. `UINT8`. Format minor version. Current value: 0.



5. UINT16. Sector size.
6. UINT8. Cipher algorithm. Valid values are: 0 = AES (Rijndael), 1 = Blowfish, 2 = Camellia, 3 = CAST256, 4 = DES\_EDE2, 5 = DES\_EDE3, 6 = MARS, 7 = IDEA, 8 = RC5, 9 = RC6, 10 = Serpent, 11 = SHACAL2, 12 = SKIPJACK, 13 = Twofish, 14 = XTEA.
7. UINT8. Block cipher mode of operation. Valid values are: 0 = CBC, 1 = CFB, 2 = OFB, 3 = CTR, 4 = CBC\_CTS.
8. UINT8. Size in bytes of the Initial Vector.
9. N bytes with the content of the Initial Vector, where N is the value of the previous field.
10. UINT16\_LE. Size in bytes of the free text.
11. N bytes with the content of the free text, where N is the value of the previous field.
12. UINT8. Size in bytes of encrypted content (key check), or 0 if key check is absent.
13. N bytes with encrypted content (key check), where N is the value of the previous field.
14. UINT64\_LE. Size of the unencrypted file, in bytes.
15. UINT16\_LE. Size in bytes of extra content (of unspecified semantics). For v1.0, fixed value of 0
16. N bytes with extra content (of unspecified semantics), where N is the value of the previous field.

This design does not provide any means of authentication or integrity check.

Each sector is encrypted/decrypted independently of other sectors. For that, the Initial Vector contained in the header is XOR'ed with the file offset (relative to plain text file) of the start of the sector being processed, as a 8-byte integer. More precisely, the first byte of the main IV is XOR'ed with the 8 least-significant bits of the sector offset, the second byte of the main IV is XOR'ed with the following 8 bits of the sector offset, etc... until the 8th byte.

This design could potentially be prone to chosen-plaintext attack, for example if the attacker managed to get (part of) an existing encrypted file to be encrypted from plaintext he might have selected.

Note: if "hostile" code can explore process content, or attach to it with a debugger, it might be relatively easy to retrieve the encryption key. A GDAL plugin could for example get the content of configuration options, or list opened datasets and see the key/key\_b64 values, so disabling plugin loading might be a first step, as well as linking statically GDAL to application code. If plugin loading is enabled or GDAL dynamically linked, using **VSISetCryptKey()** (p. ??) to set the key might make it a bit more complicated to spy the key. But, as said initially, this is in no way a perfect protection.

Since

GDAL 2.1.0

References CPLSPrintf().

### 12.13.5.35 VSIInstallCurlFileHandler()

```
void VSIInstallCurlFileHandler (
    void )
```

Install /vsicurl/ HTTP/FTP file system handler (requires libcurl)

See also

/vsicurl/ documentation

Since

GDAL 1.8.0

References CPLAtGIntBig(), and CPLGetConfigOption().

#### 12.13.5.36 VSInstallCurlStreamingFileHandler()

```
void VSInstallCurlStreamingFileHandler (
    void )
```

Install /vsicurl\_streaming/ HTTP/FTP file system handler (requires libcurl).

##### See also

/vsicurl\_streaming/ documentation

##### Since

GDAL 1.10

#### 12.13.5.37 VSInstallGSFileHandler()

```
void VSInstallGSFileHandler (
    void )
```

Install /vsigs/ Google Cloud Storage file system handler (requires libcurl)

##### See also

/vsigs/ documentation

##### Since

GDAL 2.2

#### 12.13.5.38 VSInstallGSStreamingFileHandler()

```
void VSInstallGSStreamingFileHandler (
    void )
```

Install /vsigs\_streaming/ Google Cloud Storage file system handler (requires libcurl)

##### See also

/vsigs\_streaming/ documentation

##### Since

GDAL 2.2

**12.13.5.39 VSInstallGZipFileHandler()**

```
void VSInstallGZipFileHandler (
    void )
```

Install GZip file system handler.

A special file handler is installed that allows reading on-the-fly and writing in GZip (.gz) files.

All portions of the file system underneath the base path "/vsigzip/" will be handled by this driver.

Additional documentation is to be found at: <http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip>

Since

GDAL 1.6.0

**12.13.5.40 VSInstallMemFileHandler()**

```
void VSInstallMemFileHandler (
    void )
```

Install "memory" file system handler.

A special file handler is installed that allows block of memory to be treated as files. All portions of the file system underneath the base path "/vsimem/" will be handled by this driver.

Normal VSI\*L functions can be used freely to create and destroy memory arrays treating them as if they were real file system objects. Some additional methods exist to efficient create memory file system objects without duplicating original copies of the data or to "steal" the block of memory associated with a memory file.

Directory related functions are supported.

This code example demonstrates using GDAL to translate from one memory buffer to another.

```
GByte *ConvertBufferFormat( GByte *pabyInData, vsi_l_offset nInDataLength,
                           vsi_l_offset *pnOutDataLength )
{
    // create memory file system object from buffer.
    VSIFCloseL( VSIFFileFromMemBuffer( "/vsimem/work.dat", pabyInData,
                                       nInDataLength, FALSE ) );

    // Open memory buffer for read.
    GDALDatasetH hDS = GDALOpen( "/vsimem/work.dat", GA_ReadOnly );

    // Get output format driver.
    GDALDriverH hDriver = GDALGetDriverByName( "GTiff" );
    GDALDatasetH hOutDS;

    hOutDS = GDALCreateCopy( hDriver, "/vsimem/out.tif", hDS, TRUE, NULL,
                           NULL, NULL );

    // close source file, and "unlink" it.
    GDALClose( hDS );
    VSIUnlink( "/vsimem/work.dat" );

    // seize the buffer associated with the output file.
    return VSIGetMemFileBuffer( "/vsimem/out.tif", pnOutDataLength, TRUE );
}
```

Referenced by VSIFFileFromMemBuffer().

#### 12.13.5.41 VSInstallOSSFileHandler()

```
void VSInstallOSSFileHandler (
    void )
```

Install /vsioss/ Alibaba Cloud Object Storage Service (OSS) file system handler (requires libcurl)

##### See also

[/vsioss/ documentation](#)

##### Since

GDAL 2.3

#### 12.13.5.42 VSInstallOSSStreamingFileHandler()

```
void VSInstallOSSStreamingFileHandler (
    void )
```

Install /vsioss\_streaming/ Alibaba Cloud Object Storage Service (OSS) file system handler (requires libcurl)

##### See also

[/vsioss\\_streaming/ documentation](#)

##### Since

GDAL 2.3

#### 12.13.5.43 VSInstallS3FileHandler()

```
void VSInstallS3FileHandler (
    void )
```

Install /vsi3/ Amazon S3 file system handler (requires libcurl)

##### See also

[/vsi3/ documentation](#)

##### Since

GDAL 2.1

#### 12.13.5.44 VSInstallS3StreamingFileHandler()

```
void VSInstallS3StreamingFileHandler (
    void )
```

Install /vsi3\_streaming/ Amazon S3 file system handler (requires libcurl).

##### See also

/vsi3\_streaming/ documentation

##### Since

GDAL 2.1

#### 12.13.5.45 VSInstallSparseFileHandler()

```
void VSInstallSparseFileHandler (
    void )
```

Install /vsisparsed/ virtual file handler.

##### See also

/vsisparsed/ documentation

#### 12.13.5.46 VSInstallStdinHandler()

```
void VSInstallStdinHandler (
    void )
```

Install /vsistdin/ file system handler.

A special file handler is installed that allows reading from the standard input stream.

The file operations available are of course limited to Read() and forward Seek() (full seek in the first MB of a file).

##### Since

GDAL 1.8.0

#### 12.13.5.47 VSInstallStdoutHandler()

```
void VSInstallStdoutHandler (
    void )
```

Install /vsistdout/ file system handler.

A special file handler is installed that allows writing to the standard output stream.

The file operations available are of course limited to Write().

A variation of this file system exists as the /vsistdout\_redirect/ file system handler, where the output function can be defined with **VSStdoutSetRedirection()** (p. ??).

Since

GDAL 1.8.0

#### 12.13.5.48 VSInstallSubFileHandler()

```
void VSInstallSubFileHandler (
    void )
```

Install /vsisubfile/ virtual file handler.

See also

/vsisubfile/ documentation

#### 12.13.5.49 VSInstallSwiftFileHandler()

```
void VSInstallSwiftFileHandler (
    void )
```

Install /vsiswift/ OpenStack Swif Object Storage (Swift) file system handler (requires libcurl)

See also

/vsiswift/ documentation

Since

GDAL 2.3

**12.13.5.50 VSInstallSwiftStreamingFileHandler()**

```
void VSInstallSwiftStreamingFileHandler (
    void )
```

Install /vswift\_streamin/ OpenStack Swif Object Storage (Swift) file system handler (requires libcurl)

**See also**

/vswift\_streaming/ documentation

**Since**

GDAL 2.3

**12.13.5.51 VSInstallTarFileHandler()**

```
void VSInstallTarFileHandler (
    void )
```

Install /vsitar/ file system handler.

A special file handler is installed that allows reading on-the-fly in TAR (regular .tar, or compressed .tar.gz/.tgz) archives.

All portions of the file system underneath the base path "/vsitar/" will be handled by this driver.

The syntax to open a file inside a tar file is /vsitar/path/to/the/file.tar/path/inside/the/tar/file where 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.

Starting with GDAL 2.2, an alternate syntax is available so as to enable chaining and not being dependent on .tar extension : /vsitar/{/path/to/the/archive}/path/inside/the/tar/file. Note that /path/to/the/archive may also itself this alternate syntax.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vsitar//home/gdal/... For example gdalinfo /vsitar/myarchive.tar/subdir1/file1.tif

Syntactic sugar : if the tar archive contains only one file located at its root, just mentioning "/vsitar/path/to/the/file.<→tar" will work

**VSStatL()** (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

**12.13.5.52 VSInstallZipFileHandler()**

```
void VSInstallZipFileHandler (
    void )
```

Install ZIP file system handler.

A special file handler is installed that allows reading on-the-fly in ZIP (.zip) archives.

All portions of the file system underneath the base path "/vszip/" will be handled by this driver.

The syntax to open a file inside a zip file is /vszip/path/to/the/file.zip/path/inside/the/zip/file where path/to/the/file.zip is relative or absolute and path/inside/the/zip/file is the relative path to the file inside the archive.

Starting with GDAL 2.2, an alternate syntax is available so as to enable chaining and not being dependent on .zip extension : /vszip/{/path/to/the/archive}/path/inside/the/zip/file. Note that /path/to/the/archive may also itself use this alternate syntax.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vszip//home/gdal/... For example gdalinfo /vszip/myarchive.zip/subdir1/file1.tif

Syntactic sugar : if the .zip file contains only one file located at its root, just mentioning "/vszip/path/to/the/file.zip" will work

**VSStatL()** (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 **VSReadDir()** (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/ReadInZip>

Since

GDAL 1.6.0

**12.13.5.53 VSIsCaseSensitiveFS()**

```
int VSIsCaseSensitiveFS (
    const char * pszFilename )
```

Returns if the filenames of the filesystem are case sensitive.

This method retrieves to which filesystem belongs the passed filename and return TRUE if the filenames of that filesystem are case sensitive.

Currently, this will return FALSE only for Windows real filenames. Other VSI virtual filesystems are case sensitive.

This methods avoid ugly #ifndef WIN32 / #endif code, that is wrong when dealing with virtual filenames.



## Parameters

<i>pszFilename</i>	the path of the filesystem object to be tested. UTF-8 encoded.
--------------------	--

## Returns

TRUE if the filenames of the filesystem are case sensitive.

## Since

GDAL 1.8.0

Referenced by CPLFormCIFilename().

## 12.13.5.54 VSIMalloc()

```
void* VSIMalloc (
    size_t nSize )
```

Analog of malloc(). Use **VSIFree()** (p. ??) to free

Referenced by CPLDBCStatement::Append(), CPLDBCStatement::AppendEscaped(), CPLDebug(), VSIMalloc2Verbose(), VSIMalloc3Verbose(), VSIMallocAligned(), VSIMallocVerbose(), VSIRealloc(), and VSIStrdup().

## 12.13.5.55 VSIMalloc2()

```
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()** (p. ??) or **VSIFree()** (p. ??) can be used to free memory allocated by this function.

References VSIMalloc2Verbose().

## 12.13.5.56 VSIMalloc2Verbose()

```
void* VSIMalloc2Verbose (
    size_t nSize1,
    size_t nSize2,
    const char * pszFile,
    int nLine )
```

VSIMalloc2Verbose

References CPLERROR(), and VSIMalloc().

Referenced by VSIMalloc2().

**12.13.5.57 VSIMalloc3()**

```
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 **CPL\_Error()** (p. ??). If  $nSize1 == 0$  ||  $nSize2 == 0$  ||  $nSize3 == 0$ , a NULL pointer will also be returned. **CPLFree()** (p. ??) or **VSIFree()** (p. ??) can be used to free memory allocated by this function.

References VSIMalloc3Verbose().

**12.13.5.58 VSIMalloc3Verbose()**

```
void* VSIMalloc3Verbose (
    size_t nSize1,
    size_t nSize2,
    size_t nSize3,
    const char * pszFile,
    int nLine )
```

VSIMalloc3Verbose

References CPL\_Error(), and VSIMalloc().

Referenced by VSIMalloc3().

**12.13.5.59 VSIMallocAligned()**

```
void* VSIMallocAligned (
    size_t nAlignment,
    size_t nSize )
```

Allocates a buffer with an alignment constraint.

The return value must be freed with **VSIFreeAligned()** (p. ??).

**Parameters**

<i>nAlignment</i>	Must be a power of 2, multiple of sizeof(void*), and lesser than 256.
<i>nSize</i>	Size of the buffer to allocate.

**Returns**

a buffer aligned on nAlignment and of size nSize, or NULL

Since

GDAL 2.2

References VSIMalloc().

Referenced by VSIMallocAlignedAuto().

#### 12.13.5.60 VSIMallocAlignedAuto()

```
void* VSIMallocAlignedAuto (
    size_t nSize )
```

Allocates a buffer with an alignment constraint such that it can be used by the most demanding vector instruction set on that platform.

The return value must be freed with **VSIFreeAligned()** (p. ??).

Parameters

<i>nSize</i>	Size of the buffer to allocate.
--------------	---------------------------------

Returns

an aligned buffer of size nSize, or NULL

Since

GDAL 2.2

References VSIMallocAligned().

Referenced by VSIMallocAlignedAutoVerbose().

#### 12.13.5.61 VSIMallocAlignedAutoVerbose()

```
void* VSIMallocAlignedAutoVerbose (
    size_t nSize,
    const char * pszFile,
    int nLine )
```

See **VSIMallocAlignedAuto()** (p. ??)

References CPLError(), and VSIMallocAlignedAuto().

### 12.13.5.62 VSIMallocVerbose()

```
void* VSIMallocVerbose (
    size_t nSize,
    const char * pszFile,
    int nLine )
```

VSIMallocVerbose

References CPLError(), and VSIMalloc().

### 12.13.5.63 VSIMkdir()

```
int VSIMkdir (
    const char * pszPathname,
    long mode )
```

Create a directory.

Create a new directory with the indicated mode. The mode is ignored on some platforms. A reasonable default mode value would be 0666. This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX mkdir() function.

#### Parameters

<i>pszPathname</i>	the path to the directory to create. UTF-8 encoded.
<i>mode</i>	the permissions mode.

#### Returns

0 on success or -1 on an error.

Referenced by VSIMkdirRecursive().

### 12.13.5.64 VSIMkdirRecursive()

```
int VSIMkdirRecursive (
    const char * pszPathname,
    long mode )
```

Create a directory and all its ancestors.

#### Parameters

<i>pszPathname</i>	the path to the directory to create. UTF-8 encoded.
<i>mode</i>	the permissions mode.

**Returns**

0 on success or -1 on an error.

**Since**

GDAL 2.3

References CPLGetPath(), VSI\_ISDIR, VSIMkdir(), and VSISatL().

**12.13.5.65 VSIReadDir()**

```
char** VSIReadDir (
    const char * pszPath )
```

Read names in a directory.

This function abstracts access to directory contents. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with **CSLDestroy()** (p. ??) when no longer needed.

Note that no error is issued via **CPLError()** (p. ??) if the directory path is invalid, though NULL is returned.

This function used to be known as **CPLReadDir()** (p. ??), but the old name is now deprecated.

**Parameters**

<i>pszPath</i>	the relative, or absolute path of a directory to read. UTF-8 encoded.
----------------	---

**Returns**

The list of entries in the directory, or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

References VSIReadDirEx().

Referenced by VSIReadDirRecursive(), and VSIRmdirRecursive().

**12.13.5.66 VSIReadDirEx()**

```
char** VSIReadDirEx (
    const char * pszPath,
    int nMaxFiles )
```

Read names in a directory.

This function abstracts access to directory contents. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with **CSLDestroy()** (p. ??) when no longer needed.

Note that no error is issued via **CPLERROR()** (p. ??) if the directory path is invalid, though NULL is returned.

If nMaxFiles is set to a positive number, directory listing will stop after that limit has been reached. Note that to indicate truncate, at least one element more than the nMaxFiles limit will be returned. If **CSLCount()** (p. ??) on the result is lesser or equal to nMaxFiles, then no truncation occurred.

#### Parameters

<i>pszPath</i>	the relative, or absolute path of a directory to read. UTF-8 encoded.
<i>nMaxFiles</i>	maximum number of files after which to stop, or 0 for no limit.

#### Returns

The list of entries in the directory, or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

#### Since

GDAL 2.1

Referenced by VSIReadDir().

#### 12.13.5.67 VSIReadDirRecursive()

```
char** VSIReadDirRecursive (
    const char * pszPathIn )
```

Read names in a directory recursively.

This function abstracts access to directory contents and subdirectories. It returns a list of strings containing the names of files and directories in this directory and all subdirectories. The resulting string list becomes the responsibility of the application and should be freed with **CSLDestroy()** (p. ??) when no longer needed.

Note that no error is issued via **CPLERROR()** (p. ??) if the directory path is invalid, though NULL is returned.

#### Parameters

<i>psz↔ PathIn</i>	the relative, or absolute path of a directory to read. UTF-8 encoded.
------------------------	---

#### Returns

The list of entries in the directory and subdirectories or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

Since

GDAL 1.10.0

References CPLStringList::AddString(), CPLFree, CPLSPrintf(), CPLStrdup(), CSLCount(), CSLDestroy(), EQUAL, CPLStringList::StealList(), VSI\_ISDIR, VSI\_ISREG, VSIReadDir(), and VSISatL().

#### 12.13.5.68 VSIRealloc()

```
void* VSIRealloc (
    void * pData,
    size_t nNewSize )
```

Analog of realloc(). Use **VSIFree()** (p. ??) to free

References CPL\_FRMT\_GIB, CPL\_FRMT\_GUIB, and VSIMalloc().

Referenced by VSIReallocVerbose().

#### 12.13.5.69 VSIReallocVerbose()

```
void* VSIReallocVerbose (
    void * pOldPtr,
    size_t nNewSize,
    const char * pszFile,
    int nLine )
```

VSIReallocVerbose

References CPLError(), and VSIRealloc().

#### 12.13.5.70 VSIRename()

```
int VSIRename (
    const char * oldpath,
    const char * newpath )
```

Rename a file.

Renames a file object in the file system. It should be possible to rename a file onto a new filesystem, but it is safest if this function is only used to rename files that remain in the same directory.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rename() function.

## Parameters

<i>oldpath</i>	the name of the file to be renamed. UTF-8 encoded.
<i>newpath</i>	the name the file should be given. UTF-8 encoded.

## Returns

0 on success or -1 on an error.

Referenced by CPLMoveFile().

## 12.13.5.71 VSIRewindL()

```
void VSIRewindL (
    VSILFILE * fp )
```

Rewind the file pointer to the beginning of the file.

This is equivalent to VSIFSeekL( fp, 0, SEEK\_SET )

Analog of the POSIX rewind() call.

## Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

## 12.13.5.72 VSIRmdir()

```
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 VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rmdir() function.

## Parameters

<i>pszDirname</i>	the path of the directory to be deleted. UTF-8 encoded.
-------------------	---



**Returns**

0 on success or -1 on an error.

Referenced by VSIRmdirRecursive().

**12.13.5.73 VSIRmdirRecursive()**

```
int VSIRmdirRecursive (
    const char * pszDirname )
```

Delete a directory recursively.

Deletes a directory object and its content from the file system.

**Returns**

0 on success or -1 on an error.

**Since**

GDAL 2.3

References CPLFormFilename(), CSLDestroy(), VSI\_ISDIR, VSIReadDir(), VSIRmdir(), VSIStatL(), and VSIUnlink().

**12.13.5.74 VSISetCryptKey()**

```
void VSISetCryptKey (
    const GByte * pabyKey,
    int nKeySize )
```

Installs the encryption/decryption key.

By passing a NULL key, the previously installed key will be cleared. Note, however, that it is not guaranteed that there won't be trace of it in other places in memory or in on-disk temporary file.

**Parameters**

<i>pabyKey</i>	key. Might be NULL to clear previously set key.
<i>nKeySize</i>	length of the key in bytes. Might be 0 to clear previously set key.

**See also**

**VSIInstallCryptFileHandler()** (p. ??) for documentation on /vsicrypt/

References CPLAssert, CPLFree, and CPLMalloc().

**12.13.5.75 VSISatExL()**

```
int VSISatExL (
    const char * pszFilename,
    VSISatBufL * psStatBuf,
    int nFlags )
```

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSISatBufL structure. For portability, only use the st\_size (size in bytes) and st\_mode (file type). This method is similar to VSISat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function, with an extra parameter to specify which information is needed, which offers a potential for speed optimizations on specialized and potentially slow virtual filesystem objects (/vsigzip/, /vsicurl/)

**Parameters**

<i>pszFilename</i>	the path of the filesystem object to be queried. UTF-8 encoded.
<i>psStatBuf</i>	the structure to load with information.
<i>nFlags</i>	0 to get all information, or VSI_STAT_EXISTS_FLAG, VSI_STAT_NATURE_FLAG or VSI_STAT_SIZE_FLAG, or a combination of those to get partial info.

**Returns**

0 on success or -1 on an error.

**Since**

GDAL 1.8.0

References VSI\_STAT\_EXISTS\_FLAG, VSI\_STAT\_NATURE\_FLAG, and VSI\_STAT\_SIZE\_FLAG.

Referenced by CPLFormCIFilename(), and VSISatL().

**12.13.5.76 VSISatL()**

```
int VSISatL (
    const char * pszFilename,
    VSISatBufL * psStatBuf )
```

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSISatBufL structure. For portability, only use the st\_size (size in bytes) and st\_mode (file type). This method is similar to VSISat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function.

## Parameters

<i>pszFilename</i>	the path of the filesystem object to be queried. UTF-8 encoded.
<i>psStatBuf</i>	the structure to load with information.

## Returns

0 on success or -1 on an error.

References VSISatExL().

Referenced by CPLCheckForFile(), CPLCopyTree(), CPLDefaultFindFile(), CPLUnlinkTree(), CPLJSON↵  
Document::LoadChunks(), VSIMkdirRecursive(), VSIRmdirRecursive(), and VSIRmdirRecursive().

## 12.13.5.77 VSIStdoutSetRedirection()

```
void VSIStdoutSetRedirection (
    VSIWriteFunction pFct,
    FILE * stream )
```

Set an alternative write function and output file handle instead of fwrite() / stdout.

## Parameters

<i>pFct</i>	Function with same signature as fwrite()
<i>stream</i>	File handle on which to output. Passed to pFct.

## Since

GDAL 2.0

## 12.13.5.78 VSIStrdup()

```
char* VSIStrdup (
    const char * pszString )
```

Analog of strdup(). Use **VSIFree()** (p. ??) to free

References VSIMalloc().

Referenced by VSIStrdupVerbose().

#### 12.13.5.79 VSIStrdupVerbose()

```
char* VSIStrdupVerbose (
    const char * pszStr,
    const char * pszFile,
    int nLine )
```

VSIStrdupVerbose

References CPLError(), and VSIStrdup().

#### 12.13.5.80 VSIStrerror()

```
char* VSIStrerror (
    int nErrno )
```

Return the error string corresponding to the error number. Do not free it

#### 12.13.5.81 VSISupportsSparseFiles()

```
int VSISupportsSparseFiles (
    const char * pszPath )
```

Returns if the filesystem supports sparse files.

Only supported on Linux (and no other Unix derivatives) and Windows. On Linux, the answer depends on a few hardcoded signatures for common filesystems. Other filesystems will be considered as not supporting sparse files.

##### Parameters

<i>pszPath</i>	the path of the filesystem object to be tested. UTF-8 encoded.
----------------	--

##### Returns

TRUE if the file system is known to support sparse files. FALSE may be returned both in cases where it is known to not support them, or when it is unknown.

##### Since

GDAL 2.2

#### 12.13.5.82 VSIUnlink()

```
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

<i>pszFilename</i>	the path of the file to be deleted. UTF-8 encoded.
--------------------	--

#### Returns

0 on success or -1 on an error.

Referenced by CPLMoveFile(), and VSIRmdirRecursive().

## 12.14 cpl\_worker\_thread\_pool.h File Reference

```
#include "cpl_multiproc.h"
#include "cpl_list.h"
#include <vector>
```

### Classes

- class **CPLWorkerThreadPool**

#### 12.14.1 Detailed Description

Class to manage a pool of worker threads.

#### Since

GDAL 2.1

## 12.15 ogr\_api.h File Reference

```
#include "cpl_progress.h"
#include "cpl_minixml.h"
#include "ogr_core.h"
```

## Macros

- `#define OGR_FOR_EACH_FEATURE_BEGIN(hFeat, hLayer)`
- `#define OGR_FOR_EACH_FEATURE_END(hFeat)`

## Typedefs

- `typedef void * OGRGeometryH`
- `typedef void * OGRSpatialReferenceH`
- `typedef void * OGRCoordinateTransformationH`
- `typedef void * OGRFieldDefnH`
- `typedef void * OGRFeatureDefnH`
- `typedef void * OGRFeatureH`
- `typedef void * OGRStyleTableH`
- `typedef struct OGRGeomFieldDefnHS * OGRGeomFieldDefnH`
- `typedef void * OGRLayerH`
- `typedef void * OGRDataSourceH`
- `typedef void * OGRSFDriverH`
- `typedef void * OGRStyleMgrH`
- `typedef void * OGRStyleToolH`

## Functions

- **OGRERR OGR\_G\_CreateFromWkb** (const void \*, OGRSpatialReferenceH, OGRGeometryH \*, int)  
*Create a geometry object of the appropriate type from its well known binary representation.*
- **OGRERR OGR\_G\_CreateFromWkt** (char \*\*, OGRSpatialReferenceH, OGRGeometryH \*)  
*Create a geometry object of the appropriate type from its well known text representation.*
- **OGRERR OGR\_G\_CreateFromFgf** (const void \*, OGRSpatialReferenceH, OGRGeometryH \*, int, int \*)  
*Create a geometry object of the appropriate type from its FGF (FDO Geometry Format) binary representation.*
- **void OGR\_G\_DestroyGeometry** ( OGRGeometryH)  
*Destroy geometry object.*
- **OGRGeometryH OGR\_G\_CreateGeometry** ( OGRwkbGeometryType) CPL\_WARN\_UNUSED\_RESULT  
*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) CPL\_WARN\_UNUSED\_RESULT
- **OGRGeometryH OGR\_G\_ForceToPolygon** ( OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT  
*Convert to polygon.*
- **OGRGeometryH OGR\_G\_ForceToLineString** ( OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT  
*Convert to line string.*
- **OGRGeometryH OGR\_G\_ForceToMultiPolygon** ( OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT  
*Convert to multipolygon.*
- **OGRGeometryH OGR\_G\_ForceToMultiPoint** ( OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT  
*Convert to multipoint.*
- **OGRGeometryH OGR\_G\_ForceToMultiLineString** ( OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT  
*Convert to multilinestring.*
- **OGRGeometryH OGR\_G\_ForceTo** ( OGRGeometryH hGeom, OGRwkbGeometryType eTargetType, char \*\*papszOptions) CPL\_WARN\_UNUSED\_RESULT  
*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.*
- int **OGR\_G\_CoordinateDimension** ( OGRGeometryH)
  - Get the dimension of the coordinates in this geometry.*
- void **OGR\_G\_SetCoordinateDimension** ( OGRGeometryH, int)
  - Set the coordinate dimension.*
- int **OGR\_G\_Is3D** ( OGRGeometryH)
  - See whether this geometry has Z coordinates.*
- int **OGR\_G\_IsMeasured** ( OGRGeometryH)
  - See whether this geometry is measured.*
- void **OGR\_G\_Set3D** ( OGRGeometryH, int)
  - Add or remove the Z coordinate dimension.*
- void **OGR\_G\_SetMeasured** ( OGRGeometryH, int)
  - Add or remove the M coordinate dimension.*
- **OGRGeometryH OGR\_G\_Clone** ( OGRGeometryH) **CPL\_WARN\_UNUSED\_RESULT**
  - 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, const void \*, 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 \*) **CPL\_WARN\_UNUSED\_RESULT**
  - Create geometry from GML.*
- char \* **OGR\_G\_ExportToGML** ( OGRGeometryH) **CPL\_WARN\_UNUSED\_RESULT**
  - Convert a geometry into GML format.*
- char \* **OGR\_G\_ExportToGMLEx** ( OGRGeometryH, char \*\*papszOptions) **CPL\_WARN\_UNUSED\_RESULT**

*Convert a geometry into GML format.*

- **OGRGeometryH OGR\_G\_CreateFromGMLTree** (const **CPLXMLNode** \*) **CPL\_WARN\_UNUSED\_RESULT**
- **CPLXMLNode \* OGR\_G\_ExportToGMLTree** ( **OGRGeometryH** ) **CPL\_WARN\_UNUSED\_RESULT**
- **CPLXMLNode \* OGR\_G\_ExportEnvelopeToGMLTree** ( **OGRGeometryH** ) **CPL\_WARN\_UNUSED\_RESULT**
- **char \* OGR\_G\_ExportToKML** ( **OGRGeometryH**, const **char** \*pszAltitudeMode) **CPL\_WARN\_UNUSED\_RESULT**

*Convert a geometry into KML format.*

- **char \* OGR\_G\_ExportToJson** ( **OGRGeometryH** ) **CPL\_WARN\_UNUSED\_RESULT**

*Convert a geometry into GeoJSON format.*

- **char \* OGR\_G\_ExportToJsonEx** ( **OGRGeometryH**, **char** \*\*papszOptions) **CPL\_WARN\_UNUSED\_RESULT**

*Convert a geometry into GeoJSON format.*

- **OGRGeometryH OGR\_G\_CreateGeometryFromJson** (const **char** \*) **CPL\_WARN\_UNUSED\_RESULT**
- **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) **CPL\_WARN\_UNUSED\_RESULT**

*Compute a simplified geometry.*

- **OGRGeometryH OGR\_G\_SimplifyPreserveTopology** ( **OGRGeometryH** hThis, double tolerance) **CPL\_WARN\_UNUSED\_RESULT**

*Simplify the geometry while preserving topology.*

- **OGRGeometryH OGR\_G\_DelaunayTriangulation** ( **OGRGeometryH** hThis, double dfTolerance, int bOnlyEdges) **CPL\_WARN\_UNUSED\_RESULT**

*Return a Delaunay triangulation of the vertices of the geometry.*

- **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**) **CPL\_WARN\_UNUSED\_RESULT**

*Compute boundary.*



- **OGRGeometryH OGR\_G\_ConvexHull ( OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT**  
*Compute convex hull.*
- **OGRGeometryH OGR\_G\_Buffer ( OGRGeometryH, double, int) CPL\_WARN\_UNUSED\_RESULT**  
*Compute buffer of geometry.*
- **OGRGeometryH OGR\_G\_Intersection ( OGRGeometryH, OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT**  
*Compute intersection.*
- **OGRGeometryH OGR\_G\_Union ( OGRGeometryH, OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT**  
*Compute union.*
- **OGRGeometryH OGR\_G\_UnionCascaded ( OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT**  
*Compute union using cascading.*
- **OGRGeometryH OGR\_G\_PointOnSurface ( OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT**  
*Returns a point guaranteed to lie on the surface.*
- **OGRGeometryH OGR\_G\_Difference ( OGRGeometryH, OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT**  
*Compute difference.*
- **OGRGeometryH OGR\_G\_SymDifference ( OGRGeometryH, OGRGeometryH) CPL\_WARN\_UNUSED\_RESULT**  
*Compute symmetric difference.*
- **double OGR\_G\_Distance ( OGRGeometryH, OGRGeometryH)**  
*Compute distance between two geometries.*
- **double OGR\_G\_Distance3D ( OGRGeometryH, OGRGeometryH)**  
*Returns the 3D 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) CPL\_WARN\_UNUSED\_RESULT**  
*Fetch point at given distance along curve.*
- **void OGR\_G\_Empty ( OGRGeometryH)**  
*Clear geometry information. This restores the geometry to its 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) CPL\_WARN\_UNUSED\_RESULT**  
*Polygonizes a set of sparse edges.*
- **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.*
- **int OGR\_G\_GetPointsZM ( OGRGeometryH hGeom, void \*pabyX, int nXStride, void \*pabyY, int nYStride, void \*pabyZ, int nZStride, void \*pabyM, int nMStride)**

- 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.*
- double **OGR\_G\_GetM** ( **OGRGeometryH**, int)
  - Fetch the m 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\_GetPointZM** ( **OGRGeometryH**, int iPoint, double \*, 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\_SetPointM** ( **OGRGeometryH**, int iPoint, double, double, double)
  - Set the location of a vertex in a point or linestring geometry.*
- void **OGR\_G\_SetPointZM** ( **OGRGeometryH**, int iPoint, double, double, 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\_AddPointM** ( **OGRGeometryH**, double, double, double)
  - Add a point to a geometry (line string or point).*
- void **OGR\_G\_AddPointZM** ( **OGRGeometryH**, double, double, double, double)
  - Add a point to a geometry (line string or point).*
- void **OGR\_G\_SetPoints** ( **OGRGeometryH** hGeom, int nPointsIn, const void \*pabyX, int nXStride, const void \*pabyY, int nYStride, const void \*pabyZ, int nZStride)
  - Assign all points in a point or a line string geometry.*
- void **OGR\_G\_SetPointsZM** ( **OGRGeometryH** hGeom, int nPointsIn, const void \*pabyX, int nXStride, const void \*pabyY, int nYStride, const void \*pabyZ, int nZStride, const void \*pabyM, int nMStride)
  - Assign all points in a point or a line string geometry.*
- void **OGR\_G\_SwapXY** ( **OGRGeometryH** hGeom)
  - Swap x and y coordinates.*
- 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) **CPL\_WARN\_UNUSED\_RESULT**  
*Return, possibly approximate, linear version of this geometry.*
- **OGRGeometryH OGR\_G\_GetCurveGeometry** ( **OGRGeometryH** hGeom, char \*\*papszOptions) **CPL↔\_WARN\_UNUSED\_RESULT**  
*Return curve version of this geometry.*
- 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\_GetFieldName** ( 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\_UNUSED\_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.*
- OGRErr **OGR\_FD\_ReorderFieldDefns** ( OGRFeatureDefnH hDefn, int \*panMap)  
*Reorder the field definitions in the array of the feature 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) CPL\_WARN\_UNUSED\_RESULT  
*Take away ownership of geometry.*
- OGRFeatureH **OGR\_F\_Clone** ( OGRFeatureH) CPL\_WARN\_UNUSED\_RESULT

- 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.*

  - int **OGR\_F\_IsFieldNull** ( OGRFeatureH, int)
- Test if a field is null.*

  - int **OGR\_F\_IsFieldSetAndNotNull** ( OGRFeatureH, int)
- Test if a field is set and not null.*

  - void **OGR\_F\_SetFieldNull** ( OGRFeatureH, int)
- Clear a field, marking it as null.*

  - **OGRField \* OGR\_F\_GetRawFieldRef** ( OGRFeatureH, int)
- Fetch an handle to the internal field value given the index.*

  - int **OGR\_RawField\_IsUnset** (const OGRField \*)
- Returns whether a raw field is unset.*

  - int **OGR\_RawField\_IsNull** (const OGRField \*)
- Returns whether a raw field is null.*

  - void **OGR\_RawField\_SetUnset** ( OGRField \*)
- Mark a raw field as unset.*

  - void **OGR\_RawField\_SetNull** ( OGRField \*)
- Mark a raw field as null.*

  - 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, const 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, const double \*)

*Set field to list of doubles value.*
- void **OGR\_F\_SetFieldStringList** ( **OGRFeatureH**, int, **CSLConstList**)

*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, const 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.*

  - void **OGR\_F\_SetStyleStringDirectly** ( OGRFeatureH, char \*)
- Set feature style string.*

  - **OGRStyleTableH OGR\_F\_GetStyleTable** ( OGRFeatureH)
- void **OGR\_F\_SetStyleTableDirectly** ( OGRFeatureH, OGRStyleTableH)
- void **OGR\_F\_SetStyleTable** ( OGRFeatureH, OGRStyleTableH)
- const char \* **OGR\_F\_GetNativeData** ( OGRFeatureH)

*Returns the native data for the feature.*
- void **OGR\_F\_SetNativeData** ( OGRFeatureH, const char \*)

*Sets the native data for the feature.*
- const char \* **OGR\_F\_GetNativeMediaType** ( OGRFeatureH)

*Returns the native media type for the feature.*
- void **OGR\_F\_SetNativeMediaType** ( OGRFeatureH, const char \*)

*Sets the native media type for the feature.*
- 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) CPL\_WARN\_UNUSED\_RESULT

*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) CPL\_WARN\_UNUSED\_RESULT

*Fetch a feature by its identifier.*
- **OGRerr OGR\_L\_SetFeature** ( OGRLayerH, OGRFeatureH) CPL\_WARN\_UNUSED\_RESULT

*Rewrite an existing feature.*
- **OGRerr OGR\_L\_CreateFeature** ( OGRLayerH, OGRFeatureH) CPL\_WARN\_UNUSED\_RESULT

*Create and write a new feature within a layer.*
- **OGRerr OGR\_L\_DeleteFeature** ( OGRLayerH, GIntBig) CPL\_WARN\_UNUSED\_RESULT

*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)**  
*Delete an existing 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) CPL\_WARN\_UNUSED\_RESULT**  
*For datasources which support transactions, StartTransaction creates a transaction.*
- **OGRERR OGR\_L\_CommitTransaction ( OGRLayerH) CPL\_WARN\_UNUSED\_RESULT**  
*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.*
- **OGRStyleTableH OGR\_L\_GetStyleTable ( OGRLayerH)**
- **void OGR\_L\_SetStyleTableDirectly ( OGRLayerH, OGRStyleTableH)**
- **void OGR\_L\_SetStyleTable ( OGRLayerH, OGRStyleTableH)**
- **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, OGR↵  
**RwkbGeometryType**, char \*\*)  
*This function attempts to create a new layer on the data source with the indicated name, coordinate system, geometry type.*
- **OGRLayerH OGR\_DS\_CopyLayer** ( OGRDataSourceH, OGRLayerH, const char \*, char \*\*)  
*Duplicate an existing layer.*
- **int OGR\_DS\_TestCapability** ( OGRDataSourceH, const char \*)  
*Test if capability is available.*
- **OGRLayerH OGR\_DS\_ExecuteSQL** ( OGRDataSourceH, const char \*, OGRGeometryH, const char \*)  
*Execute an SQL statement against the data store.*
- **void OGR\_DS\_ReleaseResultSet** ( OGRDataSourceH, OGRLayerH)  
*Release results of OGR\_DS\_ExecuteSQL() (p. ??).*
- **OGRERR OGR\_DS\_SyncToDisk** ( OGRDataSourceH)
- **OGRStyleTableH OGR\_DS\_GetStyleTable** ( OGRDataSourceH)
- **void OGR\_DS\_SetStyleTableDirectly** ( OGRDataSourceH, OGRStyleTableH)
- **void OGR\_DS\_SetStyleTable** ( OGRDataSourceH, OGRStyleTableH)
- **const char \* OGR\_Dr\_GetName** ( OGRSFDriverH)  
*Fetch name of driver (file format). This name should be relatively short (10-40 characters), and should reflect the underlying file format. For instance "ESRI Shapefile".*
- **OGRDataSourceH OGR\_Dr\_Open** ( OGRSFDriverH, const char \*, int) **CPL\_WARN\_UNUSED\_RESULT**  
*Attempt to open file with this driver.*
- **int OGR\_Dr\_TestCapability** ( OGRSFDriverH, const char \*)  
*Test if capability is available.*
- **OGRDataSourceH OGR\_Dr\_CreateDataSource** ( OGRSFDriverH, const char \*, char \*\*) **CPL\_WARN\_UNUSED\_RESULT**  
*This function attempts to create a new data source based on the passed driver.*
- **OGRDataSourceH OGR\_Dr\_CopyDataSource** ( OGRSFDriverH, OGRDataSourceH, const char \*, char \*\*) **CPL\_WARN\_UNUSED\_RESULT**

*This function creates a new datasource by copying all the layers from the source datasource.*

- **OGRErr OGR\_Dr\_DeleteDataSource** ( OGRSFDriverH, const char \*)  
*Delete a datasource.*
- **OGRDataSourceH OGROpen** (const char \*, int, OGRSFDriverH \*) **CPL\_WARN\_UNUSED\_RESULT**  
*Open a file / data source with one of the registered drivers.*
- **OGRDataSourceH OGROpenShared** (const char \*, int, OGRSFDriverH \*) **CPL\_WARN\_UNUSED\_RESULT**  
*Open a file / data source with one of the registered drivers if not already opened, or increment reference count of already opened data source previously opened with **OGROpenShared()** (p. ??)*
- **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)
- **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 \*bValuesNull)  
*Get Style Tool parameter value as string.*
- **int OGR\_ST\_GetParamNum** ( OGRStyleToolH hST, int eParam, int \*bValuesNull)  
*Get Style Tool parameter value as an integer.*
- **double OGR\_ST\_GetParamDbf** ( OGRStyleToolH hST, int eParam, int \*bValuesNull)

- 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\_SetParamDbf** ( **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)

### 12.15.1 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. ??)

### 12.15.2 Macro Definition Documentation

## 12.15.2.1 OGR\_FOR\_EACH\_FEATURE\_BEGIN

```
#define OGR_FOR_EACH_FEATURE_BEGIN(
    hFeat,
    hLayer )
```

**Value:**

```
{ \
    OGRFeatureH hFeat = CPL_NULLPTR; \
    OGR_L_ResetReading(hLayer); \
    while( true ) \
    { \
        if( hFeat ) \
            OGR_F_Destroy(hFeat); \
        hFeat = OGR_L_GetNextFeature(hLayer); \
        if( !hFeat ) \
            break;
```

Convenience macro to iterate over features of a layer.

Typical usage is:

```
OGR_FOR_EACH_FEATURE_BEGIN(hFeat, hLayer) (p.??)
{
    // Do something, including continue, break;
    // Do not explicitly destroy the feature (unless you use return or goto
    // outside of the loop, in which case use OGR_F_Destroy(hFeat))
}
OGR_FOR_EACH_FEATURE_END(hFeat) (p.??)
```

In C++, you might want to use instead range-based loop:

```
for( auto&& poFeature: poLayer )
{
}
```

**Parameters**

<i>hFeat</i>	variable name for OGRFeatureH. The variable will be declared inside the macro body.
<i>hLayer</i>	layer to iterate over.

**Since**

GDAL 2.3

## 12.15.2.2 OGR\_FOR\_EACH\_FEATURE\_END

```
#define OGR_FOR_EACH_FEATURE_END(
    hFeat )
```

**Value:**

```
} \
    OGR_F_Destroy(hFeat); \
}
```

End of iterator.

### 12.15.3 Typedef Documentation

#### 12.15.3.1 OGRCoordinateTransformationH

```
typedef void* OGRCoordinateTransformationH
```

Opaque type for a coordinate transformation object

#### 12.15.3.2 OGRDataSourceH

```
typedef void* OGRDataSourceH
```

Opaque type for a OGR datasource (**OGRDataSource** (p. ??))

#### 12.15.3.3 OGRFeatureDefnH

```
typedef void* OGRFeatureDefnH
```

Opaque type for a feature definition (**OGRFeatureDefn** (p. ??))

#### 12.15.3.4 OGRFeatureH

```
typedef void* OGRFeatureH
```

Opaque type for a feature (**OGRFeature** (p. ??))

#### 12.15.3.5 OGRFieldDefnH

```
typedef void* OGRFieldDefnH
```

Opaque type for a field definition (**OGRFieldDefn** (p. ??))

#### 12.15.3.6 OGRGeometryH

```
typedef void* OGRGeometryH
```

Opaque type for a geometry

### 12.15.3.7 OGRGeomFieldDefnH

```
typedef struct OGRGeomFieldDefnHS* OGRGeomFieldDefnH
```

Opaque type for a geometry field definition (**OGRGeomFieldDefn** (p. ??))

### 12.15.3.8 OGRLayerH

```
typedef void* OGRLayerH
```

Opaque type for a layer (**OGRLayer** (p. ??))

### 12.15.3.9 OGRSFDriverH

```
typedef void* OGRSFDriverH
```

Opaque type for a OGR driver (**OGRSFDriver** (p. ??))

### 12.15.3.10 OGRSpatialReferenceH

```
typedef void* OGRSpatialReferenceH
```

Opaque type for a spatial reference system

### 12.15.3.11 OGRStyleMgrH

```
typedef void* OGRStyleMgrH
```

Style manager opaque type

### 12.15.3.12 OGRStyleTableH

```
typedef void* OGRStyleTableH
```

Opaque type for a style table (**OGRStyleTable** (p. ??))

### 12.15.3.13 OGRStyleToolH

```
typedef void* OGRStyleToolH
```

Style tool opaque type

## 12.15.4 Function Documentation

### 12.15.4.1 OGR\_AreTypeSubTypeCompatible()

```
int OGR_AreTypeSubTypeCompatible (
    OGRFieldType eType,
    OGRFieldSubType eSubType )
```

Return if type and subtype are compatible.

## Parameters

<i>eType</i>	the field type.
<i>eSubType</i>	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().

## 12.15.4.2 OGR\_Dr\_CopyDataSource()

```
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

<i>hDriver</i>	handle to the driver on which data source creation is based.
<i>hSrcDS</i>	source datasource
<i>pszNewName</i>	the name for the new data source.
<i>papszOptions</i>	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: <a href="http://www.gdal.org/ogr_formats.html">http://www.gdal.org/ogr_formats.html</a>

## Returns

NULL is returned on failure, or a new **OGRDataSource** (p. ??) handle on success.



## 12.15.4.3 OGR\_Dr\_CreateDataSource()

```
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

<i>hDriver</i>	handle to the driver on which data source creation is based.
<i>pszName</i>	the name for the new data source. UTF-8 encoded.
<i>papszOptions</i>	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: <a href="http://www.gdal.org/ogr_formats.html">http://www.gdal.org/ogr_formats.html</a>

## Returns

NULL is returned on failure, or a new **OGRDataSource** (p. ??) handle on success.

## 12.15.4.4 OGR\_Dr\_DeleteDataSource()

```
OGRERR OGR_Dr_DeleteDataSource (
    OGRSFDriverH hDriver,
    const char * pszDataSource )
```

Delete a datasource.

Delete (from the disk, in the database, ...) the named datasource. Normally it would be safest if the datasource was not open at the time.

Whether this is a supported operation on this driver case be tested using TestCapability() on ODrCDeleteDataSource.

**Deprecated** Use GDALDeleteDataset() in GDAL 2

## Parameters

<i>hDriver</i>	handle to the driver on which data source deletion is based.
<i>pszDataSource</i>	the name of the datasource to delete.

**Returns**

OGRERR\_NONE on success, and OGRERR\_UNSUPPORTED\_OPERATION if this is not supported by this driver.

**12.15.4.5 OGR\_Dr\_GetName()**

```
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**

<i>hDriver</i>	handle to the driver to get the name from.
----------------	--

**Returns**

driver name. This is an internal string and should not be modified or freed.

**12.15.4.6 OGR\_Dr\_Open()**

```
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**

<i>hDriver</i>	handle to the driver that is used to open file.
<i>pszName</i>	the name of the file, or data source to try and open.
<i>bUpdate</i>	TRUE if update access is required, otherwise FALSE (the default).

**Returns**

NULL on error or if the pass name is not supported by this driver, otherwise an handle to a GDALDataset. This GDALDataset should be closed by deleting the object when it is no longer needed.

**12.15.4.7 OGR\_Dr\_TestCapability()**

```
int OGR_Dr_TestCapability (
    OGRSFDriverH hDriver,
    const char * pszCap )
```

Test if capability is available.

One of the following data source capability names can be passed into this function, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

- **ODrCCreateDataSource**: True if this driver can support creating data sources.
- **ODrCDeleteDataSource**: True if this driver supports deleting data sources.

The #define macro forms of the capability names should be used in preference to the strings themselves to avoid misspelling.

**Deprecated** Use GDALGetMetadataItem(hDriver, GDAL\_DCAP\_CREATE) in GDAL 2.0

**Parameters**

<i>hDriver</i>	handle to the driver to test the capability against.
<i>pszCap</i>	the capability to test.

**Returns**

TRUE if capability available otherwise FALSE.

**12.15.4.8 OGR\_DS\_CopyLayer()**

```
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

<i>hDS</i>	handle to the data source where to create the new layer
<i>hSrcLayer</i>	handle to the source layer.
<i>pszNewName</i>	the name of the layer to create.
<i>papszOptions</i>	a StringList of name=value options. Options are driver specific.

#### Returns

an handle to the layer, or NULL if an error occurs.

References `OGRLayer::FromHandle()`, `OGRLayer::ToHandle()`, and `VALIDATE_POINTER1`.

#### 12.15.4.9 OGR\_DS\_CreateLayer()

```
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

<i>hDS</i>	The dataset handle.
<i>pszName</i>	the name for the new layer. This should ideally not match any existing layer on the datasource.
<i>hSpatialRef</i>	handle to the coordinate system to use for the new layer, or NULL if no coordinate system is available. The driver might only increase the reference counter of the object to take ownership, and not make a full copy, so do not use <b>OSRDestroySpatialReference()</b> (p. ??), but <b>OSRRelease()</b> (p. ??) instead when you are done with the object.
<i>eType</i>	the geometry type for the layer. Use <code>wkbUnknown</code> if there are no constraints on the types geometry to be written.
<i>papszOptions</i>	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: <a href="http://www.gdal.org/ogr_formats.html">http://www.gdal.org/ogr_formats.html</a>

**Returns**

NULL is returned on failure, or a new **OGRLayer** (p. ??) handle on success.

**Example:**

```
#include "ogr_sfrmts.h"
#include "cpl_string.h"

...

OGRLayerH *hLayer;
char **papszOptions;

if( OGR_DS_TestCapability( hDS, ODS_CCreateLayer ) )
{
    ...
}

papszOptions = CSLSetNameValue( papszOptions, "DIM", "2" );
hLayer = OGR_DS_CreateLayer( hDS, "NewLayer", NULL, wkbUnknown,
                             papszOptions );
CSLDestroy( papszOptions );

if( hLayer == NULL )
{
    ...
}
```

References CPLError(), and VALIDATE\_POINTER1.

**12.15.4.10 OGR\_DS\_DeleteLayer()**

```
OGRERR OGR_DS_DeleteLayer (
    OGRDataSourceH hDS,
    int iLayer )
```

Delete the indicated layer from the datasource.

If this method is supported the ODS\_CDeleteLayer capability will test TRUE on the **OGRDataSource** (p. ??).

**Deprecated** Use GDALDatasetDeleteLayer() in GDAL 2.0

**Parameters**

<i>hDS</i>	handle to the datasource
<i>iLayer</i>	the index of the layer to delete.

**Returns**

OGRERR\_NONE on success, or OGRERR\_UNSUPPORTED\_OPERATION if deleting layers is not supported for this datasource.

References OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

### 12.15.4.11 OGR\_DS\_Destroy()

```
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

<i>hDataSource</i>	handle to allocated datasource object.
--------------------	--

### 12.15.4.12 OGR\_DS\_ExecuteSQL()

```
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 (i.e. Oracle and PostGIS) pass the SQL directly through to the underlying RDBMS.

Starting with OGR 1.10, the `SQLITE dialect` can also be used.

**Deprecated** Use `GDALDatasetExecuteSQL()` in GDAL 2.0

#### Parameters

<i>hDS</i>	handle to the data source on which the SQL query is executed.
<i>pszSQLCommand</i>	the SQL statement to execute.
<i>hSpatialFilter</i>	handle to a geometry which represents a spatial filter. Can be NULL.
<i>pszDialect</i>	allows control of the statement dialect. If set to NULL, the OGR SQL engine will be used, except for RDBMS drivers that will use their dedicated SQL engine, unless OGRSQL is explicitly passed as the dialect. Starting with OGR 1.10, the <code>SQLITE dialect</code> can also be used.

## Returns

an handle to a **OGRLayer** (p. ??) containing the results of the query. Deallocate with **OGR\_DS\_ReleaseResultSet()** (p. ??).

References OGRGeometry::FromHandle(), OGRLayer::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.13 OGR\_DS\_GetDriver()

```
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

<i>hDS</i>	handle to the datasource
------------	--------------------------

## Returns

NULL if driver info is not available, or pointer to a driver owned by the OGRSFDriverManager.

References VALIDATE\_POINTER1.

## 12.15.4.14 OGR\_DS\_GetLayer()

```
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

<i>hDS</i>	handle to the data source from which to get the layer.
<i>iLayer</i>	a layer number between 0 and <b>OGR_DS_GetLayerCount()</b> (p. ??)-1.

**Returns**

an handle to the layer, or NULL if iLayer is out of range or an error occurs.

References OGRLayer::ToHandle(), and VALIDATE\_POINTER1.

**12.15.4.15 OGR\_DS\_GetLayerByName()**

```
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**

<i>hDS</i>	handle to the data source from which to get the layer.
<i>pszLayerName</i>	Layer the layer name of the layer to fetch.

**Returns**

an handle to the layer, or NULL if the layer is not found or an error occurs.

References OGRLayer::ToHandle(), and VALIDATE\_POINTER1.

**12.15.4.16 OGR\_DS\_GetLayerCount()**

```
int OGR_DS_GetLayerCount (
    OGRDataSourceH hDS )
```

Get the number of layers in this data source.

**Deprecated** Use GDALDatasetGetLayerCount() in GDAL 2.0

**Parameters**

<i>hDS</i>	handle to the data source from which to get the number of layers.
------------	---



**Returns**

layer count.

References VALIDATE\_POINTER1.

**12.15.4.17 OGR\_DS\_GetName()**

```
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**

<i>hDS</i>	handle to the data source to get the name from.
------------	---

**Returns**

pointer to an internal name string which should not be modified or freed by the caller.

References VALIDATE\_POINTER1.

**12.15.4.18 OGR\_DS\_GetStyleTable()**

```
OGRStyleTableH OGR_DS_GetStyleTable (
    OGRDataSourceH )
```

Get style table

References VALIDATE\_POINTER1.

#### 12.15.4.19 OGR\_DS\_ReleaseResultSet()

```
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

<i>hDS</i>	an handle to the data source on which was executed an SQL query.
<i>hLayer</i>	handle to the result of a previous <b>OGR_DS_ExecuteSQL()</b> (p. ??) call.

References OGRLayer::FromHandle(), and VALIDATE\_POINTER0.

## 12.15.4.20 OGR\_DS\_SetStyleTable()

```
void OGR_DS_SetStyleTable (
    OGRDataSourceH ,
    OGRStyleTableH )
```

Set style table

References VALIDATE\_POINTER0.

## 12.15.4.21 OGR\_DS\_SetStyleTableDirectly()

```
void OGR_DS_SetStyleTableDirectly (
    OGRDataSourceH ,
    OGRStyleTableH )
```

Set style table (and take ownership)

References VALIDATE\_POINTER0.

## 12.15.4.22 OGR\_DS\_SyncToDisk()

```
OGRERR OGR_DS_SyncToDisk (
    OGRDataSourceH )
```

Flush pending changes to disk. See GDALDataset::FlushCache()

References CPLGetLastErrorType(), OGRERR\_FAILURE, OGRERR\_INVALID\_HANDLE, OGRERR\_NONE, and VALIDATE\_POINTER1.

### 12.15.4.23 OGR\_DS\_TestCapability()

```
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, OLCurveGeometries 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 misspelling.

**Deprecated** Use GDALDatasetTestCapability() in GDAL 2.0

#### Parameters

<i>hDS</i>	handle to the data source against which to test the capability.
<i>pszCapability</i>	the capability to test.

#### Returns

TRUE if capability available otherwise FALSE.

References VALIDATE\_POINTER1.

### 12.15.4.24 OGR\_F\_Clone()

```
OGRFeatureH OGR_F_Clone (
    OGRFeatureH hFeat )
```

Duplicate feature.

The newly created feature is owned by the caller, and will have its own reference to the **OGRFeatureDefn** (p. ??).

This function is the same as the C++ method **OGRFeature::Clone()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature to clone.
--------------	---------------------------------

## Returns

an handle to the new feature, exactly matching this feature.

References `OGRFeature::FromHandle()`, `OGRFeature::ToHandle()`, and `VALIDATE_POINTER1`.

12.15.4.25 `OGR_F_Create()`

```
OGRFeatureH OGR_F_Create (
    OGRFeatureDefnH hDefn )
```

Feature factory.

Note that the **OGRFeature** (p. ??) will increment the reference count of its defining **OGRFeatureDefn** (p. ??). Destruction of the **OGRFeatureDefn** (p. ??) before destruction of all **OGRFeatures** that depend on it is likely to result in a crash.

This function is the same as the C++ method **OGRFeature::OGRFeature()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature class (layer) definition to which the feature will adhere.
--------------	--

## Returns

an handle to the new feature object with null fields and no geometry, or, starting with GDAL 2.1, NULL in case out of memory situation.

References `OGRFeature::CreateFeature()`, `OGRFeatureDefn::FromHandle()`, `OGRFeature::ToHandle()`, and `VALIDATE_POINTER1`.

12.15.4.26 `OGR_F_Destroy()`

```
void OGR_F_Destroy (
    OGRFeatureH hFeat )
```

Destroy feature.

The feature is deleted, but within the context of the GDAL/OGR heap. This is necessary when higher level applications use GDAL/OGR from a DLL and they want to delete a feature created within the DLL. If the delete is done in the calling application the memory will be freed onto the application heap which is inappropriate.

This function is the same as the C++ method **OGRFeature::DestroyFeature()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature to destroy.
--------------	-----------------------------------

References `OGRFeature::FromHandle()`.

12.15.4.27 `OGR_F_DumpReadable()`

```
void OGR_F_DumpReadable (
    OGRFeatureH hFeat,
    FILE * fpOut )
```

Dump this feature in a human readable form.

This dumps the attributes, and geometry; however, it doesn't definition information (other than field types and names), nor does it report the geometry spatial reference system.

This function is the same as the C++ method `OGRFeature::DumpReadable()` (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature to dump.
<i>fpOut</i>	the stream to write to, such as <code>strout</code> .

References `OGRFeature::DumpReadable()`, `OGRFeature::FromHandle()`, and `VALIDATE_POINTER0`.

12.15.4.28 `OGR_F_Equal()`

```
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

<i>hFeat</i>	handle to one of the feature.
<i>hOtherFeat</i>	handle to the other feature to test this one against.

**Returns**

TRUE if they are equal, otherwise FALSE.

References OGRFeature::FromHandle(), and VALIDATE\_POINTER1.

**12.15.4.29 OGR\_F\_FillUnsetWithDefault()**

```
void OGR_F_FillUnsetWithDefault (
    OGRFeatureH hFeat,
    int bNotNullableOnly,
    char ** ppszOptions )
```

Fill unset fields with default values that might be defined.

This function is the same as the C++ method **OGRFeature::FillUnsetWithDefault()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature.
<i>bNotNullableOnly</i>	if we should fill only unset fields with a not-null constraint.
<i>ppszOptions</i>	unused currently. Must be set to NULL.

**Since**

GDAL 2.0

References OGRFeature::FromHandle(), and VALIDATE\_POINTER0.

**12.15.4.30 OGR\_F\_GetDefnRef()**

```
OGRFeatureDefnH OGR_F_GetDefnRef (
    OGRFeatureH hFeat )
```

Fetch feature definition.

This function is the same as the C++ method **OGRFeature::GetDefnRef()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to get the feature definition from.
--------------	---

**Returns**

an handle to the feature definition object on which feature depends.

References `OGRFeature::FromHandle()`, `OGRFeatureDefn::ToHandle()`, and `VALIDATE_POINTER1`.

#### 12.15.4.31 `OGR_F_GetFID()`

```
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

<i>hFeat</i>	handle to the feature from which to get the feature identifier.
--------------	---

##### Returns

feature id or `OGRNullFID` if none has been assigned.

References `OGRFeature::FromHandle()`, `OGRFeature::GetFID()`, and `VALIDATE_POINTER1`.

#### 12.15.4.32 `OGR_F_GetFieldAsBinary()`

```
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

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to <code>GetFieldCount()-1</code> .
<i>pnBytes</i>	location to place count of bytes returned.

##### Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

References `OGRFeature::FromHandle()`, and `VALIDATE_POINTER1`.



## 12.15.4.33 OGR\_F\_GetFieldAsDateTime()

```
int OGR_F_GetFieldAsDateTime (
    OGRFeatureH hFeat,
    int iField,
    int * pnYear,
    int * pnMonth,
    int * pnDay,
    int * pnHour,
    int * pnMinute,
    int * pnSecond,
    int * pnTZFlag )
```

Fetch field value as date and time.

Currently this method only works for OFTDate, OFTTime and OFTDateTime fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsDateTime()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnYear</i>	(including century)
<i>pnMonth</i>	(1-12)
<i>pnDay</i>	(1-31)
<i>pnHour</i>	(0-23)
<i>pnMinute</i>	(0-59)
<i>pnSecond</i>	(0-59)
<i>pnTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

## Returns

TRUE on success or FALSE on failure.

## See also

Use **OGR\_F\_GetFieldAsDateTimeEx()** (p. ??) for second with millisecond accuracy.

References VALIDATE\_POINTER1.

## 12.15.4.34 OGR\_F\_GetFieldAsDateTimeEx()

```
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()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnYear</i>	(including century)
<i>pnMonth</i>	(1-12)
<i>pnDay</i>	(1-31)
<i>pnHour</i>	(0-23)
<i>pnMinute</i>	(0-59)
<i>pfSecond</i>	(0-59 with millisecond accuracy)
<i>pnTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

#### Returns

TRUE on success or FALSE on failure.

#### Since

GDAL 2.0

References OGRFeature::FromHandle(), and VALIDATE\_POINTER1.

### 12.15.4.35 OGR\_F\_GetFieldAsDouble()

```

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

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

**Returns**

the field value.

References OGRFeature::FromHandle(), OGRFeature::GetFieldAsDouble(), and VALIDATE\_POINTER1.

**12.15.4.36 OGR\_F\_GetFieldAsDoubleList()**

```
const double* OGR_F_GetFieldAsDoubleList (
    OGRFeatureH hFeat,
    int iField,
    int * pnCount )
```

Fetch field value as a list of doubles.

Currently this function only works for OFTRealList fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsDoubleList()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnCount</i>	an integer to put the list count (number of doubles) into.

**Returns**

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeature::FromHandle(), and VALIDATE\_POINTER1.

**12.15.4.37 OGR\_F\_GetFieldAsInteger()**

```
int OGR_F_GetFieldAsInteger (
    OGRFeatureH hFeat,
    int iField )
```

Fetch field value as integer.

OFTString features will be translated using atoi(). OFTReal fields will be cast to integer. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsInteger()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

**Returns**

the field value.

References `OGRFeature::FromHandle()`, `OGRFeature::GetFieldAsInteger()`, and `VALIDATE_POINTER1`.

**12.15.4.38 OGR\_F\_GetFieldAsInteger64()**

```
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**

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to <code>GetFieldCount()</code> -1.

**Returns**

the field value.

**Since**

GDAL 2.0

References `OGRFeature::FromHandle()`, `OGRFeature::GetFieldAsInteger64()`, and `VALIDATE_POINTER1`.

**12.15.4.39 OGR\_F\_GetFieldAsInteger64List()**

```
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

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnCount</i>	an integer to put the list count (number of integers) into.

## Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

## Since

GDAL 2.0

References OGRFeature::FromHandle(), and VALIDATE\_POINTER1.

## 12.15.4.40 OGR\_F\_GetFieldAsIntegerList()

```
const int* OGR_F_GetFieldAsIntegerList (
    OGRFeatureH hFeat,
    int iField,
    int * pnCount )
```

Fetch field value as a list of integers.

Currently this function only works for OFTIntegerList fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsIntegerList()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnCount</i>	an integer to put the list count (number of integers) into.

## Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeature::FromHandle(), and VALIDATE\_POINTER1.

#### 12.15.4.41 OGR\_F\_GetFieldAsString()

```
const char* OGR_F_GetFieldAsString (
    OGRFeatureH hFeat,
    int iField )
```

Fetch field value as a string.

OFTReal and OFTInteger fields will be translated to string using `sprintf()`, but not necessarily using the established formatting rules. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsString()** (p. ??).

##### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to <code>GetFieldCount()-1</code> .

##### Returns

the field value. This string is internal, and should not be modified, or freed. Its lifetime may be very brief.

References `OGRFeature::FromHandle()`, `OGRFeature::GetFieldAsString()`, and `VALIDATE_POINTER1`.

#### 12.15.4.42 OGR\_F\_GetFieldAsStringList()

```
char** OGR_F_GetFieldAsStringList (
    OGRFeatureH hFeat,
    int iField )
```

Fetch field value as a list of strings.

Currently this method only works for OFTStringList fields.

The returned list is terminated by a NULL pointer. The number of elements can also be calculated using **CSLCount()** (p. ??).

This function is the same as the C++ method **OGRFeature::GetFieldAsStringList()** (p. ??).

##### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to <code>GetFieldCount()-1</code> .

##### Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

References `OGRFeature::FromHandle()`, `OGRFeature::GetFieldAsStringList()`, and `VALIDATE_POINTER1`.

## 12.15.4.43 OGR\_F\_GetFieldCount()

```
int OGR_F_GetFieldCount (
    OGRFeatureH hFeat )
```

Fetch number of fields on this feature This will always be the same as the field count for the **OGRFeatureDefn** (p. ??).

This function is the same as the C++ method **OGRFeature::GetFieldCount()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature to get the fields count from.
--------------	---

## Returns

count of fields.

References **OGRFeature::FromHandle()**, **OGRFeature::GetFieldCount()**, and **VALIDATE\_POINTER1**.

## 12.15.4.44 OGR\_F\_GetFieldDefnRef()

```
OGRFieldDefnH OGR_F_GetFieldDefnRef (
    OGRFeatureH hFeat,
    int i )
```

Fetch definition for this field.

This function is the same as the C++ method **OGRFeature::GetFieldDefnRef()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature on which the field is found.
<i>i</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> -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 **OGRFeature::FromHandle()**, **OGRFeature::GetFieldDefnRef()**, **OGRFieldDefn::ToHandle()**, and **VALIDATE\_POINTER1**.

## 12.15.4.45 OGR\_F\_GetFieldIndex()

```
int OGR_F_GetFieldIndex (
    OGRFeatureH hFeat,
    const char * pszName )
```

Fetch the field index given field name.

This is a cover for the **OGRFeatureDefn::GetFieldIndex()** (p. ??) method.

This function is the same as the C++ method **OGRFeature::GetFieldIndex()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature on which the field is found.
<i>pszName</i>	the name of the field to search for.

#### Returns

the field index, or -1 if no matching field is found.

References **OGRFeature::FromHandle()**, **OGRFeature::GetFieldIndex()**, and **VALIDATE\_POINTER1**.

#### 12.15.4.46 OGR\_F\_GetGeometryRef()

```
OGRGeometryH OGR_F_GetGeometryRef (
    OGRFeatureH hFeat )
```

Fetch an handle to feature geometry.

This function is essentially the same as the C++ method **OGRFeature::GetGeometryRef()** (p. ??) (the only difference is that this C function honours **OGRGetNonLinearGeometriesEnabledFlag()** (p. ??))

#### Parameters

<i>hFeat</i>	handle to the feature to get geometry from.
--------------	---

#### Returns

an handle to internal feature geometry. This object should not be modified.

References **OGRGeometryFactory::forceTo()**, **OGRFeature::FromHandle()**, **OGRFeature::GetGeometryRef()**, **OGRGeometry::getGeometryType()**, **OGR\_GT\_GetLinear()**, **OGR\_GT\_IsNonLinear()**, **OGRGetNonLinearGeometriesEnabledFlag()**, **OGRFeature::SetGeomFieldDirectly()**, **OGRFeature::StealGeometry()**, **OGRGeometry::ToHandle()**, and **VALIDATE\_POINTER1**.

#### 12.15.4.47 OGR\_F\_GetGeomFieldCount()

```
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

<i>hFeat</i>	handle to the feature to get the geometry fields count from.
--------------	--

## Returns

count of geometry fields.

## Since

GDAL 1.11

References `OGRFeature::FromHandle()`, `OGRFeature::GetGeomFieldCount()`, and `VALIDATE_POINTER1`.

12.15.4.48 `OGR_F_GetGeomFieldDefnRef()`

```
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

<i>hFeat</i>	handle to the feature on which the field is found.
<i>i</i>	the field to fetch, from 0 to <code>GetGeomFieldCount()-1</code> .

## 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

References `OGRFeature::FromHandle()`, `OGRGeomFieldDefn::ToHandle()`, and `VALIDATE_POINTER1`.

12.15.4.49 `OGR_F_GetGeomFieldIndex()`

```
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**

<i>hFeat</i>	handle to the feature on which the geometry field is found.
<i>pszName</i>	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 `OGRFeature::FromHandle()`, `OGRFeature::GetGeomFieldIndex()`, and `VALIDATE_POINTER1`.

**12.15.4.50 OGR\_F\_GetGeomFieldRef()**

```
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**

<i>hFeat</i>	handle to the feature to get geometry from.
<i>iField</i>	geometry field to get.

**Returns**

an handle to internal feature geometry. This object should not be modified.

**Since**

GDAL 1.11

References `OGRGeometryFactory::forceTo()`, `OGRFeature::FromHandle()`, `OGRGeometry::getGeometryType()`, `OGRFeature::GetGeomFieldRef()`, `OGR_GT_GetLinear()`, `OGR_GT_IsNonLinear()`, `OGRGetNonLinearGeometriesEnabledFlag()`, `OGRFeature::SetGeomFieldDirectly()`, `OGRFeature::StealGeometry()`, `OGRGeometry::ToHandle()`, and `VALIDATE_POINTER1`.

## 12.15.4.51 OGR\_F\_GetNativeData()

```
const char* OGR_F_GetNativeData (
    OGRFeatureH hFeat )
```

Returns the native data for the feature.

The native data is the representation in a "natural" form that comes from the driver that created this feature, or that is aimed at an output driver. The native data may be in different format, which is indicated by **OGR\_F\_GetNativeMedia**↵**MediaType()** (p. ??).

Note that most drivers do not support storing the native data in the feature object, and if they do, generally the NATIVE\_DATA open option must be passed at dataset opening.

The "native data" does not imply it is something more performant or powerful than what can be obtained with the rest of the API, but it may be useful in round-tripping scenarios where some characteristics of the underlying format are not captured otherwise by the OGR abstraction.

This function is the same as the C++ method **OGRFeature::GetNativeData()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature.
--------------	------------------------

## Returns

a string with the native data, or NULL if there is none.

## Since

GDAL 2.1

## See also

[https://trac.osgeo.org/gdal/wiki/rfc60\\_improved\\_roundtripping\\_in\\_ogr](https://trac.osgeo.org/gdal/wiki/rfc60_improved_roundtripping_in_ogr)

References **OGRFeature::FromHandle()**, **OGRFeature::GetNativeData()**, and **VALIDATE\_POINTER1**.

## 12.15.4.52 OGR\_F\_GetNativeMediaType()

```
const char* OGR_F_GetNativeMediaType (
    OGRFeatureH hFeat )
```

Returns the native media type for the feature.

The native media type is the identifier for the format of the native data. It follows the IANA RFC 2045 (see [https://en.wikipedia.org/wiki/Media\\_type](https://en.wikipedia.org/wiki/Media_type)), e.g. "application/vnd.geo+json" for JSon.

This function is the same as the C function **OGR\_F\_GetNativeMediaType()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature.
--------------	------------------------

## Returns

a string with the native media type, or NULL if there is none.

## Since

GDAL 2.1

## See also

[https://trac.osgeo.org/gdal/wiki/rfc60\\_improved\\_roundtripping\\_in\\_ogr](https://trac.osgeo.org/gdal/wiki/rfc60_improved_roundtripping_in_ogr)

References `OGRFeature::FromHandle()`, `OGRFeature::GetNativeMediaType()`, and `VALIDATE_POINTER1`.

12.15.4.53 `OGR_F_GetRawFieldRef()`

```
OGRField* OGR_F_GetRawFieldRef (
    OGRFeatureH hFeat,
    int iField )
```

Fetch an handle to the internal field value given the index.

This function is the same as the C++ method `OGRFeature::GetRawFieldRef()` (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature on which field is found.
<i>iField</i>	the field to fetch, from 0 to <code>GetFieldCount()-1</code> .

## Returns

the returned handle is to an internal data structure, and should not be freed, or modified.

References `OGRFeature::FromHandle()`, `OGRFeature::GetRawFieldRef()`, and `VALIDATE_POINTER1`.

12.15.4.54 `OGR_F_GetStyleString()`

```
const char* OGR_F_GetStyleString (
    OGRFeatureH hFeat )
```

Fetch style string for this feature.

Set the OGR Feature Style Specification for details on the format of this string, and **ogr\_featurestyle.h** (p. ??) for services available to parse it.

This function is the same as the C++ method **OGRFeature::GetStyleString()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to get the style from.
--------------	--

**Returns**

a reference to a representation in string format, or NULL if there isn't one.

References OGRFeature::FromHandle(), OGRFeature::GetStyleString(), and VALIDATE\_POINTER1.

**12.15.4.55 OGR\_F\_GetStyleTable()**

```
OGRStyleTableH OGR_F_GetStyleTable (
    OGRFeatureH )
```

Return style table

References OGRFeature::FromHandle(), OGRFeature::GetStyleTable(), and VALIDATE\_POINTER1.

**12.15.4.56 OGR\_F\_IsFieldNull()**

```
int OGR_F_IsFieldNull (
    OGRFeatureH hFeat,
    int iField )
```

Test if a field is null.

This function is the same as the C++ method **OGRFeature::IsFieldNull()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to test.

**Returns**

TRUE if the field is null, otherwise false.

**Since**

GDAL 2.2

References CPLError(), OGRFeature::FromHandle(), OGRFeature::GetFieldCount(), and VALIDATE\_POINTER1.

#### 12.15.4.57 OGR\_F\_IsFieldSet()

```
int OGR_F_IsFieldSet (
    OGRFeatureH hFeat,
    int iField )
```

Test if a field has ever been assigned a value or not.

This function is the same as the C++ method **OGRFeature::IsFieldSet()** (p. ??).

##### Parameters

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to test.

##### Returns

TRUE if the field has been set, otherwise false.

References CPLError(), OGRFeature::FromHandle(), OGRFeature::GetFieldCount(), and VALIDATE\_POINTER1.

#### 12.15.4.58 OGR\_F\_IsFieldSetAndNotNull()

```
int OGR_F_IsFieldSetAndNotNull (
    OGRFeatureH hFeat,
    int iField )
```

Test if a field is set and not null.

This function is the same as the C++ method **OGRFeature::IsFieldSetAndNotNull()** (p. ??).

##### Parameters

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to test.

##### Returns

TRUE if the field is set and not null, otherwise false.

##### Since

GDAL 2.2

References CPLError(), OGRFeature::FromHandle(), OGRFeature::GetFieldCount(), and VALIDATE\_POINTER1.

## 12.15.4.59 OGR\_F\_SetFID()

```
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

<i>hFeat</i>	handle to the feature to set the feature id to.
<i>nFID</i>	the new feature identifier value to assign.

## Returns

On success OGRERR\_NONE, or on failure some other value.

< Failure

References OGRFeature::FromHandle(), OGRERR\_FAILURE, OGRFeature::SetFID(), and VALIDATE\_POINTER1.

## 12.15.4.60 OGR\_F\_SetFieldBinary()

```
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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.



## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nBytes</i>	the number of bytes in pabyData array.
<i>pabyData</i>	the data to apply.

References OGRFeature::FromHandle(), OGRFeature::SetField(), and VALIDATE\_POINTER0.

## 12.15.4.61 OGR\_F\_SetFieldDateTime()

```
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.

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nYear</i>	(including century)
<i>nMonth</i>	(1-12)
<i>nDay</i>	(1-31)
<i>nHour</i>	(0-23)
<i>nMinute</i>	(0-59)
<i>nSecond</i>	(0-59)
<i>nTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

## See also

Use **OGR\_F\_SetFieldDateTimeEx()** (p. ??) for second with millisecond accuracy.

References OGRFeature::FromHandle(), and VALIDATE\_POINTER0.

#### 12.15.4.62 OGR\_F\_SetFieldDateTimeEx()

```
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.

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nYear</i>	(including century)
<i>nMonth</i>	(1-12)
<i>nDay</i>	(1-31)
<i>nHour</i>	(0-23)
<i>nMinute</i>	(0-59)
<i>fSecond</i>	(0-59, with millisecond accuracy)
<i>nTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

#### Since

GDAL 2.0

References OGRFeature::FromHandle(), and VALIDATE\_POINTER0.

#### 12.15.4.63 OGR\_F\_SetFieldDouble()

```
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. ??).

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>dfValue</i>	the value to assign.

References OGRFeature::FromHandle(), OGRFeature::SetField(), and VALIDATE\_POINTER0.

#### 12.15.4.64 OGR\_F\_SetFieldDoubleList()

```
void OGR_F_SetFieldDoubleList (
    OGRFeatureH hFeat,
    int iField,
    int nCount,
    const 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. ??).

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>padfValues</i>	the values to assign.

References `OGRFeature::FromHandle()`, `OGRFeature::SetField()`, and `VALIDATE_POINTER0`.

#### 12.15.4.65 `OGR_F_SetFieldInteger()`

```
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. ??).

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, `OGR_L_SetFeature()` (p. ??) must be used afterwards. Or if this is a new feature, `OGR_L_CreateFeature()` (p. ??) must be used afterwards.

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to <code>GetFieldCount()-1</code> .
<i>nValue</i>	the value to assign.

References `OGRFeature::FromHandle()`, `OGRFeature::SetField()`, and `VALIDATE_POINTER0`.

#### 12.15.4.66 `OGR_F_SetFieldInteger64()`

```
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. ??).

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, `OGR_L_SetFeature()` (p. ??) must be used afterwards. Or if this is a new feature, `OGR_L_CreateFeature()` (p. ??) must be used afterwards.

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>nValue</i>	the value to assign.

## Since

GDAL 2.0

References OGRFeature::FromHandle(), OGRFeature::SetField(), and VALIDATE\_POINTER0.

## 12.15.4.67 OGR\_F\_SetFieldInteger64List()

```
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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

## Since

GDAL 2.0

References OGRFeature::FromHandle(), OGRFeature::SetField(), and VALIDATE\_POINTER0.

**12.15.4.68 OGR\_F\_SetFieldIntegerList()**

```
void OGR_F_SetFieldIntegerList (
    OGRFeatureH hFeat,
    int iField,
    int nCount,
    const 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. ??).

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

References OGRFeature::FromHandle(), and VALIDATE\_POINTER0.

**12.15.4.69 OGR\_F\_SetFieldNull()**

```
void OGR_F_SetFieldNull (
    OGRFeatureH hFeat,
    int iField )
```

Clear a field, marking it as null.

This function is the same as the C++ method **OGRFeature::SetFieldNull()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to set to null.

**Since**

GDAL 2.2

References OGRFeature::FromHandle(), OGRFeature::SetFieldNull(), and VALIDATE\_POINTER0.

## 12.15.4.70 OGR\_F\_SetFieldRaw()

```
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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>psValue</i>	handle on the value to assign.

References **OGRFeature::FromHandle()**, **OGRFeature::SetField()**, and **VALIDATE\_POINTER0**.

## 12.15.4.71 OGR\_F\_SetFieldString()

```
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 **CPLAtolIntBig()** (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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pszValue</i>	the value to assign.

References OGRFeature::FromHandle(), OGRFeature::SetField(), and VALIDATE\_POINTER0.

## 12.15.4.72 OGR\_F\_SetFieldStringList()

```
void OGR_F_SetFieldStringList (
    OGRFeatureH hFeat,
    int iField,
    CSLConstList 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. ??).

## Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>papszValues</i>	the values to assign.

References OGRFeature::FromHandle(), OGRFeature::SetField(), and VALIDATE\_POINTER0.

## 12.15.4.73 OGR\_F\_SetFrom()

```
OGRERR OGR_F_SetFrom (
    OGRFeatureH hFeat,
    OGRFeatureH hOtherFeat,
    int bForgiving )
```

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The hOtherFeature does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied by corresponding field names. Field types do not have to exactly match. OGR\_F\_SetField\*() function conversion rules will be applied as needed.

This function is the same as the C++ method **OGRFeature::SetFrom()** (p. ??).



## Parameters

<i>hFeat</i>	handle to the feature to set to.
<i>hOtherFeat</i>	handle to the feature from which geometry, and field values will be copied.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

## Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

< Failure

< Failure

References OGRFeature::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.15.4.74 OGR\_F\_SetFromWithMap()

```
OGRERR OGR_F_SetFromWithMap (
    OGRFeatureH hFeat,
    OGRFeatureH hOtherFeat,
    int bForgiving,
    const int * panMap )
```

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The hOtherFeature does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied according to the provided indices map. Field types do not have to exactly match. OGR\_F\_SetField\*() function conversion rules will be applied as needed. This is more efficient than **OGR\_F\_SetFrom()** (p. ??) in that this doesn't lookup the fields by their names. Particularly useful when the field names don't match.

This function is the same as the C++ method **OGRFeature::SetFrom()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature to set to.
<i>hOtherFeat</i>	handle to the feature from which geometry, and field values will be copied.
<i>panMap</i>	Array of the indices of the destination feature's fields stored at the corresponding index of the source feature's fields. A value of -1 should be used to ignore the source's field. The array should not be NULL and be as long as the number of fields in the source feature.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

## Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

< Failure

< Failure  
< Failure

References OGRFeature::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.15.4.75 OGR\_F\_SetGeometry()

```
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. ??).

#### Note

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

#### Parameters

<i>hFeat</i>	handle to the feature on which new geometry is applied to.
<i>hGeom</i>	handle to the new geometry to apply to feature.

#### Returns

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

< Failure

References OGRGeometry::FromHandle(), OGRFeature::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.15.4.76 OGR\_F\_SetGeometryDirectly()

```
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. ??).

**Note**

This method has only an effect on the in-memory feature object. If this object comes from a layer and the modifications must be serialized back to the datasource, **OGR\_L\_SetFeature()** (p. ??) must be used afterwards. Or if this is a new feature, **OGR\_L\_CreateFeature()** (p. ??) must be used afterwards.

**Parameters**

<i>hFeat</i>	handle to the feature on which to apply the geometry.
<i>hGeom</i>	handle to the new geometry to apply to feature.

**Returns**

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

< Failure

References OGRGeometry::FromHandle(), OGRFeature::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_↔ POINTER1.

**12.15.4.77 OGR\_F\_SetGeomField()**

```
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**

<i>hFeat</i>	handle to the feature on which new geometry is applied to.
<i>iField</i>	geometry field to set.
<i>hGeom</i>	handle to the new geometry to apply to feature.

**Returns**

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

< Failure

References OGRGeometry::FromHandle(), OGRFeature::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_↔ POINTER1.

## 12.15.4.78 OGR\_F\_SetGeomFieldDirectly()

```
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

<i>hFeat</i>	handle to the feature on which to apply the geometry.
<i>iField</i>	geometry field to set.
<i>hGeom</i>	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

< Failure

References OGRGeometry::FromHandle(), OGRFeature::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.15.4.79 OGR\_F\_SetNativeData()

```
void OGR_F_SetNativeData (
    OGRFeatureH hFeat,
    const char * pszNativeData )
```

Sets the native data for the feature.

The native data is the representation in a "natural" form that comes from the driver that created this feature, or that is aimed at an output driver. The native data may be in different format, which is indicated by **OGR\_F\_GetNativeMediaTypes** (p. ??).

This function is the same as the C++ method **OGRFeature::SetNativeData** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature.
<i>pszNativeData</i>	a string with the native data, or NULL if there is none.

## Since

GDAL 2.1

## See also

[https://trac.osgeo.org/gdal/wiki/rfc60\\_improved\\_roundtripping\\_in\\_ogr](https://trac.osgeo.org/gdal/wiki/rfc60_improved_roundtripping_in_ogr)

References `OGRFeature::FromHandle()`, `OGRFeature::SetNativeData()`, and `VALIDATE_POINTER0`.

## 12.15.4.80 OGR\_F\_SetNativeMediaType()

```
void OGR_F_SetNativeMediaType (
    OGRFeatureH hFeat,
    const char * pszNativeMediaType )
```

Sets the native media type for the feature.

The native media type is the identifier for the format of the native data. It follows the IANA RFC 2045 (see [https://en.wikipedia.org/wiki/Media\\_type](https://en.wikipedia.org/wiki/Media_type)), e.g. "application/vnd.geo+json" for JSon.

This function is the same as the C++ method `OGRFeature::SetNativeMediaType()` (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature.
<i>pszNativeMediaType</i>	a string with the native media type, or NULL if there is none.

## Since

GDAL 2.1

## See also

[https://trac.osgeo.org/gdal/wiki/rfc60\\_improved\\_roundtripping\\_in\\_ogr](https://trac.osgeo.org/gdal/wiki/rfc60_improved_roundtripping_in_ogr)

References `OGRFeature::FromHandle()`, and `VALIDATE_POINTER0`.

**12.15.4.81 OGR\_F\_SetStyleString()**

```
void OGR_F_SetStyleString (
    OGRFeatureH hFeat,
    const char * pszStyle )
```

Set feature style string.

This method operate exactly as **OGR\_F\_SetStyleStringDirectly()** (p. ??) except that it does not assume ownership of the passed string, but instead makes a copy of it.

This function is the same as the C++ method **OGRFeature::SetStyleString()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to set style to.
<i>pszStyle</i>	the style string to apply to this feature, cannot be NULL.

References **OGRFeature::FromHandle()**, **OGRFeature::SetStyleString()**, and **VALIDATE\_POINTER0**.

**12.15.4.82 OGR\_F\_SetStyleStringDirectly()**

```
void OGR_F_SetStyleStringDirectly (
    OGRFeatureH hFeat,
    char * pszStyle )
```

Set feature style string.

This method operate exactly as **OGR\_F\_SetStyleString()** (p. ??) except that it assumes ownership of the passed string.

This function is the same as the C++ method **OGRFeature::SetStyleStringDirectly()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to set style to.
<i>pszStyle</i>	the style string to apply to this feature, cannot be NULL.

References **OGRFeature::FromHandle()**, **OGRFeature::SetStyleStringDirectly()**, and **VALIDATE\_POINTER0**.

**12.15.4.83 OGR\_F\_SetStyleTable()**

```
void OGR_F_SetStyleTable (
    OGRFeatureH ,
    OGRStyleTableH )
```

Set style table

References **OGRFeature::FromHandle()**, and **VALIDATE\_POINTER0**.

## 12.15.4.84 OGR\_F\_SetStyleTableDirectly()

```
void OGR_F_SetStyleTableDirectly (
    OGRFeatureH ,
    OGRStyleTableH )
```

Set style table and take ownership

References OGRFeature::FromHandle(), and VALIDATE\_POINTER0.

## 12.15.4.85 OGR\_F\_StealGeometry()

```
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 ownership 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.

References OGRFeature::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.86 OGR\_F\_UnsetField()

```
void OGR_F_UnsetField (
    OGRFeatureH hFeat,
    int iField )
```

Clear a field, marking it as unset.

This function is the same as the C++ method **OGRFeature::UnsetField()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to unset.

References OGRFeature::FromHandle(), OGRFeature::UnsetField(), and VALIDATE\_POINTER0.

### 12.15.4.87 OGR\_F\_Validate()

```
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

<i>hFeat</i>	handle to the feature to validate.
<i>nValidateFlags</i>	<code>OGR_F_VAL_ALL</code> or combination of <code>OGR_F_VAL_NULL</code> , <code>OGR_F_VAL_GEOM_TYPE</code> , <code>OGR_F_VAL_WIDTH</code> and <code>OGR_F_VAL_ALLOW_NULL_WHEN_DEFAULT</code> with ' ' operator
<i>bEmitError</i>	TRUE if a <code>CPLError()</code> (p. ??) must be emitted when a check fails

#### Returns

TRUE if all enabled validation tests pass.

#### Since

GDAL 2.0

References `OGRFeature::FromHandle()`, and `VALIDATE_POINTER1`.

### 12.15.4.88 OGR\_FD\_AddFieldDefn()

```
void OGR_FD_AddFieldDefn (
    OGRFeatureDefnH hDefn,
    OGRFieldDefnH hNewField )
```

Add a new field definition to the passed feature definition.

To add a new field definition to a layer definition, do not use this function directly, but use `OGR_L_CreateField()` (p. ??) instead.

This function should only be called while there are no `OGRFeature` (p. ??) objects in existence based on this `OGRFeatureDefn` (p. ??). The `OGRFieldDefn` (p. ??) passed in is copied, and remains the responsibility of the caller.

This function is the same as the C++ method `OGRFeatureDefn::AddFieldDefn()` (p. ??).



## Parameters

<i>hDefn</i>	handle to the feature definition to add the field definition to.
<i>hNewField</i>	handle to the new field definition.

References `OGRFieldDefn::FromHandle()`, and `OGRFeatureDefn::FromHandle()`.

## 12.15.4.89 OGR\_FD\_AddGeomFieldDefn()

```
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_CreateGeomField()` (p. ??) instead.

This function should only be called while there are no `OGRFeature` (p. ??) objects in existence based on this `OGRFeatureDefn` (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

<i>hDefn</i>	handle to the feature definition to add the geometry field definition to.
<i>hNewGeomField</i>	handle to the new field definition.

## Since

GDAL 1.11

References `OGRFeatureDefn::AddGeomFieldDefn()`, `OGRGeomFieldDefn::FromHandle()`, and `OGRFeatureDefn::FromHandle()`.

## 12.15.4.90 OGR\_FD\_Create()

```
OGRFeatureDefnH OGR_FD_Create (
    const char * pszName )
```

Create a new feature definition object to hold the field definitions.

The `OGRFeatureDefn` (p. ??) maintains a reference count, but this starts at zero, and should normally be incremented by the owner.

This function is the same as the C++ method `OGRFeatureDefn::OGRFeatureDefn()` (p. ??).

## Parameters

<i>pszName</i>	the name to be assigned to this layer/class. It does not need to be unique.
----------------	---

## Returns

handle to the newly created feature definition.

References `OGRFeatureDefn::ToHandle()`.

12.15.4.91 `OGR_FD_DeleteFieldDefn()`

```
OGRERR OGR_FD_DeleteFieldDefn (
    OGRFeatureDefnH hDefn,
    int iField )
```

Delete an existing field definition.

To delete an existing field definition from a layer definition, do not use this function directly, but use **OGR\_L\_DeleteField()** (p. ??) instead.

This method should only be called while there are no **OGRFeature** (p. ??) objects in existence based on this **OGRFeatureDefn** (p. ??).

This method is the same as the C++ method **OGRFeatureDefn::DeleteFieldDefn()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition.
<i>iField</i>	the index of the field definition.

## Returns

`OGRERR_NONE` in case of success.

## Since

OGR 1.9.0

References `OGRFeatureDefn::DeleteFieldDefn()`, and `OGRFeatureDefn::FromHandle()`.

12.15.4.92 `OGR_FD_DeleteGeomFieldDefn()`

```
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 `OG↔R_L_DeleteGeomField()` instead (*not implemented yet*).

This method should only be called while there are no **OGRFeature** (p. ??) objects in existence based on this **OG↔RFeatureDefn** (p. ??).

This method is the same as the C++ method **OGRFeatureDefn::DeleteGeomFieldDefn()** (p. ??).

#### Parameters

<i>hDefn</i>	handle to the feature definition.
<i>iGeomField</i>	the index of the geometry field definition.

#### Returns

`OGRERR_NONE` in case of success.

#### Since

GDAL 1.11

References **OGRFeatureDefn::FromHandle()**.

#### 12.15.4.93 OGR\_FD\_Dereference()

```
int OGR_FD_Dereference (
    OGRFeatureDefnH hDefn )
```

Decrements the reference count by one.

This function is the same as the C++ method **OGRFeatureDefn::Dereference()** (p. ??).

#### Parameters

<i>hDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
--------------	---

#### Returns

the updated reference count.

References **OGRFeatureDefn::Dereference()**, and **OGRFeatureDefn::FromHandle()**.

#### 12.15.4.94 OGR\_FD\_Destroy()

```
void OGR_FD_Destroy (
    OGRFeatureDefnH hDefn )
```

Destroy a feature definition object and release all memory associated with it.

This function is the same as the C++ method `OGRFeatureDefn::~~OGRFeatureDefn()`.

##### Parameters

<i>hDefn</i>	handle to the feature definition to be destroyed.
--------------	---

References `OGRFeatureDefn::FromHandle()`.

#### 12.15.4.95 OGR\_FD\_GetFieldCount()

```
int OGR_FD_GetFieldCount (
    OGRFeatureDefnH hDefn )
```

Fetch number of fields on the passed feature definition.

This function is the same as the C++ `OGRFeatureDefn::GetFieldCount()` (p. ??).

##### Parameters

<i>hDefn</i>	handle to the feature definition to get the fields count from.
--------------	--

##### Returns

count of fields.

References `OGRFeatureDefn::FromHandle()`, and `OGRFeatureDefn::GetFieldCount()`.

#### 12.15.4.96 OGR\_FD\_GetFieldDefn()

```
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. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition to get the field definition from.
<i>iField</i>	the field to fetch, between 0 and GetFieldCount()-1.

## Returns

an handle to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

References OGRFeatureDefn::FromHandle(), and OGRFieldDefn::ToHandle().

## 12.15.4.97 OGR\_FD\_GetFieldIndex()

```
int OGR_FD_GetFieldIndex (
    OGRFeatureDefnH hDefn,
    const char * pszFieldName )
```

Find field by name.

The field index of the first field matching the passed field name (case insensitively) is returned.

This function is the same as the C++ method **OGRFeatureDefn::GetFieldIndex** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition to get field index from.
<i>pszFieldName</i>	the field name to search for.

## Returns

the field index, or -1 if no match found.

References OGRFeatureDefn::FromHandle(), and OGRFeatureDefn::GetFieldIndex().

## 12.15.4.98 OGR\_FD\_GetGeomFieldCount()

```
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

<i>hDefn</i>	handle to the feature definition to get the fields count from.
--------------	--

## Returns

count of geometry fields.

## Since

GDAL 1.11

References `OGRFeatureDefn::FromHandle()`, and `OGRFeatureDefn::GetGeomFieldCount()`.

12.15.4.99 `OGR_FD_GetGeomFieldDefn()`

```
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

<i>hDefn</i>	handle to the feature definition to get the field definition from.
<i>iGeomField</i>	the geometry field to fetch, between 0 and <code>GetGeomFieldCount()</code> - 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

References `OGRFeatureDefn::FromHandle()`, and `OGRGeomFieldDefn::ToHandle()`.

12.15.4.100 `OGR_FD_GetGeomFieldIndex()`

```
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

<i>hDefn</i>	handle to the feature definition to get field index from.
<i>pszGeomFieldName</i>	the geometry field name to search for.

## Returns

the geometry field index, or -1 if no match found.

References `OGRFeatureDefn::FromHandle()`.

12.15.4.101 `OGR_FD_GetGeomType()`

```
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

<i>hDefn</i>	handle to the feature definition to get the geometry type from.
--------------	---

## Returns

the base type for all geometry related to this definition.

References `OGRFeatureDefn::FromHandle()`, `OGRFeatureDefn::GetGeomType()`, `OGR_GT_GetLinear()`, `OGR_GT_IsNonLinear()`, and `OGRGetNonLinearGeometriesEnabledFlag()`.

12.15.4.102 `OGR_FD_GetName()`

```
const char* OGR_FD_GetName (
    OGRFeatureDefnH hDefn )
```

Get name of the `OGRFeatureDefn` (p. ??) passed as an argument.

This function is the same as the C++ method `OGRFeatureDefn::GetName()` (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition to get the name from.
--------------	--



**Returns**

the name. This name is internal and should not be modified, or freed.

References OGRFeatureDefn::FromHandle(), and OGRFeatureDefn::GetName().

**12.15.4.103 OGR\_FD\_GetReferenceCount()**

```
int OGR_FD_GetReferenceCount (
    OGRFeatureDefnH hDefn )
```

Fetch current reference count.

This function is the same as the C++ method **OGRFeatureDefn::GetReferenceCount()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
--------------	---

**Returns**

the current reference count.

References OGRFeatureDefn::FromHandle(), and OGRFeatureDefn::GetReferenceCount().

**12.15.4.104 OGR\_FD\_IsGeometryIgnored()**

```
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**

<i>hDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
--------------	---

**Returns**

ignore state

References OGRFeatureDefn::FromHandle(), and OGRFeatureDefn::IsGeometryIgnored().

## 12.15.4.105 OGR\_FD\_IsSame()

```
int OGR_FD_IsSame (
    OGRFeatureDefnH hFDefn,
    OGRFeatureDefnH hOtherFDefn )
```

Test if the feature definition is identical to the other one.

## Parameters

<i>hFDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
<i>hOtherFDefn</i>	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

References OGRFeatureDefn::FromHandle(), and VALIDATE\_POINTER1.

## 12.15.4.106 OGR\_FD\_IsStyleIgnored()

```
int OGR_FD_IsStyleIgnored (
    OGRFeatureDefnH hDefn )
```

Determine whether the style can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::IsStyleIgnored()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition on which <b>OGRFeature</b> (p. ??) are based on.
--------------	---

## Returns

ignore state

References OGRFeatureDefn::FromHandle(), and OGRFeatureDefn::IsStyleIgnored().

## 12.15.4.107 OGR\_FD\_Reference()

```
int OGR_FD_Reference (
    OGRFeatureDefnH hDefn )
```

Increments the reference count by one.

The reference count is used keep track of the number of **OGRFeature** (p. ??) objects referencing this definition.

This function is the same as the C++ method **OGRFeatureDefn::Reference()** (p. ??).

#### Parameters

<i>hDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
--------------	---

#### Returns

the updated reference count.

References **OGRFeatureDefn::FromHandle()**, and **OGRFeatureDefn::Reference()**.

#### 12.15.4.108 OGR\_FD\_Release()

```
void OGR_FD_Release (
    OGRFeatureDefnH hDefn )
```

Drop a reference, and destroy if unreferenced.

This function is the same as the C++ method **OGRFeatureDefn::Release()** (p. ??).

#### Parameters

<i>hDefn</i>	handle to the feature definition to be released.
--------------	--

References **OGRFeatureDefn::FromHandle()**, and **OGRFeatureDefn::Release()**.

#### 12.15.4.109 OGR\_FD\_ReorderFieldDefns()

```
OGRErr OGR_FD_ReorderFieldDefns (
    OGRFeatureDefnH hDefn,
    int * panMap )
```

Reorder the field definitions in the array of the feature definition.

To reorder the field definitions in a layer definition, do not use this function directly, but use **OGR\_L\_ReorderFields()** (p. ??) instead.

This method should only be called while there are no **OGRFeature** (p. ??) objects in existence based on this **OGRFeatureDefn** (p. ??).

This method is the same as the C++ method **OGRFeatureDefn::ReorderFieldDefns()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition.
<i>panMap</i>	an array of <code>GetFieldCount()</code> elements which is a permutation of <code>[0, GetFieldCount()-1]</code> . <code>panMap</code> is such that, for each field definition at position <code>i</code> after reordering, its position before reordering was <code>panMap[i]</code> .

## Returns

`OGRERR_NONE` in case of success.

## Since

OGR 2.1.0

References `OGRFeatureDefn::FromHandle()`, and `OGRFeatureDefn::ReorderFieldDefns()`.

12.15.4.110 `OGR_FD_SetGeometryIgnored()`

```
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

<i>hDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
<i>bIgnore</i>	ignore state

References `OGRFeatureDefn::FromHandle()`, and `OGRFeatureDefn::SetGeometryIgnored()`.

12.15.4.111 `OGR_FD_SetGeomType()`

```
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

<i>hDefn</i>	handle to the layer or feature definition to set the geometry type to.
<i>eType</i>	the new type to assign.

References `OGRFeatureDefn::FromHandle()`, and `OGRFeatureDefn::SetGeomType()`.

12.15.4.112 `OGR_FD_SetStyleIgnored()`

```
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

<i>hDefn</i>	handle to the feature definition on witch <code>OGRFeature</code> (p. ??) are based on.
<i>bIgnore</i>	ignore state

References `OGRFeatureDefn::FromHandle()`, and `OGRFeatureDefn::SetStyleIgnored()`.

12.15.4.113 `OGR_Fld_Create()`

```
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

<i>pszName</i>	the name of the new field definition.
<i>eType</i>	the type of the new field definition.

## Returns

handle to the new field definition.

References `OGRFieldDefn::ToHandle()`.

#### 12.15.4.114 OGR\_Fld\_Destroy()

```
void OGR_Fld_Destroy (
    OGRFieldDefnH hDefn )
```

Destroy a field definition.

##### Parameters

<i>hDefn</i>	handle to the field definition to destroy.
--------------	--

References OGRFieldDefn::FromHandle().

#### 12.15.4.115 OGR\_Fld\_GetDefault()

```
const char* OGR_Fld_GetDefault (
    OGRFieldDefnH hDefn )
```

Get default field value.

This function is the same as the C++ method **OGRFieldDefn::GetDefault()** (p. ??).

##### Parameters

<i>hDefn</i>	handle to the field definition.
--------------	---------------------------------

##### Returns

default field value or NULL.

##### Since

GDAL 2.0

References OGRFieldDefn::FromHandle(), and OGRFieldDefn::GetDefault().

#### 12.15.4.116 OGR\_Fld\_GetJustify()

```
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**

<i>hDefn</i>	handle to the field definition to get justification from.
--------------	---

**Returns**

the justification.

References OGRFieldDefn::FromHandle(), and OGRFieldDefn::GetJustify().

**12.15.4.117 OGR\_Fld\_GetNameRef()**

```
const char* OGR_Fld_GetNameRef (
    OGRFieldDefnH hDefn )
```

Fetch name of this field.

This function is the same as the CPP method **OGRFieldDefn::GetNameRef()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the field definition.
--------------	---------------------------------

**Returns**

the name of the field definition.

References OGRFieldDefn::FromHandle(), and OGRFieldDefn::GetNameRef().

**12.15.4.118 OGR\_Fld\_GetPrecision()**

```
int OGR_Fld_GetPrecision (
    OGRFieldDefnH hDefn )
```

Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.

This function is the same as the CPP method **OGRFieldDefn::GetPrecision()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the field definition to get precision from.
--------------	---



**Returns**

the precision.

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::GetPrecision()`.

**12.15.4.119 OGR\_Fld\_GetSubType()**

```
OGRFieldType OGR_Fld_GetSubType (
    OGRFieldDefnH hDefn )
```

Fetch subtype of this field.

This function is the same as the CPP method `OGRFieldDefn::GetSubType()` (p. ??).

**Parameters**

<i>hDefn</i>	handle to the field definition to get subtype from.
--------------	---

**Returns**

field subtype.

**Since**

GDAL 2.0

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::GetSubType()`.

**12.15.4.120 OGR\_Fld\_GetType()**

```
OGRFieldType OGR_Fld_GetType (
    OGRFieldDefnH hDefn )
```

Fetch type of this field.

This function is the same as the CPP method `OGRFieldDefn::GetType()` (p. ??).

**Parameters**

<i>hDefn</i>	handle to the field definition to get type from.
--------------	--

**Returns**

field type.

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::GetType()`.

#### 12.15.4.121 `OGR_Fld_GetWidth()`

```
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

<code>hDefn</code>	handle to the field definition to get width from.
--------------------	---

##### Returns

the width, zero means no specified width.

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::GetWidth()`.

#### 12.15.4.122 `OGR_Fld_IsDefaultDriverSpecific()`

```
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 literal 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

<code>hDefn</code>	handle to the field definition
--------------------	--------------------------------

##### Returns

TRUE if the default value is driver specific.

##### Since

GDAL 2.0

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::IsDefaultDriverSpecific()`.

## 12.15.4.123 OGR\_Fld\_IsIgnored()

```
int OGR_Fld_IsIgnored (
    OGRFieldDefnH hDefn )
```

Return whether this field should be omitted when fetching features.

This method is the same as the C++ method **OGRFieldDefn::IsIgnored()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition
--------------	--------------------------------

## Returns

ignore state

References **OGRFieldDefn::FromHandle()**, and **OGRFieldDefn::IsIgnored()**.

## 12.15.4.124 OGR\_Fld\_IsNullable()

```
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

<i>hDefn</i>	handle to the field definition
--------------	--------------------------------

## Returns

TRUE if the field is authorized to be null.

## Since

GDAL 2.0

References **OGRFieldDefn::FromHandle()**, and **OGRFieldDefn::IsNullable()**.

## 12.15.4.125 OGR\_Fld\_Set()

```
void OGR_Fld_Set (
    OGRFieldDefnH hDefn,
    const char * pszNameIn,
    OGRFieldType eTypeIn,
    int nWidthIn,
    int nPrecisionIn,
    OGRJustification eJustifyIn )
```

Set defining parameters for a field in one call.

This function is the same as the CPP method **OGRFieldDefn::Set()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to set to.
<i>pszNameIn</i>	the new name to assign.
<i>eTypeIn</i>	the new type (one of the OFT values like OFTInteger).
<i>nWidthIn</i>	the preferred formatting width. Defaults to zero indicating undefined.
<i>nPrecisionIn</i>	number of decimals places for formatting, defaults to zero indicating undefined.
<i>eJustifyIn</i>	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References OGRFieldDefn::FromHandle().

## 12.15.4.126 OGR\_Fld\_SetDefault()

```
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 literal value enclosed between single quote characters (and inner single quote characters escaped by repetition of the single quote character), CURRENT\_TIMESTAMP, CURRENT\_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

<i>hDefn</i>	handle to the field definition.
<i>pszDefault</i>	new default field value or NULL pointer.

Since

GDAL 2.0

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::SetDefault()`.

#### 12.15.4.127 OGR\_Fld\_SetIgnored()

```
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

<i>hDefn</i>	handle to the field definition
<i>ignore</i>	ignore state

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::SetIgnored()`.

#### 12.15.4.128 OGR\_Fld\_SetJustify()

```
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

<i>hDefn</i>	handle to the field definition to set justification to.
<i>eJustify</i>	the new justification.

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::SetJustify()`.

**12.15.4.129 OGR\_Fld\_SetName()**

```
void OGR_Fld_SetName (
    OGRFieldDefnH hDefn,
    const char * pszName )
```

Reset the name of this field.

This function is the same as the CPP method **OGRFieldDefn::SetName()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the field definition to apply the new name to.
<i>pszName</i>	the new name to apply.

References **OGRFieldDefn::FromHandle()**, and **OGRFieldDefn::SetName()**.

**12.15.4.130 OGR\_Fld\_SetNullable()**

```
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**

<i>hDefn</i>	handle to the field definition
<i>bNullableIn</i>	FALSE if the field must have a not-null constraint.

**Since**

GDAL 2.0

References **OGRFieldDefn::FromHandle()**, and **OGRFieldDefn::SetNullable()**.

**12.15.4.131 OGR\_Fld\_SetPrecision()**

```
void OGR_Fld_SetPrecision (
    OGRFieldDefnH hDefn,
    int nPrecision )
```

Set the formatting precision for this field in characters.

This should normally be zero for fields of types other than OFTReal.

This function is the same as the CPP method **OGRFieldDefn::SetPrecision()** (p. ??).

#### Parameters

<i>hDefn</i>	handle to the field definition to set precision to.
<i>nPrecision</i>	the new precision.

References **OGRFieldDefn::FromHandle()**, and **OGRFieldDefn::SetPrecision()**.

#### 12.15.4.132 OGR\_Fld\_SetSubType()

```
void OGR_Fld_SetSubType (
    OGRFieldDefnH hDefn,
    OGRFieldType 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

<i>hDefn</i>	handle to the field definition to set type to.
<i>eSubType</i>	the new field subtype.

Since

GDAL 2.0

References **OGRFieldDefn::FromHandle()**, and **OGRFieldDefn::SetSubType()**.

#### 12.15.4.133 OGR\_Fld\_SetType()

```
void OGR_Fld_SetType (
    OGRFieldDefnH hDefn,
    OGRFieldType eType )
```

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. ??) that is already part of an **OGR↵FeatureDefn** (p. ??).

This function is the same as the CPP method **OGRFieldDefn::SetType()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to set type to.
<i>eType</i>	the new field type.

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::SetType()`.

12.15.4.134 `OGR_Fld_SetWidth()`

```
void OGR_Fld_SetWidth (
    OGRFieldDefnH hDefn,
    int nNewWidth )
```

Set the formatting width for this field in characters.

This function is the same as the CPP method `OGRFieldDefn::SetWidth()` (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to set width to.
<i>nNewWidth</i>	the new width.

References `OGRFieldDefn::FromHandle()`, and `OGRFieldDefn::SetWidth()`.

12.15.4.135 `OGR_G_AddGeometry()`

```
OGRERR OGR_G_AddGeometry (
    OGRGeometryH hGeom,
    OGRGeometryH hNewSubGeom )
```

Add a geometry to a geometry container.

Some subclasses of `OGRGeometryCollection` (p. ??) restrict the types of geometry that can be added, and may return an error. The passed geometry is cloned to make an internal copy.

There is no SFCOM analog to this method.

This function is the same as the CPP method `OGRGeometryCollection::addGeometry` (p. ??).

For a polygon, `hNewSubGeom` must be a linearring. If the polygon is empty, the first added subgeometry will be the exterior ring. The next ones will be the interior rings.

## Parameters

<i>hGeom</i>	existing geometry container.
<i>hNewSubGeom</i>	geometry to add to the container.



**Returns**

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of existing geometry.

< Unsupported operation

< Unsupported operation

< Unsupported geometry type

References OGR\_GT\_IsCurve(), OGR\_GT\_IsSubClassOf(), OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, OGRERR\_UNSUPPORTED\_OPERATION, VALIDATE\_POINTER1, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, and wkbPolyhedralSurface.

**12.15.4.136 OGR\_G\_AddGeometryDirectly()**

```
OGRERR OGR_G_AddGeometryDirectly (
    OGRGeometryH hGeom,
    OGRGeometryH hNewSubGeom )
```

Add a geometry directly to an existing geometry container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as addGeometry() does.

This function is the same as the CPP method **OGRGeometryCollection::addGeometryDirectly** (p. ??).

There is no SFCOM analog to this method.

For a polygon, hNewSubGeom must be a linearring. If the polygon is empty, the first added subgeometry will be the exterior ring. The next ones will be the interior rings.

**Parameters**

<i>hGeom</i>	existing geometry.
<i>hNewSubGeom</i>	geometry to add to the existing geometry.

**Returns**

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

< Unsupported operation

< Unsupported operation

< Unsupported geometry type

< Success

References `OGR_GT_IsCurve()`, `OGR_GT_IsSubClassOf()`, `OGRERR_NONE`, `OGRERR_UNSUPPORTED_`  
`GEOMETRY_TYPE`, `OGRERR_UNSUPPORTED_OPERATION`, `VALIDATE_POINTER1`, `wkbCompoundCurve`,  
`wkbCurvePolygon`, `wkbFlatten`, `wkbGeometryCollection`, and `wkbPolyhedralSurface`.

#### 12.15.4.137 OGR\_G\_AddPoint()

```
void OGR_G_AddPoint (
    OGRGeometryH hGeom,
    double dfX,
    double dfY,
    double dfZ )
```

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

##### Parameters

<i>hGeom</i>	handle to the geometry to add a point to.
<i>dfX</i>	x coordinate of point to add.
<i>dfY</i>	y coordinate of point to add.
<i>dfZ</i>	z coordinate of point to add.

References `OGRSimpleCurve::addPoint()`, `CPLError()`, `OGRPoint::setX()`, `OGRPoint::setY()`, `OGRPoint::setZ()`,  
`OGRGeometry::toPoint()`, `OGRGeometry::toSimpleCurve()`, `VALIDATE_POINTER0`, `wkbCircularString`, `wkb`  
`Flatten`, `wkbLineString`, and `wkbPoint`.

#### 12.15.4.138 OGR\_G\_AddPoint\_2D()

```
void OGR_G_AddPoint_2D (
    OGRGeometryH hGeom,
    double dfX,
    double dfY )
```

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

##### Parameters

<i>hGeom</i>	handle to the geometry to add a point to.
<i>dfX</i>	x coordinate of point to add.
<i>dfY</i>	y coordinate of point to add.

References `OGRSimpleCurve::addPoint()`, `CPLError()`, `OGRPoint::setX()`, `OGRPoint::setY()`, `OGRGeometry::to`  
`Point()`, `OGRGeometry::toSimpleCurve()`, `VALIDATE_POINTER0`, `wkbCircularString`, `wkbFlatten`, `wkbLineString`,  
and `wkbPoint`.

## 12.15.4.139 OGR\_G\_AddPointM()

```
void OGR_G_AddPointM (
    OGRGeometryH hGeom,
    double dfX,
    double dfY,
    double dfM )
```

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

## Parameters

<i>hGeom</i>	handle to the geometry to add a point to.
<i>dfX</i>	x coordinate of point to add.
<i>dfY</i>	y coordinate of point to add.
<i>dfM</i>	m coordinate of point to add.

References OGRSimpleCurve::addPointM(), CPLError(), OGRPoint::setM(), OGRPoint::setX(), OGRPoint::setY(), OGRGeometry::toPoint(), OGRGeometry::toSimpleCurve(), VALIDATE\_POINTER0, wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

## 12.15.4.140 OGR\_G\_AddPointZM()

```
void OGR_G_AddPointZM (
    OGRGeometryH hGeom,
    double dfX,
    double dfY,
    double dfZ,
    double dfM )
```

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

## Parameters

<i>hGeom</i>	handle to the geometry to add a point to.
<i>dfX</i>	x coordinate of point to add.
<i>dfY</i>	y coordinate of point to add.
<i>dfZ</i>	z coordinate of point to add.
<i>dfM</i>	m coordinate of point to add.

References OGRSimpleCurve::addPoint(), CPLError(), OGRPoint::setM(), OGRPoint::setX(), OGRPoint::setY(),

OGRPoint::setZ(), OGRGeometry::toPoint(), OGRGeometry::toSimpleCurve(), VALIDATE\_POINTER0, wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

#### 12.15.4.141 OGR\_G\_ApproximateArcAngles()

```
OGRGeometryH OGR_G_ApproximateArcAngles (
    double dfCenterX,
    double dfCenterY,
    double dfZ,
    double dfPrimaryRadius,
    double dfSecondaryRadius,
    double dfRotation,
    double dfStartAngle,
    double dfEndAngle,
    double dfMaxAngleStepSizeDegrees )
```

Stroke arc to linestring.

Stroke an arc of a circle to a linestring based on a center point, radius, start angle and end angle, all angles in degrees.

If the dfMaxAngleStepSizeDegrees is zero, then a default value will be used. This is currently 4 degrees unless the user has overridden the value with the OGR\_ARC\_STEPSIZE configuration variable.

See also

**CPLSetConfigOption()** (p. ??)

#### Parameters

<i>dfCenterX</i>	center X
<i>dfCenterY</i>	center Y
<i>dfZ</i>	center Z
<i>dfPrimaryRadius</i>	X radius of ellipse.
<i>dfSecondaryRadius</i>	Y radius of ellipse.
<i>dfRotation</i>	rotation of the ellipse clockwise.
<i>dfStartAngle</i>	angle to first point on arc (clockwise of X-positive)
<i>dfEndAngle</i>	angle to last point on arc (clockwise of X-positive)
<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.

#### Returns

**OGRLineString** (p. ??) geometry representing an approximation of the arc.

#### Since

OGR 1.8.0

References OGRGeometryFactory::approximateArcAngles().

## 12.15.4.142 OGR\_G\_Area()

```
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

<code>hGeom</code>	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()`, `VALIDATE_POI`, `NTERR1`, `wkbFlatten`, `wkbGeometryCollection`, and `wkbMultiSurface`.

Referenced by `OGRFeature::GetFieldAsDouble()`, `OGRFeature::GetFieldAsInteger64()`, `OGRFeature::GetFieldAsString()`, and `OGRFeature::IsFieldSet()`.

## 12.15.4.143 OGR\_G\_AssignSpatialReference()

```
void OGR_G_AssignSpatialReference (
    OGRGeometryH hGeom,
    OGRSpatialReferenceH hSRS )
```

Assign spatial reference to this object.

Any existing spatial reference is replaced, but under no circumstances does this result in the object being re-projected. It is just changing the interpretation of the existing geometry. Note that assigning a spatial reference increments the reference count on the **OGRSpatialReference** (p. ??), but does not copy it.

Starting with GDAL 2.3, this will also assign the spatial reference to potential sub-geometries of the geometry (**OGRGeometryCollection** (p. ??), **OGRCurvePolygon**/**OGRPolygon**, **OGRCompoundCurve** (p. ??), **OGRPolyhedralSurface** (p. ??) and their derived classes).

This is similar to the SFCOM `IGeometry::put_SpatialReference()` method.

This function is the same as the CPP method **OGRGeometry::assignSpatialReference** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to apply the new spatial reference system.
<i>hSRS</i>	handle on the new spatial reference system to apply.

References OGRGeometry::FromHandle(), OGRSpatialReference::FromHandle(), and VALIDATE\_POINTER0.

## 12.15.4.144 OGR\_G\_Boundary()

```
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

<i>hTarget</i>	The Geometry to calculate the boundary of.
----------------	--

## Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

## Since

OGR 1.8.0

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.145 OGR\_G\_Buffer()

```
OGRGeometryH OGR_G_Buffer (
    OGRGeometryH hTarget,
    double dfDist,
    int nQuadSegs )
```

Compute buffer of geometry.

Builds a new geometry containing the buffer region around the geometry on which it is invoked. The buffer is a polygon containing the region within the buffer distance of the original geometry.

Some buffer sections are properly described as curves, but are converted to approximate polygons. The `nQuadSegs` parameter can be used to control how many segments should be used to define a 90 degree curve - a quadrant of a circle. A value of 30 is a reasonable default. Large values result in large numbers of vertices in the resulting buffer geometry while small numbers reduce the accuracy of the result.

This function is the same as the C++ method **OGRGeometry::Buffer()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPLE_NotSupported` error.

#### Parameters

<i>hTarget</i>	the geometry.
<i>dfDist</i>	the buffer distance to be applied. Should be expressed into the same unit as the coordinates of the geometry.
<i>nQuadSegs</i>	the number of segments used to approximate a 90 degree (quadrant) of curvature.

#### Returns

the newly created geometry, or NULL if an error occurs.

References `OGRGeometry::FromHandle()`, `OGRGeometry::ToHandle()`, and `VALIDATE_POINTER1`.

#### 12.15.4.146 OGR\_G\_Centroid()

```
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.

< Failure

< Failure

< Failure

References `CPL_Error()`, `OGRGeometry::FromHandle()`, `OGRGeometry::getGeometryType()`, `OGRERR_FAILURE`, `VALIDATE_POINTER1`, `wkbFlatten`, and `wkbPoint`.

**12.15.4.147 OGR\_G\_Clone()**

```
OGRGeometryH OGR_G_Clone (
    OGRGeometryH hGeom )
```

Make a copy of this object.

This function relates to the SFCOM IGeometry::clone() method.

This function is the same as the CPP method **OGRGeometry::clone()** (p. ??).

**Parameters**

<i>hGeom</i>	handle on the geometry to clone from.
--------------	---------------------------------------

**Returns**

an handle on the copy of the geometry with the spatial reference system as the original.

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

**12.15.4.148 OGR\_G\_CloseRings()**

```
void OGR_G_CloseRings (
    OGRGeometryH hGeom )
```

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

**Parameters**

<i>hGeom</i>	handle to the geometry.
--------------	-------------------------

References OGRGeometry::closeRings(), OGRGeometry::FromHandle(), and VALIDATE\_POINTER0.

**12.15.4.149 OGR\_G\_Contains()**

```
int OGR_G_Contains (
    OGRGeometryH hThis,
    OGRGeometryH hOther )
```

Test for containment.

Tests if this geometry contains the other geometry.



This function is the same as the C++ method **OGRGeometry::Contains()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

## Returns

TRUE if *hThis* contains *hOther* geometry, otherwise FALSE.

References `OGRGeometry::FromHandle()`, and `VALIDATE_POINTER1`.

12.15.4.150 `OGR_G_ConvexHull()`

```
OGRGeometryH OGR_G_ConvexHull (
    OGRGeometryH hTarget )
```

Compute convex hull.

A new geometry object is created and returned containing the convex hull of the geometry on which the method is invoked.

This function is the same as the C++ method `OGRGeometry::ConvexHull()` (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPLE_NotSupported` error.

## Parameters

<i>hTarget</i>	The Geometry to calculate the convex hull of.
----------------	---

## Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

References `OGRGeometry::FromHandle()`, `OGRGeometry::ToHandle()`, and `VALIDATE_POINTER1`.

12.15.4.151 `OGR_G_CoordinateDimension()`

```
int OGR_G_CoordinateDimension (
    OGRGeometryH hGeom )
```

Get the dimension of the coordinates in this geometry.

This function is the same as the CPP method `OGRGeometry::CoordinateDimension()` (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to get the dimension of the coordinates from.
--------------	--

## Returns

this will return 2 for XY, 3 for XYZ and XYM, and 4 for XYZM data.

## Since

GDAL 2.1

References `OGRGeometry::CoordinateDimension()`, `OGRGeometry::FromHandle()`, and `VALIDATE_POINTER1`.

## 12.15.4.152 OGR\_G\_CreateFromFgf()

```
OGRERR OGR_G_CreateFromFgf (
    const void * pabyData,
    OGRSpatialReferenceH hSRS,
    OGRGeometryH * phGeometry,
    int nBytes,
    int * pnBytesConsumed )
```

Create a geometry object of the appropriate type from its FGF (FDO Geometry Format) binary representation.

See `OGRGeometryFactory::createFromFgf()` (p. ??)

References `OGRGeometryFactory::createFromFgf()`, and `OGRSpatialReference::FromHandle()`.

## 12.15.4.153 OGR\_G\_CreateFromGML()

```
OGRGeometryH OGR_G_CreateFromGML (
    const char * pszGML )
```

Create geometry from GML.

This method translates a fragment of GML containing only the geometry portion into a corresponding **OGR↔Geometry** (p. ??). There are many limitations on the forms of GML geometries supported by this parser, but they are too numerous to list here.

The following GML2 elements are parsed : Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, MultiGeometry.

(OGR >= 1.8.0) The following GML3 elements are parsed : Surface, MultiSurface, PolygonPatch, Triangle, Rectangle, Curve, MultiCurve, CompositeCurve, LineStringSegment, Arc, Circle, CompositeSurface, OrientableSurface, Solid, Tin, TriangulatedSurface.

Arc and Circle elements are stroked to linestring, by using a 4 degrees step, unless the user has overridden the value with the `OGR_ARC_STEPSIZE` configuration variable.

The C++ method `OGRGeometryFactory::createFromGML()` (p. ??) is the same as this function.

## Parameters

<i>pszGML</i>	The GML fragment for the geometry.
---------------	------------------------------------

## Returns

a geometry on success, or NULL on error.

References CPLError().

Referenced by OGRGeometryFactory::createFromGML().

## 12.15.4.154 OGR\_G\_CreateFromGMLTree()

```
OGRGeometryH OGR_G_CreateFromGMLTree (
    const CPLXMLNode * psTree )
```

Create geometry from GML

## 12.15.4.155 OGR\_G\_CreateFromWkb()

```
OGRERR OGR_G_CreateFromWkb (
    const void * pabyData,
    OGRSpatialReferenceH hSRS,
    OGRGeometryH * phGeometry,
    int nBytes )
```

Create a geometry object of the appropriate type from its well known binary representation.

Note that if nBytes is passed as zero, no checking can be done on whether the pabyData is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **OGR\_G\_WkbSize()** (p. ??) on the returned geometry to establish the number of bytes it required in WKB format.

The **OGRGeometryFactory::createFromWkb()** (p. ??) CPP method is the same as this function.

## Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>hSRS</i>	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>phGeometry</i>	the newly created geometry object will be assigned to the indicated handle on return. This will be NULL in case of failure. If not NULL, *phGeometry should be freed with <b>OGR_G_DestroyGeometry()</b> (p. ??) after use.
<i>nBytes</i>	the number of bytes of data available in pabyData, or -1 if it is not known, but assumed to be sufficient.

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

References OGRGeometryFactory::createFromWkb(), and OGRSpatialReference::FromHandle().

## 12.15.4.156 OGR\_G\_CreateFromWkt()

```
OGRERR OGR_G_CreateFromWkt (
    char ** ppszData,
    OGRSpatialReferenceH hSRS,
    OGRGeometryH * phGeometry )
```

Create a geometry object of the appropriate type from its well known text representation.

The **OGRGeometryFactory::createFromWkt** (p. ??) CPP method is the same as this function.

## Parameters

<i>ppszData</i>	input zero terminated string containing well known text representation of the geometry to be created. The pointer is updated to point just beyond that last character consumed.
<i>hSRS</i>	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>phGeometry</i>	the newly created geometry object will be assigned to the indicated handle on return. This will be NULL if the method fails. If not NULL, *phGeometry should be freed with <b>OGR_G_DestroyGeometry()</b> (p. ??) after use.

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

References OGRGeometryFactory::createFromWkt().

## 12.15.4.157 OGR\_G\_CreateGeometry()

```
OGRGeometryH OGR_G_CreateGeometry (
    OGRwkbGeometryType eGeometryType )
```

Create an empty geometry of desired type.

This is equivalent to allocating the desired geometry with new, but the allocation is guaranteed to take place in the context of the GDAL/OGR heap.

This function is the same as the CPP method **OGRGeometryFactory::createGeometry** (p. ??).

## Parameters

<i>eGeometryType</i>	the type code of the geometry to be created.
----------------------	--

## Returns

handle to the newly create geometry or NULL on failure. Should be freed with **OGR\_G\_DestroyGeometry()** (p. ??) after use.

References OGRGeometryFactory::createGeometry().

## 12.15.4.158 OGR\_G\_CreateGeometryFromJson()

```
OGRGeometryH OGR_G_CreateGeometryFromJson (
    const char * )
```

Create a OGR geometry from a GeoJSON geometry object

## 12.15.4.159 OGR\_G\_Crosses()

```
int OGR_G_Crosses (
    OGRGeometryH hThis,
    OGRGeometryH hOther )
```

Test for crossing.

Tests if this geometry and the other geometry are crossing.

This function is the same as the C++ method **OGRGeometry::Crosses()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

## Returns

TRUE if they are crossing, otherwise FALSE.

References OGRGeometry::FromHandle(), and VALIDATE\_POINTER1.

## 12.15.4.160 OGR\_G\_DelaunayTriangulation()

```
OGRGeometryH OGR_G_DelaunayTriangulation (
    OGRGeometryH hThis,
    double dfTolerance,
    int bOnlyEdges )
```

Return a Delaunay triangulation of the vertices of the geometry.

This function is the same as the C++ method **OGRGeometry::DelaunayTriangulation()** (p. ??).

This function is built on the GEOS library, v3.4 or above. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
<i>dfTolerance</i>	optional snapping tolerance to use for improved robustness
<i>bOnlyEdges</i>	if TRUE, will return a MULTILINESTRING, otherwise it will return a GEOMETRYCOLLECTION containing triangular POLYGONS.

## Returns

the geometry resulting from the Delaunay triangulation or NULL if an error occurs.

## Since

OGR 2.1

References **OGRGeometry::FromHandle()**, **OGRGeometry::ToHandle()**, and **VALIDATE\_POINTER1**.

## 12.15.4.161 OGR\_G\_DestroyGeometry()

```
void OGR_G_DestroyGeometry (
    OGRGeometryH hGeom )
```

Destroy geometry object.

Equivalent to invoking delete on a geometry, but it guaranteed to take place within the context of the GDAL/OGR heap.

This function is the same as the CPP method **OGRGeometryFactory::destroyGeometry** (p. ??).

## Parameters

<i>hGeom</i>	handle to the geometry to delete.
--------------	-----------------------------------

References **OGRGeometryFactory::destroyGeometry()**.

## 12.15.4.162 OGR\_G\_Difference()

```
OGRGeometryH OGR_G_Difference (
    OGRGeometryH hThis,
    OGRGeometryH hOther )
```

Compute difference.

Generates a new geometry which is the region of this geometry with the region of the other geometry removed.

This function is the same as the C++ method **OGRGeometry::Difference()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

## Returns

a new geometry representing the difference or NULL if the difference is empty or an error occurs.

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.163 OGR\_G\_Disjoint()

```
int OGR_G_Disjoint (
    OGRGeometryH hThis,
    OGRGeometryH hOther )
```

Test for disjointness.

Tests if this geometry and the other geometry are disjoint.

This function is the same as the C++ method **OGRGeometry::Disjoint()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.



**Returns**

TRUE if they are disjoint, otherwise FALSE.

References OGRGeometry::FromHandle(), and VALIDATE\_POINTER1.

**12.15.4.164 OGR\_G\_Distance()**

```
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**

<i>hFirst</i>	the first geometry to compare against.
<i>hOther</i>	the other geometry to compare against.

**Returns**

the distance between the geometries or -1 if an error occurs.

References OGRGeometry::FromHandle(), and VALIDATE\_POINTER1.

**12.15.4.165 OGR\_G\_Distance3D()**

```
double OGR_G_Distance3D (
    OGRGeometryH hFirst,
    OGRGeometryH hOther )
```

Returns the 3D distance between two geometries.

The distance is expressed into the same unit as the coordinates of the geometries.

This method is built on the SFCGAL library, check it for the definition of the geometry operation. If OGR is built without the SFCGAL library, this method will always return -1.0

This function is the same as the C++ method **OGRGeometry::Distance3D()** (p. ??).

**Parameters**

<i>hFirst</i>	the first geometry to compare against.
<i>hOther</i>	the other geometry to compare against.

**Returns**

distance between the two geometries

**Since**

GDAL 2.2

**Returns**

the distance between the geometries or -1 if an error occurs.

References `OGRGeometry::FromHandle()`, and `VALIDATE_POINTER1`.

**12.15.4.166 OGR\_G\_DumpReadable()**

```
void OGR_G_DumpReadable (
    OGRGeometryH hGeom,
    FILE * fp,
    const char * pszPrefix )
```

Dump geometry in well known text format to indicated output file.

This method is the same as the CPP method `OGRGeometry::dumpReadable` (p. ??).

**Parameters**

<i>hGeom</i>	handle on the geometry to dump.
<i>fp</i>	the text file to write the geometry to.
<i>pszPrefix</i>	the prefix to put on each line of output.

References `OGRGeometry::dumpReadable()`, `OGRGeometry::FromHandle()`, and `VALIDATE_POINTER0`.

**12.15.4.167 OGR\_G\_Empty()**

```
void OGR_G_Empty (
    OGRGeometryH hGeom )
```

Clear geometry information. This restores the geometry to its initial state after construction, and before assignment of actual geometry.

This function relates to the SFCOM IGeometry::Empty() method.

This function is the same as the CPP method **OGRGeometry::empty()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to empty.
--------------	----------------------------------

References `OGRGeometry::empty()`, `OGRGeometry::FromHandle()`, and `VALIDATE_POINTER0`.

12.15.4.168 `OGR_G_Equals()`

```
int OGR_G_Equals (
    OGRGeometryH hGeom,
    OGRGeometryH hOther )
```

Returns TRUE if two geometries are equivalent.

This operation implements the SQL/MM `ST_OrderingEquals()` operation.

The comparison is done in a structural way, that is to say that the geometry types must be identical, as well as the number and ordering of sub-geometries and vertices. Or equivalently, two geometries are considered equal by this method if their WKT/WKB representation is equal. Note: this must be distinguished for equality in a spatial way (which is the purpose of the `ST_Equals()` operation).

This function is the same as the CPP method `OGRGeometry::Equals()` (p. ??) method.

## Parameters

<i>hGeom</i>	handle on the first geometry.
<i>hOther</i>	handle on the other geometry to test against.

## Returns

TRUE if equivalent or FALSE otherwise.

References `CPL_Error()`, and `VALIDATE_POINTER1`.

12.15.4.169 `OGR_G_ExportEnvelopeToGMLTree()`

```
CPLXMLNode* OGR_G_ExportEnvelopeToGMLTree (
    OGRGeometryH hGeometry )
```

Export the envelope of a geometry as a `gml:Box`.

References `CPL_CreateXMLNode()`, `CPL_Error()`, and `CXT_Element`.

## 12.15.4.170 OGR\_G\_ExportToGML()

```
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()** (p. ??) when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToGML()** (p. ??).

## Parameters

<i>hGeometry</i>	handle to the geometry.
------------------	-------------------------

## Returns

A GML fragment or NULL in case of error.

References OGR\_G\_ExportToGMLEx().

Referenced by OGR\_G\_ExportToGMLTree().

## 12.15.4.171 OGR\_G\_ExportToGMLEx()

```
char* OGR_G_ExportToGMLEx (
    OGRGeometryH hGeometry,
    char ** ppszOptions )
```

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()** (p. ??) when no longer required.

The supported options are :

- **FORMAT=GML2/GML3/GML32** (GML2 or GML32 added in GDAL 2.1). If not set, 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**, deprecated by **SRSNAME\_FORMAT** in GDAL >=2.2). Defaults to YES. If YES, SRS with EPSG authority will be written with the "urn:ogc:def:crs:EP↵SG::" prefix. In the case the SRS is a SRS without explicit AXIS order, but that the same SRS authority code imported with **ImportFromEPSGA()** should be treated as lat/long or northing/easting, 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.

- **SRSNAME\_FORMAT=SHORT/OGC\_URN/OGC\_URL** (Only valid for **FORMAT=GML3**, added in GDAL 2.2). Defaults to **OGC\_URN**. If **SHORT**, then **srName** will be in the form **AUTHORITY\_NAME:AUTHORITY\_CODE**. If **OGC\_URN**, then **srName** will be in the form **urn:ogc:def:crs:AUTHORITY\_NAME::AUTHORITY\_CODE**. If **OGC\_URL**, then **srName** will be in the form **http://www.opengis.net/def/crs/AUTHORITY\_NAME/0/AUTHORITY\_CODE**. For **OGC\_URN** and **OGC\_URL**, in the case the SRS is a SRS without explicit **AXIS** order, but that the same SRS authority code imported with **ImportFromEPSGA()** should be treated as **lat/long** or **northing/easting**, then the function will take care of coordinate order swapping.
- **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/GML32**, GDAL >= 2.0) Default to **POSLIST**. For 2.5D geometries, define the location where to attach the **srDimension** attribute. There are diverging implementations. Some put in on the **<gml:posList>** element, other on the top geometry element.
- **NAMESPACE\_DECL=YES/NO**. If set to **YES**, **xmlns:gml="http://www.opengis.net/gml"** will be added to the root node for **GML < 3.2** or **xmlns:gml="http://www.opengis.net/gml/3.2"** for **GML 3.2**.

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

<i>hGeometry</i>	handle to the geometry.
<i>papszOptions</i>	NULL-terminated list of options.

#### Returns

A GML fragment or **NULL** in case of error.

#### Since

OGR 1.8.0

< Success

References **CPLError()**, **CPLMalloc()**, **CPLStrdup()**, **CPLTestBool()**, **CSLFetchNameValue()**, **CSLFetchNameValueDef()**, and **EQUAL**.

Referenced by **OGRGeometry::exportToGML()**, and **OGR\_G\_ExportToGML()**.

#### 12.15.4.172 OGR\_G\_ExportToGMLTree()

```
CPLXMLNode* OGR_G_ExportToGMLTree (
    OGRGeometryH hGeometry )
```

Convert a geometry into GML format.

References **CPLFree**, **CPLParseXMLString()**, and **OGR\_G\_ExportToGML()**.

## 12.15.4.173 OGR\_G\_ExportToIsoWkb()

```

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

<i>hGeom</i>	handle on the geometry to convert to a well know binary data from.
<i>eOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyDstBuffer</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGR_G_WkbSize()</b> (p. ??) byte in size.

## Returns

Currently OGRERR\_NONE is always returned.

## Since

GDAL 2.0

< Failure

References OGRGeometry::FromHandle(), OGRERR\_FAILURE, VALIDATE\_POINTER1, and wkbVariantIso.

## 12.15.4.174 OGR\_G\_ExportToIsoWkt()

```

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

<i>hGeom</i>	handle on the geometry to convert to a text format from.
<i>ppszSrcText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use,
Generated by Doxygen*ppszDstText should be freed with <b>CPLFree()</b> (p. ??).	

**Returns**

Currently OGRERR\_NONE is always returned.

**Since**

GDAL 2.0

## &lt; Failure

References OGRGeometry::FromHandle(), OGRERR\_FAILURE, VALIDATE\_POINTER1, and wkbVariantIso.

**12.15.4.175 OGR\_G\_ExportToJson()**

```
char* OGR_G_ExportToJson (
    OGRGeometryH hGeometry )
```

Convert a geometry into GeoJSON format.

The returned string should be freed with **CPLFree()** (p. ??) when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToJson()** (p. ??).

**Parameters**

<i>hGeometry</i>	handle to the geometry.
------------------	-------------------------

**Returns**

A GeoJSON fragment or NULL in case of error.

References OGR\_G\_ExportToJsonEx().

Referenced by OGRGeometry::exportToJson().

**12.15.4.176 OGR\_G\_ExportToJsonEx()**

```
char* OGR_G_ExportToJsonEx (
    OGRGeometryH hGeometry,
    char ** papszOptions )
```

Convert a geometry into GeoJSON format.

The returned string should be freed with **CPLFree()** (p. ??) when no longer required.

The following options are supported :



- COORDINATE\_PRECISION=number: maximum number of figures after decimal separator to write in coordinates.
- SIGNIFICANT\_FIGURES=number: maximum number of significant figures (GDAL >= 2.1).

If COORDINATE\_PRECISION is defined, SIGNIFICANT\_FIGURES will be ignored if specified. When none are defined, the default is COORDINATE\_PRECISION=15.

This method is the same as the C++ method **OGRGeometry::exportToJson()** (p. ??).

#### Parameters

<i>hGeometry</i>	handle to the geometry.
<i>papszOptions</i>	a null terminated list of options.

#### Returns

A GeoJSON fragment or NULL in case of error.

#### Since

OGR 1.9.0

References CPLStrdup(), CSLFetchNameValueDef(), and VALIDATE\_POINTER1.

Referenced by OGR\_G\_ExportToJson().

#### 12.15.4.177 OGR\_G\_ExportToKML()

```
char* OGR_G_ExportToKML (
    OGRGeometryH hGeometry,
    const char * pszAltitudeMode )
```

Convert a geometry into KML format.

The returned string should be freed with **CPLFree()** (p. ??) when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToKML()** (p. ??).

#### Parameters

<i>hGeometry</i>	handle to the geometry.
<i>pszAltitudeMode</i>	value to write in altitudeMode element, or NULL.

#### Returns

A KML fragment or NULL in case of error.

References CPLFree, CPLMalloc(), and CPLStrdup().

Referenced by OGRGeometry::exportToKML().

#### 12.15.4.178 OGR\_G\_ExportToWkb()

```
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** (p. ??)(OGRwkbByteOrder, unsigned char \*, OGRwkbVariant) with eWkbVariant = wkbVariantOldOgc.

##### Parameters

<i>hGeom</i>	handle on the geometry to convert to a well know binary data from.
<i>eOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyDstBuffer</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGR_G_WkbSize()</b> (p. ??) byte in size.

##### Returns

Currently OGRERR\_NONE is always returned.

< Failure

References OGRGeometry::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.15.4.179 OGR\_G\_ExportToWkt()

```
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

<i>hGeom</i>	handle on the geometry to convert to a text format from.
<i>ppszSrcText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with <b>CPLFree()</b> (p. ??).

## Returns

Currently OGRERR\_NONE is always returned.

< Failure

References OGRGeometry::exportToWkt(), OGRGeometry::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.15.4.180 OGR\_G\_FlattenTo2D()

```
void OGR_G_FlattenTo2D (
    OGRGeometryH hGeom )
```

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This function is the same as the CPP method **OGRGeometry::flattenTo2D()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to convert.
--------------	------------------------------------

References OGRGeometry::flattenTo2D(), and OGRGeometry::FromHandle().

## 12.15.4.181 OGR\_G\_ForceTo()

```
OGRGeometryH OGR_G_ForceTo (
    OGRGeometryH hGeom,
    OGRwkbGeometryType eTargetType,
    char ** ppszOptions )
```

Convert to another geometry type.

This function is the same as the C++ method **OGRGeometryFactory::forceTo()** (p. ??).

## Parameters

<i>hGeom</i>	the input geometry - ownership is passed to the method.
<i>eTargetType</i>	target output geometry type.
<i>ppszOptions</i>	options as a null-terminated list of strings or NULL.

**Returns**

new geometry.

**Since**

GDAL 2.0

References `OGRGeometryFactory::forceTo()`.

**12.15.4.182 OGR\_G\_ForceToLineString()**

```
OGRGeometryH OGR_G_ForceToLineString (
    OGRGeometryH hGeom )
```

Convert to line string.

This function is the same as the C++ method `OGRGeometryFactory::forceToLineString()` (p. ??).

**Parameters**

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

**Returns**

the converted geometry (ownership to caller).

**Since**

GDAL/OGR 1.10.0

References `OGRGeometryFactory::forceToLineString()`.

**12.15.4.183 OGR\_G\_ForceToMultiLineString()**

```
OGRGeometryH OGR_G_ForceToMultiLineString (
    OGRGeometryH hGeom )
```

Convert to multilinestring.

This function is the same as the C++ method `OGRGeometryFactory::forceToMultiLineString()` (p. ??).

**Parameters**

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

**Returns**

the converted geometry (ownership to caller).

**Since**

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiLineString().

**12.15.4.184 OGR\_G\_ForceToMultiPoint()**

```
OGRGeometryH OGR_G_ForceToMultiPoint (
    OGRGeometryH hGeom )
```

Convert to multipoint.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiPoint()** (p. ??).

**Parameters**

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

**Returns**

the converted geometry (ownership to caller).

**Since**

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiPoint().

**12.15.4.185 OGR\_G\_ForceToMultiPolygon()**

```
OGRGeometryH OGR_G_ForceToMultiPolygon (
    OGRGeometryH hGeom )
```

Convert to multipolygon.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiPolygon()** (p. ??).

**Parameters**

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

**Returns**

the converted geometry (ownership to caller).

**Since**

GDAL/OGR 1.8.0

References `OGRGeometryFactory::forceToMultiPolygon()`.

**12.15.4.186 OGR\_G\_ForceToPolygon()**

```
OGRGeometryH OGR_G_ForceToPolygon (
    OGRGeometryH hGeom )
```

Convert to polygon.

This function is the same as the C++ method `OGRGeometryFactory::forceToPolygon()` (p. ??).

**Parameters**

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

**Returns**

the converted geometry (ownership to caller).

**Since**

GDAL/OGR 1.8.0

References `OGRGeometryFactory::forceToPolygon()`.

**12.15.4.187 OGR\_G\_GetCoordinateDimension()**

```
int OGR_G_GetCoordinateDimension (
    OGRGeometryH hGeom )
```

Get the dimension of the coordinates in this geometry.

This function is the same as the CPP method `OGRGeometry::getCoordinateDimension()` (p. ??).

**Parameters**

<i>hGeom</i>	handle on the geometry to get the dimension of the coordinates from.
--------------	--

**Deprecated** use **OGR\_G\_CoordinateDimension()** (p. ??), **OGR\_G\_Is3D()** (p. ??), or **OGR\_G\_IsMeasured()** (p. ??).

#### Returns

this will return 2 or 3.

References **OGRGeometry::FromHandle()**, **OGRGeometry::getCoordinateDimension()**, and **VALIDATE\_POINTER1**.

#### 12.15.4.188 OGR\_G\_GetCurveGeometry()

```
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

<i>hGeom</i>	the geometry to operate on.
<i>papszOptions</i>	options as a null-terminated list of strings. Unused for now. Must be set to NULL.

#### Returns

a new geometry.

#### Since

GDAL 2.0

References **VALIDATE\_POINTER1**.

**12.15.4.189 OGR\_G\_GetDimension()**

```
int OGR_G_GetDimension (
    OGRGeometryH hGeom )
```

Get the dimension of this geometry.

This function corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the geometry, but does not indicate the dimension of the underlying space (as indicated by **OGR\_G\_GetCoordinateDimension()** (p. ??) function).

This function is the same as the CPP method **OGRGeometry::getDimension()** (p. ??).

**Parameters**

<i>hGeom</i>	handle on the geometry to get the dimension from.
--------------	---

**Returns**

0 for points, 1 for lines and 2 for surfaces.

References OGRGeometry::FromHandle(), OGRGeometry::getDimension(), and VALIDATE\_POINTER1.

**12.15.4.190 OGR\_G\_GetEnvelope()**

```
void OGR_G_GetEnvelope (
    OGRGeometryH hGeom,
    OGREnvelope * psEnvelope )
```

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This function is the same as the CPP method **OGRGeometry::getEnvelope()** (p. ??).

**Parameters**

<i>hGeom</i>	handle of the geometry to get envelope from.
<i>psEnvelope</i>	the structure in which to place the results.

References OGRGeometry::FromHandle(), OGRGeometry::getEnvelope(), and VALIDATE\_POINTER0.

**12.15.4.191 OGR\_G\_GetEnvelope3D()**

```
void OGR_G_GetEnvelope3D (
    OGRGeometryH hGeom,
    OGREnvelope3D * psEnvelope )
```

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This function is the same as the CPP method **OGRGeometry::getEnvelope()** (p. ??).



## Parameters

<i>hGeom</i>	handle of the geometry to get envelope from.
<i>psEnvelope</i>	the structure in which to place the results.

## Since

OGR 1.9.0

References OGRGeometry::FromHandle(), OGRGeometry::getEnvelope(), and VALIDATE\_POINTER0.

## 12.15.4.192 OGR\_G\_GetGeometryCount()

```
int OGR_G_GetGeometryCount (
    OGRGeometryH hGeom )
```

Fetch the number of elements in a geometry or number of geometries in container.

Only geometries of type wkbPolygon[25D], wkbMultiPoint[25D], wkbMultiLineString[25D], wkbMultiPolygon[25D] or wkbGeometryCollection[25D] may return a valid value. Other geometry types will silently return 0.

For a polygon, the returned number is the number of rings (exterior ring + interior rings).

## Parameters

<i>hGeom</i>	single geometry or geometry container from which to get the number of elements.
--------------	---

## Returns

the number of elements.

References OGR\_GT\_IsSubClassOf(), VALIDATE\_POINTER1, wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, and wkbPolyhedralSurface.

## 12.15.4.193 OGR\_G\_GetGeometryName()

```
const char* OGR_G_GetGeometryName (
    OGRGeometryH hGeom )
```

Fetch WKT name for geometry type.

There is no SFCOM analog to this function.

This function is the same as the CPP method **OGRGeometry::getGeometryName()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to get name from.
--------------	--

## Returns

name used for this geometry type in well known text format.

References `OGRGeometry::FromHandle()`, `OGRGeometry::getGeometryName()`, and `VALIDATE_POINTER1`.

12.15.4.194 `OGR_G_GetGeometryRef()`

```
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 until the next change to the geometry container. Use `OGR_G_Clone()` (p. ??) to make a copy.

This function relates to the SFCOM `IGeometryCollection::get_Geometry()` method.

This function is the same as the CPP method `OGRGeometryCollection::getGeometryRef()` (p. ??).

For a polygon, `OGR_G_GetGeometryRef(iSubGeom)` returns the exterior ring if `iSubGeom == 0`, and the interior rings for `iSubGeom > 0`.

## Parameters

<i>hGeom</i>	handle to the geometry container from which to get a geometry from.
<i>iSubGeom</i>	the index of the geometry to fetch, between 0 and <code>getNumGeometries()</code> - 1.

## Returns

handle to the requested geometry.

References `CPLError()`, `OGR_GT_IsSubClassOf()`, `VALIDATE_POINTER1`, `wkbCompoundCurve`, `wkbCurve`, `Polygon`, `wkbFlatten`, `wkbGeometryCollection`, and `wkbPolyhedralSurface`.

12.15.4.195 `OGR_G_GetGeometryType()`

```
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

<i>hGeom</i>	handle on the geometry to get type from.
--------------	--

## Returns

the geometry type code.

References `OGRGeometry::FromHandle()`, `OGRGeometry::getGeometryType()`, `VALIDATE_POINTER1`, and `wkbUnknown`.

12.15.4.196 `OGR_G_GetLinearGeometry()`

```
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

<i>hGeom</i>	the geometry to operate on.
<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.
<i>papszOptions</i>	options as a null-terminated list of strings or NULL. See <code>OGRGeometryFactory::curveToLineString()</code> (p. ??) for valid options.

## Returns

a new geometry.

## Since

GDAL 2.0

References `VALIDATE_POINTER1`.

## 12.15.4.197 OGR\_G\_GetM()

```
double OGR_G_GetM (
    OGRGeometryH hGeom,
    int i )
```

Fetch the m coordinate of a point from a geometry.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the M coordinate.
<i>i</i>	point to get the M coordinate.

## Returns

the M coordinate of this point.

References `VALIDATE_POINTER1`.

## 12.15.4.198 OGR\_G\_GetPoint()

```
void OGR_G_GetPoint (
    OGRGeometryH hGeom,
    int i,
    double * pdfX,
    double * pdfY,
    double * pdfZ )
```

Fetch a point in line string or a point geometry.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the coordinates.
<i>i</i>	the vertex to fetch, from 0 to <code>getNumPoints()-1</code> , zero for a point.
<i>pdfX</i>	value of x coordinate.
<i>pdfY</i>	value of y coordinate.
<i>pdfZ</i>	value of z coordinate.

References `CPLError()`, `OGRPoint::getX()`, `OGRPoint::getY()`, `OGRPoint::getZ()`, `OGRGeometry::toPoint()`, `VALIDATE_POINTER0`, `wkbFlatten`, and `wkbPoint`.

## 12.15.4.199 OGR\_G\_GetPointCount()

```
int OGR_G_GetPointCount (
    OGRGeometryH hGeom )
```

Fetch number of points from a geometry.

Only `wkbPoint[25D]` or `wkbLineString[25D]` may return a valid value. Other geometry types will silently return 0.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the number of points.
--------------	--

## Returns

the number of points.

References OGRCurve::getNumPoints(), OGR\_GT\_IsCurve(), OGRGeometry::toCurve(), VALIDATE\_POINTER1, wkbFlatten, and wkbPoint.

## 12.15.4.200 OGR\_G\_GetPoints()

```
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 consecutive elements of the array.

On some CPU architectures, care must be taken so that the arrays are properly aligned.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the coordinates.
<i>pabyX</i>	a buffer of at least (sizeof(double) * nXStride * nPointCount) bytes, may be NULL.
<i>nXStride</i>	the number of bytes between 2 elements of pabyX.
<i>pabyY</i>	a buffer of at least (sizeof(double) * nYStride * nPointCount) bytes, may be NULL.
<i>nYStride</i>	the number of bytes between 2 elements of pabyY.
<i>pabyZ</i>	a buffer of at last size (sizeof(double) * nZStride * nPointCount) bytes, may be NULL.
<i>nZStride</i>	the number of bytes between 2 elements of pabyZ.

## Returns

the number of points

## Since

OGR 1.9.0

References CPLError(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getPoints(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), OGRGeometry::toPoint(), OGRGeometry::toSimpleCurve(), VALIDATE\_POINTER1, wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

**12.15.4.201 OGR\_G\_GetPointsZM()**

```
int OGR_G_GetPointsZM (
    OGRGeometryH hGeom,
    void * pabyX,
    int nXStride,
    void * pabyY,
    int nYStride,
    void * pabyZ,
    int nZStride,
    void * pabyM,
    int nMStride )
```

Returns all points of line string.

This method copies all points into user arrays. The user provides the stride between 2 consecutive elements of the array.

On some CPU architectures, care must be taken so that the arrays are properly aligned.

**Parameters**

<i>hGeom</i>	handle to the geometry from which to get the coordinates.
<i>pabyX</i>	a buffer of at least (sizeof(double) * nXStride * nPointCount) bytes, may be NULL.
<i>nXStride</i>	the number of bytes between 2 elements of pabyX.
<i>pabyY</i>	a buffer of at least (sizeof(double) * nYStride * nPointCount) bytes, may be NULL.
<i>nYStride</i>	the number of bytes between 2 elements of pabyY.
<i>pabyZ</i>	a buffer of at last size (sizeof(double) * nZStride * nPointCount) bytes, may be NULL.
<i>nZStride</i>	the number of bytes between 2 elements of pabyZ.
<i>pabyM</i>	a buffer of at last size (sizeof(double) * nMStride * nPointCount) bytes, may be NULL.
<i>nMStride</i>	the number of bytes between 2 elements of pabyM.

**Returns**

the number of points

**Since**

OGR 1.9.0

References CPLError(), OGRPoint::getM(), OGRSimpleCurve::getNumPoints(), OGRSimpleCurve::getPoints(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), OGRGeometry::toPoint(), OGRGeometry::toSimpleCurve(), VALIDATE\_POINTER1, wkbCircularString, wkbFlatten, wkbLineString, and wkbPoint.

**12.15.4.202 OGR\_G\_GetPointZM()**

```
void OGR_G_GetPointZM (
    OGRGeometryH hGeom,
    int i,
```



```
double * pdfX,  
double * pdfY,  
double * pdfZ,  
double * pdfM )
```

Fetch a point in line string or a point geometry.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the coordinates.
<i>i</i>	the vertex to fetch, from 0 to getNumPoints()-1, zero for a point.
<i>pdfX</i>	value of x coordinate.
<i>pdfY</i>	value of y coordinate.
<i>pdfZ</i>	value of z coordinate.
<i>pdfM</i>	value of m coordinate.

References CPLError(), OGRPoint::getM(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), OGRGeometry::toPoint(), VALIDATE\_POINTER0, wkbFlatten, and wkbPoint.

## 12.15.4.203 OGR\_G\_GetSpatialReference()

```
OGRSpatialReferenceH OGR_G_GetSpatialReference (
    OGRGeometryH hGeom )
```

Returns spatial reference system for geometry.

This function relates to the SFCOM IGeometry::get\_SpatialReference() method.

This function is the same as the CPP method **OGRGeometry::getSpatialReference()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to get spatial reference from.
--------------	---

## Returns

a reference to the spatial reference geometry.

References OGRGeometry::FromHandle(), OGRSpatialReference::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.204 OGR\_G\_GetX()

```
double OGR_G_GetX (
    OGRGeometryH hGeom,
    int i )
```

Fetch the x coordinate of a point from a geometry.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the x coordinate.
<i>i</i>	point to get the x coordinate.

**Returns**

the X coordinate of this point.

References `VALIDATE_POINTER1`.

**12.15.4.205 OGR\_G\_GetY()**

```
double OGR_G_GetY (
    OGRGeometryH hGeom,
    int i )
```

Fetch the x coordinate of a point from a geometry.

**Parameters**

<i>hGeom</i>	handle to the geometry from which to get the y coordinate.
<i>i</i>	point to get the Y coordinate.

**Returns**

the Y coordinate of this point.

References `VALIDATE_POINTER1`.

**12.15.4.206 OGR\_G\_GetZ()**

```
double OGR_G_GetZ (
    OGRGeometryH hGeom,
    int i )
```

Fetch the z coordinate of a point from a geometry.

**Parameters**

<i>hGeom</i>	handle to the geometry from which to get the Z coordinate.
<i>i</i>	point to get the Z coordinate.

**Returns**

the Z coordinate of this point.

References `VALIDATE_POINTER1`.

**12.15.4.207 OGR\_G\_HasCurveGeometry()**

```
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**

<i>hGeom</i>	the geometry to operate on.
<i>bLookForNonLinear</i>	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

References `VALIDATE_POINTER1`.

**12.15.4.208 OGR\_G\_ImportFromWkb()**

```
OGRERR OGR_G_ImportFromWkb (
    OGRGeometryH hGeom,
    const void * pabyData,
    int nSize )
```

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type.

This function relates to the SFCOM IWks::ImportFromWKB() method.

This function is the same as the CPP method **OGRGeometry::importFromWkb()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to assign the well know binary data to.
<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or -1 if not known.

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

< Failure

References OGRGeometry::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.15.4.209 OGR\_G\_ImportFromWkt()

```
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

<i>hGeom</i>	handle on the geometry to assign well know text data to.
<i>ppszSrcText</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

< Failure

References OGRGeometry::FromHandle(), OGRGeometry::importFromWkt(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.15.4.210 OGR\_G\_Intersection()

```
OGRGeometryH OGR_G_Intersection (
    OGRGeometryH hThis,
    OGRGeometryH hOther )
```

Compute intersection.

Generates a new geometry which is the region of intersection of the two geometries operated on. The **OGR\_G\_Intersection()** (p. ??) function can be used to test if two geometries intersect.

This function is the same as the C++ method **OGRGeometry::Intersection()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

## Returns

a new geometry representing the intersection or NULL if there is no intersection or an error occurs.

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.211 OGR\_G\_Intersects()

```
int OGR_G_Intersects (
    OGRGeometryH hGeom,
    OGRGeometryH hOtherGeom )
```

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigorous fashion otherwise TRUE is returned if the envelopes (bounding boxes) of the two geometries overlap.

This function is the same as the CPP method **OGRGeometry::Intersects** (p. ??).

## Parameters

<i>hGeom</i>	handle on the first geometry.
<i>hOtherGeom</i>	handle on the other geometry to test against.

## Returns

TRUE if the geometries intersect, otherwise FALSE.

References OGRGeometry::FromHandle(), and VALIDATE\_POINTER1.

#### 12.15.4.212 OGR\_G\_Is3D()

```
int OGR_G_Is3D (
    OGRGeometryH hGeom )
```

See whether this geometry has Z coordinates.

This function is the same as the CPP method **OGRGeometry::Is3D()** (p. ??).

##### Parameters

<i>hGeom</i>	handle on the geometry to check whether it has Z coordinates.
--------------	---

##### Returns

TRUE if the geometry has Z coordinates.

##### Since

GDAL 2.1

References **OGRGeometry::FromHandle()**, **OGRGeometry::Is3D()**, and **VALIDATE\_POINTER1**.

#### 12.15.4.213 OGR\_G\_IsEmpty()

```
int OGR_G_IsEmpty (
    OGRGeometryH hGeom )
```

Test if the geometry is empty.

This method is the same as the CPP method **OGRGeometry::IsEmpty()** (p. ??).

##### Parameters

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

##### Returns

TRUE if the geometry has no points, otherwise FALSE.

References **OGRGeometry::FromHandle()**, **OGRGeometry::IsEmpty()**, and **VALIDATE\_POINTER1**.

**12.15.4.214 OGR\_G\_IsMeasured()**

```
int OGR_G_IsMeasured (
    OGRGeometryH hGeom )
```

See whether this geometry is measured.

This function is the same as the CPP method **OGRGeometry::IsMeasured()** (p. ??).

**Parameters**

<i>hGeom</i>	handle on the geometry to check whether it is measured.
--------------	---

**Returns**

TRUE if the geometry has M coordinates.

**Since**

GDAL 2.1

References **OGRGeometry::FromHandle()**, **OGRGeometry::IsMeasured()**, and **VALIDATE\_POINTER1**.

**12.15.4.215 OGR\_G\_IsRing()**

```
int OGR_G_IsRing (
    OGRGeometryH hGeom )
```

Test if the geometry is a ring.

This function is the same as the C++ method **OGRGeometry::IsRing()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always return FALSE.

**Parameters**

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

**Returns**

TRUE if the geometry has no points, otherwise FALSE.

References **OGRGeometry::FromHandle()**, **OGRGeometry::IsRing()**, and **VALIDATE\_POINTER1**.



## 12.15.4.216 OGR\_G\_IsSimple()

```
int OGR_G_IsSimple (
    OGRGeometryH hGeom )
```

Returns TRUE if the geometry is simple.

Returns TRUE if the geometry has no anomalous geometric points, such as self intersection or self tangency. The description of each instantiable geometric class will include the specific conditions that cause an instance of that class to be classified as not simple.

This function is the same as the C++ method **OGRGeometry::IsSimple()** (p. ??) method.

If OGR is built without the GEOS library, this function will always return FALSE.

## Parameters

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

## Returns

TRUE if object is simple, otherwise FALSE.

References OGRGeometry::FromHandle(), OGRGeometry::IsSimple(), and VALIDATE\_POINTER1.

## 12.15.4.217 OGR\_G\_IsValid()

```
int OGR_G_IsValid (
    OGRGeometryH hGeom )
```

Test if the geometry is valid.

This function is the same as the C++ method **OGRGeometry::IsValid()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always return FALSE.

## Parameters

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

## Returns

TRUE if the geometry has no points, otherwise FALSE.

References OGRGeometry::FromHandle(), OGRGeometry::IsValid(), and VALIDATE\_POINTER1.

**12.15.4.218 OGR\_G\_Length()**

```
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**

<i>hGeom</i>	the geometry to operate on.
--------------	-----------------------------

**Returns**

the length or 0.0 for unsupported geometry types.

**Since**

OGR 1.8.0

References `CPL_Error()`, `OGR_GT_IsCurve()`, `OGR_GT_IsSubClassOf()`, `VALIDATE_POINTER1`, `wkbFlatten`, `wkbGeometryCollection`, and `wkbMultiCurve`.

**12.15.4.219 OGR\_G\_Overlaps()**

```
int OGR_G_Overlaps (
    OGRGeometryH hThis,
    OGRGeometryH hOther )
```

Test for overlap.

Tests if this geometry and the other geometry overlap, that is their intersection has a non-zero area.

This function is the same as the C++ method **OGRGeometry::Overlaps()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPL_NotSupported` error.

**Parameters**

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

**Returns**

TRUE if they are overlapping, otherwise FALSE.

References OGRGeometry::FromHandle(), and VALIDATE\_POINTER1.

**12.15.4.220 OGR\_G\_PointOnSurface()**

```
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**

<i>hGeom</i>	the geometry to operate on.
--------------	-----------------------------

**Returns**

a point guaranteed to lie on the surface or NULL if an error occurred.

**Since**

OGR 1.10

References CPLError(), and VALIDATE\_POINTER1.

**12.15.4.221 OGR\_G\_Polygonize()**

```
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 topological inconsistencies.

This function is the same as the C++ method **OGRGeometry::Polygonize()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hTarget</i>	The Geometry to be polygonized.
----------------	---------------------------------

## Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

## Since

OGR 1.9.0

References `OGRGeometry::FromHandle()`, `OGRGeometry::ToHandle()`, and `VALIDATE_POINTER1`.

## 12.15.4.222 OGR\_G\_RemoveGeometry()

```
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. ??) for geometry collections, `OGRCurvePolygon::removeRing()` (p. ??) for polygons / curve polygons and `OGRPolyhedralSurface::removeGeometry()` (p. ??) for polyhedral surfaces and TINs.

## Parameters

<i>hGeom</i>	the existing geometry to delete from.
<i>iGeom</i>	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
<i>bDelete</i>	if TRUE the geometry will be destroyed, otherwise it will not. The default is TRUE as the existing geometry is considered to own the geometries in it.

## Returns

`OGRERR_NONE` if successful, or `OGRERR_FAILURE` if the index is out of range.

< Failure

< Unsupported operation

References `OGR_GT_IsSubClassOf()`, `OGRERR_FAILURE`, `VALIDATE_POINTER1`, `wkbCurvePolygon`, and `wkbFlatten`.

## 12.15.4.223 OGR\_G\_Segmentize()

```
void OGR_G_Segmentize (
    OGRGeometryH hGeom,
    double dfMaxLength )
```

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only.

This function is the same as the CPP method **OGRGeometry::segmentize()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to segmentize
<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization

References CPLError(), and VALIDATE\_POINTER0.

## 12.15.4.224 OGR\_G\_Set3D()

```
void OGR_G_Set3D (
    OGRGeometryH hGeom,
    int bIs3D )
```

Add or remove the Z coordinate dimension.

This method adds or removes the explicit Z coordinate dimension. Removing the Z coordinate dimension of a geometry will remove any existing Z values. Adding the Z dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

## Parameters

<i>hGeom</i>	handle on the geometry to set or unset the Z dimension.
<i>bIs3D</i>	Should the geometry have a Z dimension, either TRUE or FALSE.

Since

GDAL 2.1

References OGRGeometry::FromHandle(), OGRGeometry::set3D(), and VALIDATE\_POINTER0.

## 12.15.4.225 OGR\_G\_SetCoordinateDimension()

```
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, a compound curve, a polygon, etc. will affect the children geometries. This will also remove the M dimension if present before this call.

**Deprecated** use **OGR\_G\_Set3D()** (p. ??) or **OGR\_G\_SetMeasured()** (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to set the dimension of the coordinates.
<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.

References OGRGeometry::FromHandle(), and VALIDATE\_POINTER0.

#### 12.15.4.226 OGR\_G\_SetMeasured()

```
void OGR_G_SetMeasured (
    OGRGeometryH hGeom,
    int bIsMeasured )
```

Add or remove the M coordinate dimension.

This method adds or removes the explicit M coordinate dimension. Removing the M coordinate dimension of a geometry will remove any existing M values. Adding the M dimension to a geometry collection, a compound curve, a polygon, etc. will affect the children geometries.

#### Parameters

<i>hGeom</i>	handle on the geometry to set or unset the M dimension.
<i>bIsMeasured</i>	Should the geometry have a M dimension, either TRUE or FALSE.

Since

GDAL 2.1

References OGRGeometry::FromHandle(), OGRGeometry::setMeasured(), and VALIDATE\_POINTER0.

#### 12.15.4.227 OGR\_G\_SetPoint()

```
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

<i>hGeom</i>	handle to the geometry to add a vertex to.
<i>i</i>	the index of the vertex to assign (zero based) or zero for a point.
<i>dfX</i>	input X coordinate to assign.
<i>dfY</i>	input Y coordinate to assign.
<i>dfZ</i>	input Z coordinate to assign (defaults to zero).

References `CPL::Error()`, `OGRPoint::setX()`, `OGRPoint::setY()`, `OGRPoint::setZ()`, `OGRGeometry::toPoint()`, `VALIDATE_POINTER0`, `wkbFlatten`, and `wkbPoint`.

#### 12.15.4.228 OGR\_G\_SetPoint\_2D()

```
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

<i>hGeom</i>	handle to the geometry to add a vertex to.
<i>i</i>	the index of the vertex to assign (zero based) or zero for a point.
<i>dfX</i>	input X coordinate to assign.
<i>dfY</i>	input Y coordinate to assign.

References `CPL::Error()`, `OGRPoint::setX()`, `OGRPoint::setY()`, `OGRGeometry::toPoint()`, `VALIDATE_POINTER0`, `wkbFlatten`, and `wkbPoint`.

#### 12.15.4.229 OGR\_G\_SetPointCount()

```
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

<i>hGeom</i>	handle to the geometry.
<i>nNewPointCount</i>	the new number of points for geometry.

References `CPLError()`, `OGRSimpleCurve::setNumPoints()`, `OGRGeometry::toSimpleCurve()`, `VALIDATE_POINTER0`, `wkbCircularString`, `wkbFlatten`, and `wkbLineString`.

#### 12.15.4.230 OGR\_G\_SetPointM()

```
void OGR_G_SetPointM (
    OGRGeometryH hGeom,
    int i,
    double dfX,
    double dfY,
    double dfM )
```

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

<i>hGeom</i>	handle to the geometry to add a vertex to.
<i>i</i>	the index of the vertex to assign (zero based) or zero for a point.
<i>dfX</i>	input X coordinate to assign.
<i>dfY</i>	input Y coordinate to assign.
<i>dfM</i>	input M coordinate to assign.

References `CPLError()`, `OGRPoint::setM()`, `OGRPoint::setX()`, `OGRPoint::setY()`, `OGRGeometry::toPoint()`, `VALIDATE_POINTER0`, `wkbFlatten`, and `wkbPoint`.

#### 12.15.4.231 OGR\_G\_SetPoints()

```
void OGR_G_SetPoints (
    OGRGeometryH hGeom,
    int nPointsIn,
    const void * pabyX,
    int nXStride,
    const void * pabyY,
```



```

    int nYStride,
    const 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

<i>hGeom</i>	handle to the geometry to set the coordinates.
<i>n</i> ↔ <i>PointsIn</i>	number of points being passed in padfX and padfY.
<i>pabyX</i>	list of X coordinates (double values) of points being assigned.
<i>nXStride</i>	the number of bytes between 2 elements of pabyX.
<i>pabyY</i>	list of Y coordinates (double values) of points being assigned.
<i>nYStride</i>	the number of bytes between 2 elements of pabyY.
<i>pabyZ</i>	list of Z coordinates (double values) of points being assigned (defaults to NULL for 2D objects).
<i>nZStride</i>	the number of bytes between 2 elements of pabyZ.

References CPLError(), and VALIDATE\_POINTER0.

#### 12.15.4.232 OGR\_G\_SetPointsZM()

```

void OGR_G_SetPointsZM (
    OGRGeometryH hGeom,
    int nPointsIn,
    const void * pX,
    int nXStride,
    const void * pY,
    int nYStride,
    const void * pZ,
    int nZStride,
    const void * pM,
    int nMStride )

```

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

<i>hGeom</i>	handle to the geometry to set the coordinates.
<i>n</i> ↔ <i>PointsIn</i>	number of points being passed in padfX and padfY.
<i>pX</i>	list of X coordinates (double values) of points being assigned.
<i>nXStride</i>	the number of bytes between 2 elements of pX.
<i>pY</i>	list of Y coordinates (double values) of points being assigned.
<i>nYStride</i>	the number of bytes between 2 elements of pY.
<i>pZ</i>	list of Z coordinates (double values) of points being assigned (if not NULL, upgrades the geometry to have Z coordinate).

## Parameters

<i>nZStride</i>	the number of bytes between 2 elements of pZ.
<i>pM</i>	list of M coordinates (double values) of points being assigned (if not NULL, upgrades the geometry to have M coordinate).
<i>nMStride</i>	the number of bytes between 2 elements of pM.

References CPLError(), and VALIDATE\_POINTER0.

## 12.15.4.233 OGR\_G\_SetPointZM()

```
void OGR_G_SetPointZM (
    OGRGeometryH hGeom,
    int i,
    double dfX,
    double dfY,
    double dfZ,
    double dfM )
```

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

<i>hGeom</i>	handle to the geometry to add a vertex to.
<i>i</i>	the index of the vertex to assign (zero based) or zero for a point.
<i>dfX</i>	input X coordinate to assign.
<i>dfY</i>	input Y coordinate to assign.
<i>dfZ</i>	input Z coordinate to assign.
<i>dfM</i>	input M coordinate to assign.

References CPLError(), OGRPoint::setM(), OGRPoint::setX(), OGRPoint::setY(), OGRPoint::setZ(), OGRGeometry::toPoint(), VALIDATE\_POINTER0, wkbFlatten, and wkbPoint.

## 12.15.4.234 OGR\_G\_Simplify()

```
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 CPL\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
<i>dTolerance</i>	the distance tolerance for the simplification.

## Returns

the simplified geometry or NULL if an error occurs.

## Since

OGR 1.8.0

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.235 OGR\_G\_SimplifyPreserveTopology()

```
OGRGeometryH OGR_G_SimplifyPreserveTopology (
    OGRGeometryH hThis,
    double dTolerance )
```

Simplify the geometry while preserving topology.

This function is the same as the C++ method **OGRGeometry::SimplifyPreserveTopology()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
<i>dTolerance</i>	the distance tolerance for the simplification.

## Returns

the simplified geometry or NULL if an error occurs.

## Since

OGR 1.9.0

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.236 OGR\_G\_SwapXY()

```
void OGR_G_SwapXY (
    OGRGeometryH hGeom )
```

Swap x and y coordinates.

## Parameters

<i>hGeom</i>	geometry.
--------------	-----------

## Since

OGR 2.3.0

References OGRGeometry::FromHandle(), OGRGeometry::swapXY(), and VALIDATE\_POINTER0.

## 12.15.4.237 OGR\_G\_SymDifference()

```
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().

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

## Returns

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

## Since

OGR 1.8.0

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.238 OGR\_G\_Touches()

```
int OGR_G_Touches (
    OGRGeometryH hThis,
    OGRGeometryH hOther )
```

Test for touching.

Tests if this geometry and the other geometry are touching.

This function is the same as the C++ method **OGRGeometry::Touches()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

#### Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

#### Returns

TRUE if they are touching, otherwise FALSE.

References OGRGeometry::FromHandle(), and VALIDATE\_POINTER1.

#### 12.15.4.239 OGR\_G\_Transform()

```
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

<i>hGeom</i>	handle on the geometry to apply the transform to.
<i>hTransform</i>	handle on the transformation to apply.

#### Returns

OGRERR\_NONE on success or an error code.

< Failure

References `OGRGeometry::FromHandle()`, `OGRCoordinateTransformation::FromHandle()`, `OGRERR_FAILURE`, and `VALIDATE_POINTER1`.

#### 12.15.4.240 `OGR_G_TransformTo()`

```
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 C++ method **OGRGeometry::transformTo** (p. ??).

##### Parameters

<i>hGeom</i>	handle on the geometry to apply the transform to.
<i>hSRS</i>	handle on the spatial reference system to apply.

##### Returns

`OGRERR_NONE` on success, or an error code.

< Failure

References `OGRGeometry::FromHandle()`, `OGRSpatialReference::FromHandle()`, `OGRERR_FAILURE`, and `VALIDATE_POINTER1`.

#### 12.15.4.241 `OGR_G_Union()`

```
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 `CPL_NotSupported` error.

## Parameters

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

## Returns

a new geometry representing the union or NULL if an error occurs.

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.242 OGR\_G\_UnionCascaded()

```
OGRGeometryH OGR_G_UnionCascaded (
    OGRGeometryH hThis )
```

Compute union using cascading.

This function is the same as the C++ method **OGRGeometry::UnionCascaded()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
--------------	---------------

## Returns

a new geometry representing the union or NULL if an error occurs.

References OGRGeometry::FromHandle(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

## 12.15.4.243 OGR\_G\_Value()

```
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**

<i>hGeom</i>	curve geometry.
<i>dfDistance</i>	distance along the curve at which to sample position. This distance should be between zero and <code>get_Length()</code> for this curve.

**Returns**

a point or NULL.

**Since**

GDAL 2.0

References `OGR_GT_IsCurve()`, and `VALIDATE_POINTER1`.

**12.15.4.244 OGR\_G\_Within()**

```
int OGR_G_Within (
    OGRGeometryH hThis,
    OGRGeometryH hOther )
```

Test for containment.

Tests if this geometry is within the other geometry.

This function is the same as the C++ method **OGRGeometry::Within()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPLE_NotSupported` error.

**Parameters**

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

**Returns**

TRUE if *hThis* is within *hOther*, otherwise FALSE.

References `OGRGeometry::FromHandle()`, and `VALIDATE_POINTER1`.

**12.15.4.245 OGR\_G\_WkbSize()**

```
int OGR_G_WkbSize (
    OGRGeometryH hGeom )
```



Returns size of related binary representation.

This function returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This function relates to the SFCOM IWks::WkbSize() method.

This function is the same as the CPP method **OGRGeometry::WkbSize()** (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to get the binary size from.
--------------	---

#### Returns

size of binary representation in bytes.

References OGRGeometry::FromHandle(), VALIDATE\_POINTER1, and OGRGeometry::WkbSize().

#### 12.15.4.246 OGR\_GetFieldSubTypeName()

```
const char* OGR_GetFieldSubTypeName (
    OGRFieldType eSubType )
```

Fetch human readable name for a field subtype.

This function is the same as the CPP method **OGRFieldDefn::GetFieldSubTypeName()** (p. ??).

#### Parameters

<i>eSubType</i>	the field subtype to get name for.
-----------------	------------------------------------

#### Returns

the name.

#### Since

GDAL 2.0

References OGRFieldDefn::GetFieldSubTypeName().

#### 12.15.4.247 OGR\_GetFieldTypeNames()

```
const char* OGR_GetFieldTypeNames (
    OGRFieldType eType )
```

Fetch human readable name for a field type.

This function is the same as the CPP method **OGRFieldDefn::GetFieldTypeNames()** (p. ??).

## Parameters

<i>eType</i>	the field type to get name for.
--------------	---------------------------------

## Returns

the name.

References `OGRFieldDefn::GetFieldName()`.

12.15.4.248 `OGR_GFld_Create()`

```
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

<i>pszName</i>	the name of the new field definition.
<i>eType</i>	the type of the new field definition.

## Returns

handle to the new field definition.

## Since

GDAL 1.11

References `OGRGeomFieldDefn::ToHandle()`.

12.15.4.249 `OGR_GFld_Destroy()`

```
void OGR_GFld_Destroy (
    OGRGeomFieldDefnH hDefn )
```

Destroy a geometry field definition.

## Parameters

<i>hDefn</i>	handle to the geometry field definition to destroy.
--------------	---

Since

GDAL 1.11

References `OGRGeomFieldDefn::FromHandle()`, and `VALIDATE_POINTER0`.

#### 12.15.4.250 `OGR_GFld_GetNameRef()`

```
const char* OGR_GFld_GetNameRef (
    OGRGeomFieldDefnH hDefn )
```

Fetch name of this field.

This function is the same as the C++ method `OGRGeomFieldDefn::GetNameRef()` (p. ??).

Parameters

<i>hDefn</i>	handle to the geometry field definition.
--------------	--

Returns

the name of the geometry field definition.

Since

GDAL 1.11

References `OGRGeomFieldDefn::FromHandle()`, `OGRGeomFieldDefn::GetNameRef()`, and `VALIDATE_POINTER1`.

#### 12.15.4.251 `OGR_GFld_GetSpatialRef()`

```
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

<i>hDefn</i>	handle to the geometry field definition
--------------	---

**Returns**

field spatial reference system.

**Since**

GDAL 1.11

References `OGRGeomFieldDefn::FromHandle()`, `OGRGeomFieldDefn::GetSpatialRef()`, and `VALIDATE_POINT↔ER1`.

**12.15.4.252 OGR\_GFld\_GetType()**

```
OGRwkbGeometryType OGR_GFld_GetType (
    OGRGeomFieldDefnH hDefn )
```

Fetch geometry type of this field.

This function is the same as the C++ method `OGRGeomFieldDefn::GetType()` (p. ??).

**Parameters**

<i>hDefn</i>	handle to the geometry field definition to get type from.
--------------	---

**Returns**

field geometry type.

**Since**

GDAL 1.11

References `OGRGeomFieldDefn::FromHandle()`, `OGRGeomFieldDefn::GetType()`, `OGR_GT_GetLinear()`, `OGR↔_GT_IsNonLinear()`, `OGRGetNonLinearGeometriesEnabledFlag()`, `VALIDATE_POINTER1`, and `wkbUnknown`.

**12.15.4.253 OGR\_GFld\_IsIgnored()**

```
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

<i>hDefn</i>	handle to the geometry field definition
--------------	---

## Returns

ignore state

## Since

GDAL 1.11

References `OGRGeomFieldDefn::FromHandle()`, `OGRGeomFieldDefn::IsIgnored()`, and `VALIDATE_POINTER1`.

12.15.4.254 `OGR_GFId_IsNullable()`

```
int OGR_GFId_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

<i>hDefn</i>	handle to the field definition
--------------	--------------------------------

## Returns

`TRUE` if the field is authorized to be null.

## Since

GDAL 2.0

References `OGRGeomFieldDefn::FromHandle()`, and `OGRGeomFieldDefn::IsNullable()`.

**12.15.4.255 OGR\_GFld\_SetIgnored()**

```
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**

<i>hDefn</i>	handle to the geometry field definition
<i>ignore</i>	ignore state

Since

GDAL 1.11

References **OGRGeomFieldDefn::FromHandle()**, **OGRGeomFieldDefn::SetIgnored()**, and **VALIDATE\_POINTER0**.

**12.15.4.256 OGR\_GFld\_SetName()**

```
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**

<i>hDefn</i>	handle to the geometry field definition to apply the new name to.
<i>pszName</i>	the new name to apply.

Since

GDAL 1.11

References **OGRGeomFieldDefn::FromHandle()**, **OGRGeomFieldDefn::SetName()**, and **VALIDATE\_POINTER0**.

**12.15.4.257 OGR\_GFld\_SetNullable()**

```
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

<i>hDefn</i>	handle to the field definition
<i>b↔ NullableIn</i>	FALSE if the field must have a not-null constraint.

#### Since

GDAL 2.0

References **OGRGeomFieldDefn::FromHandle()**, and **OGRGeomFieldDefn::SetNullable()**.

#### 12.15.4.258 OGR\_GFld\_SetSpatialRef()

```
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

<i>hDefn</i>	handle to the geometry field definition
<i>hSRS</i>	the new SRS to apply.

#### Since

GDAL 1.11

References **OGRGeomFieldDefn::FromHandle()**, and **VALIDATE\_POINTER0**.

## 12.15.4.259 OGR\_GFld\_SetType()

```
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

<i>hDefn</i>	handle to the geometry field definition to set type to.
<i>eType</i>	the new field geometry type.

Since

GDAL 1.11

References **OGRGeomFieldDefn::FromHandle()**, **OGRGeomFieldDefn::SetType()**, and **VALIDATE\_POINTER0**.

## 12.15.4.260 OGR\_L\_AlterFieldDefn()

```
OGRErr OGR_L_AlterFieldDefn (
    OGRLayerH hLayer,
    int iField,
    OGRFieldDefnH hNewFieldDefn,
    int nFlags )
```

Alter the definition of an existing field on a layer.

You must use this to alter the definition of an existing field of a real layer. Internally the **OGRFeatureDefn** (p. ??) for the layer will be updated to reflect the altered field. Applications should never modify the **OGRFeatureDefn** (p. ??) used by a layer directly.

This function should not be called while there are feature objects in existence that were obtained or created with the previous layer definition.

Not all drivers support this function. You can query a layer to check if it supports it with the **OLCAAlterFieldDefn** capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existing features of the backing file/database should be updated accordingly. Some drivers might also not support all update flags.

This function is the same as the C++ method **OGRLayer::AlterFieldDefn()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer.
<i>iField</i>	index of the field whose definition must be altered.
<i>hNewFieldDefn</i>	new field definition
<i>nFlags</i>	combination of <b>ALTER_NAME_FLAG</b> , <b>ALTER_TYPE_FLAG</b> , <b>ALTER_WIDTH_PRECISION_FLAG</b> , <b>ALTER_NULLABLE_FLAG</b> and <b>ALTER_DEFAULT_FLAG</b> 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

References OGRLayer::AlterFieldDefn(), OGRFieldDefn::FromHandle(), OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

## 12.15.4.261 OGR\_L\_Clip()

```
OGRERR OGR_L_Clip (
    OGRLayerH pLayerInput,
    OGRLayerH pLayerMethod,
    OGRLayerH pLayerResult,
    char ** ppszOptions,
    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 or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

This function is the same as the C++ method **OGRLayer::Clip()** (p. ??).

## Parameters

<i>pLayerInput</i>	the input layer. Should not be NULL.
<i>pLayerMethod</i>	the method layer. Should not be NULL.
<i>pLayerResult</i>	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
<i>ppszOptions</i>	NULL terminated list of options (may be NULL).
<i>pfnProgress</i>	a GDALProgressFunc() compatible callback function for reporting progress or NULL.

**Returns**

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

**Note**

The first geometry field is always used.

**Since**

OGR 1.10

References OGRLayer::Clip(), OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

**12.15.4.262 OGR\_L\_CommitTransaction()**

```
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()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer
---------------	---------------------

**Returns**

OGRERR\_NONE on success.

References OGRLayer::CommitTransaction(), OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

**12.15.4.263 OGR\_L\_CreateFeature()**

```
OGRERR OGR_L_CreateFeature (
    OGRLayerH hLayer,
    OGRFeatureH hFeat )
```

Create and write a new feature within a layer.

The passed feature is written to the layer as a new feature, rather than overwriting an existing one. If the feature has a feature id other than OGRNullFID, then the native implementation may use that as the feature id of the new feature, but not necessarily. Upon successful return the passed feature will have been updated with the new feature id.

This function is the same as the C++ method **OGRLayer::CreateFeature()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to write the feature to.
<i>hFeat</i>	the handle of the feature to write to disk.

## Returns

OGRERR\_NONE on success.

References OGRLayer::CreateFeature(), OGRLayer::FromHandle(), OGRFeature::FromHandle(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

## 12.15.4.264 OGR\_L\_CreateField()

```
OGRErr OGR_L_CreateField (
    OGRLayerH hLayer,
    OGRFieldDefnH hField,
    int bApproxOK )
```

Create a new field on a layer.

You must use this to create new fields on a real layer. Internally the **OGRFeatureDefn** (p. ??) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. ??) used by a layer directly.

This function should not be called while there are feature objects in existence that were obtained or created with the previous layer definition.

Not all drivers support this function. You can query a layer to check if it supports it with the **OLCCreateField** capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existing features of the backing file/database should be updated accordingly.

Drivers may or may not support not-null constraints. If they support creating fields with not-null constraints, this is generally before creating any feature to the layer.

This function is the same as the C++ method **OGRLayer::CreateField()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to write the field definition.
<i>hField</i>	handle of the field definition to write to disk.
<i>bApproxOK</i>	If TRUE, the field may be created in a slightly different form depending on the limitations of the format driver.

## Returns

OGRERR\_NONE on success.

References OGRLayer::CreateField(), OGRFieldDefn::FromHandle(), OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

## 12.15.4.265 OGR\_L\_CreateGeomField()

```

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 existence that were obtained or created with the previous layer definition.

Not all drivers support this function. You can query a layer to check if it supports it with the **OLCCreateField** capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existing features of the backing file/database should be updated accordingly.

Drivers may or may not support not-null constraints. If they support creating fields with not-null constraints, this is generally before creating any feature to the layer.

This function is the same as the C++ method **OGRLayer::CreateField()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to write the field definition.
<i>hField</i>	handle of the geometry field definition to write to disk.
<i>bApproxOK</i>	If TRUE, the field may be created in a slightly different form depending on the limitations of the format driver.

## Returns

OGRERR\_NONE on success.

## Since

OGR 1.11

References **OGRLayer::CreateGeomField()**, **OGRGeomFieldDefn::FromHandle()**, **OGRLayer::FromHandle()**, **OGRERR\_INVALID\_HANDLE**, and **VALIDATE\_POINTER1**.

## 12.15.4.266 OGR\_L\_DeleteFeature()

```

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

<i>hLayer</i>	handle to the layer
<i>nFID</i>	the feature id to be deleted from the layer

## Returns

OGRERR\_NONE if the operation works, otherwise an appropriate error code (e.g OGRERR\_NON\_EXISTING\_FEATURE if the feature does not exist).

References OGRLayer::DeleteFeature(), OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

## 12.15.4.267 OGR\_L\_DeleteField()

```
OGRERR OGR_L_DeleteField (
    OGRLayerH hLayer,
    int iField )
```

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 function should not be called while there are feature objects in existence that were obtained or created with the previous layer definition.

Not all drivers support this function. You can query a layer to check if it supports it with the OLCDeleteField capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existing features of the backing file/database should be updated accordingly.

This function is the same as the C++ method **OGRLayer::DeleteField()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer.
<i>iField</i>	index of the field to delete.

## Returns

OGRERR\_NONE on success.

## Since

OGR 1.9.0

References OGRLayer::DeleteField(), OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

## 12.15.4.268 OGR\_L\_Erase()

```

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 or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

This function is the same as the C++ method **OGRLayer::Erase()** (p. ??).

## Parameters

<i>pLayerInput</i>	the input layer. Should not be NULL.
<i>pLayerMethod</i>	the method layer. Should not be NULL.
<i>pLayerResult</i>	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
<i>papszOptions</i>	NULL terminated list of options (may be NULL).
<i>pfnProgress</i>	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
<i>pProgressArg</i>	argument to be passed to pfnProgress. May be NULL.

## Returns

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

**Note**

The first geometry field is always used.

**Since**

OGR 1.10

References OGRLayer::Erase(), OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

**12.15.4.269 OGR\_L\_FindFieldIndex()**

```
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

References OGRLayer::FindFieldIndex(), OGRLayer::FromHandle(), and VALIDATE\_POINTER1.

**12.15.4.270 OGR\_L\_GetExtent()**

```
OGRERR OGR_L_GetExtent (
    OGRLayerH hLayer,
    OGREnvelope * psExtent,
    int bForce )
```

Fetch the extent of this layer.

Returns the extent (MBR) of the data in the layer. If bForce is FALSE, and it would be expensive to establish the extent then OGRERR\_FAILURE will be returned indicating that the extent isn't know. If bForce is TRUE then some implementations will actually scan the entire layer once to compute the MBR of all the features in the layer.

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call **OGR\_L\_GetExtent()** (p. ??) without setting a spatial filter.

Layers without any geometry may return OGRERR\_FAILURE just indicating that no meaningful extents could be collected.

Note that some implementations of this method may alter the read cursor of the layer.

This function is the same as the C++ method **OGRLayer::GetExtent()** (p. ??).



## Parameters

<i>hLayer</i>	handle to the layer from which to get extent.
<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	Flag indicating whether the extent should be computed even if it is expensive.

## Returns

OGRERR\_NONE on success, OGRERR\_FAILURE if extent not known.

References OGRLayer::FromHandle(), OGRLayer::GetExtent(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

## 12.15.4.271 OGR\_L\_GetExtentEx()

```
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

<i>hLayer</i>	handle to the layer from which to get extent.
<i>iGeomField</i>	the index of the geometry field on which to compute the extent.
<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	Flag indicating whether the extent should be computed even if it is expensive.

## Returns

OGRERR\_NONE on success, OGRERR\_FAILURE if extent not known.

References OGRLayer::FromHandle(), OGRLayer::GetExtent(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

## 12.15.4.272 OGR\_L\_GetFeature()

```
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

<i>hLayer</i>	handle to the layer that owned the feature.
<i>nFeatureId</i>	the feature id of the feature to read.

## Returns

an handle to a feature now owned by the caller, or NULL on failure.

References **OGRLayer::FromHandle()**, **OGRLayer::GetFeature()**, **OGRFeature::ToHandle()**, and **VALIDATE\_POINTER1**.

## 12.15.4.273 OGR\_L\_GetFeatureCount()

```
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

<i>hLayer</i>	handle to the layer that owned the features.
<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.

#### Returns

feature count, -1 if count not known.

References OGRLayer::FromHandle(), OGRLayer::GetFeatureCount(), and VALIDATE\_POINTER1.

#### 12.15.4.274 OGR\_L\_GetFIDColumn()

```
const char * OGR_L_GetFIDColumn (
    OGRLayerH hLayer )
```

This method returns the name of the underlying database column being used as the FID column, or "" if not supported.

This method is the same as the C++ method **OGRLayer::GetFIDColumn()** (p. ??)

#### Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

#### Returns

fid column name.

References OGRLayer::FromHandle(), OGRLayer::GetFIDColumn(), and VALIDATE\_POINTER1.

#### 12.15.4.275 OGR\_L\_GetGeometryColumn()

```
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

<i>hLayer</i>	handle to the layer
---------------	---------------------

#### Returns

geometry column name.

References `OGRLayer::FromHandle()`, `OGRLayer::GetGeometryColumn()`, and `VALIDATE_POINTER1`.

#### 12.15.4.276 OGR\_L\_GetGeomType()

```
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

<i>hLayer</i>	handle to the layer.
---------------	----------------------

#### Returns

the geometry type

#### Since

OGR 1.8.0

References `OGRLayer::FromHandle()`, `OGRLayer::GetGeomType()`, `OGR_GT_GetLinear()`, `OGR_GT_IsNonLinear()`, `OGRGetNonLinearGeometriesEnabledFlag()`, `VALIDATE_POINTER1`, and `wkbUnknown`.

## 12.15.4.277 OGR\_L\_GetLayerDefn()

```
OGRFeatureDefnH OGR_L_GetLayerDefn (
    OGRLayerH hLayer )
```

Fetch the schema information for this layer.

The returned handle to the **OGRFeatureDefn** (p. ??) is owned by the **OGRLayer** (p. ??), and should not be modified or freed by the application. It encapsulates the attribute schema of the features of the layer.

This function is the same as the C++ method **OGRLayer::GetLayerDefn()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to get the schema information.
---------------	--

## Returns

an handle to the feature definition.

References **OGRLayer::FromHandle()**, **OGRLayer::GetLayerDefn()**, **OGRFeatureDefn::ToHandle()**, and **VALIDATE\_POINTER1**.

## 12.15.4.278 OGR\_L\_GetName()

```
const char * OGR_L_GetName (
    OGRLayerH hLayer )
```

Return the layer name.

This returns the same content as **OGR\_FD\_GetName(OGR\_L\_GetLayerDefn(hLayer))**, but for a few drivers, calling **OGR\_L\_GetName()** (p. ??) directly can avoid lengthy layer definition initialization.

This function is the same as the C++ method **OGRLayer::GetName()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer.
---------------	----------------------

## Returns

the layer name (must not been freed)

## Since

OGR 1.8.0

References **OGRLayer::FromHandle()**, **OGRLayer::GetName()**, and **VALIDATE\_POINTER1**.

## 12.15.4.279 OGR\_L\_GetNextFeature()

```
OGRFeatureH OGR_L_GetNextFeature (
    OGRLayerH hLayer )
```

Fetch the next available feature from this layer.

The returned feature becomes the responsibility of the caller to delete with **OGR\_F\_Destroy()** (p. ??). It is critical that all features associated with an **OGRLayer** (p. ??) (more specifically an **OGRFeatureDefn** (p. ??)) be deleted before that layer/datasource is deleted.

Only features matching the current spatial filter (set with **SetSpatialFilter()**) will be returned.

This function implements sequential access to the features of a layer. The **OGR\_L\_ResetReading()** (p. ??) function can be used to start at the beginning again.

Features returned by **OGR\_GetNextFeature()** may or may not be affected by concurrent modifications depending on drivers. A guaranteed way of seeing 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

<i>hLayer</i>	handle to the layer from which feature are read.
---------------	--

## Returns

an handle to a feature, or NULL if no more features are available.

References **OGRLayer::FromHandle()**, **OGRLayer::GetNextFeature()**, **OGRFeature::ToHandle()**, and **VALIDATE\_POINTER1**.

## 12.15.4.280 OGR\_L\_GetSpatialFilter()

```
OGRGeometryH OGR_L_GetSpatialFilter (
    OGRLayerH hLayer )
```

This function returns the current spatial filter for this layer.

The returned pointer is to an internally owned object, and should not be altered or deleted by the caller.

This function is the same as the C++ method **OGRLayer::GetSpatialFilter()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to get the spatial filter from.
---------------	---

**Returns**

an handle to the spatial filter geometry.

References OGRLayer::FromHandle(), OGRLayer::GetSpatialFilter(), OGRGeometry::ToHandle(), and VALIDATE\_POINTER1.

**12.15.4.281 OGR\_L\_GetSpatialRef()**

```
OGRSpatialReferenceH OGR_L_GetSpatialRef (
    OGRLayerH hLayer )
```

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. ??) and should not be modified or freed by the application.

This function is the same as the C++ method **OGRLayer::GetSpatialRef()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer to get the spatial reference from.
---------------	--

**Returns**

spatial reference, or NULL if there isn't one.

References OGRLayer::FromHandle(), OGRLayer::GetSpatialRef(), OGRSpatialReference::ToHandle(), and VALIDATE\_POINTER1.

**12.15.4.282 OGR\_L\_GetStyleTable()**

```
OGRStyleTableH OGR_L_GetStyleTable (
    OGRLayerH )
```

Get style table

References OGRLayer::FromHandle(), OGRLayer::GetStyleTable(), and VALIDATE\_POINTER1.

## 12.15.4.283 OGR\_L\_Identity()

```
OGRErr OGR_L_Identity (
    OGRLayerH pLayerInput,
    OGRLayerH pLayerMethod,
    OGRLayerH pLayerResult,
    char ** ppszOptions,
    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 or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.
- USE\_PREPARED\_GEOMETRIES=YES/NO. Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- KEEP\_LOWER\_DIMENSION\_GEOMETRIES=YES/NO. Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

This function is the same as the C++ method **OGRLayer::Identity()** (p. ??).

## Parameters

<i>pLayerInput</i>	the input layer. Should not be NULL.
<i>pLayerMethod</i>	the method layer. Should not be NULL.
<i>pLayerResult</i>	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
<i>ppszOptions</i>	NULL terminated list of options (may be NULL).
<i>pfnProgress</i>	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
<i>pProgressArg</i>	argument to be passed to pfnProgress. May be NULL.



**Returns**

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

**Note**

The first geometry field is always used.

**Since**

OGR 1.10

References OGRLayer::FromHandle(), OGRLayer::Identity(), OGRERR\_INVALID\_HANDLE, and VALIDATE\_POINTER1.

**12.15.4.284 OGR\_L\_Intersection()**

```
OGRErr OGR_L_Intersection (
    OGRLayerH pLayerInput,
    OGRLayerH pLayerMethod,
    OGRLayerH pLayerResult,
    char ** ppszOptions,
    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 or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

- **USE\_PREPARED\_GEOMETRIES=YES/NO.** Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- **PRETEST\_CONTAINMENT=YES/NO.** Set to YES to pretest the containment of features of method layer within the features of this layer. This will speed up the method significantly in some cases. Requires that the prepared geometries are in effect.
- **KEEP\_LOWER\_DIMENSION\_GEOMETRIES=YES/NO.** Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

This function is the same as the C++ method **OGRLayer::Intersection()** (p. ??).

#### Parameters

<i>pLayerInput</i>	the input layer. Should not be NULL.
<i>pLayerMethod</i>	the method layer. Should not be NULL.
<i>pLayerResult</i>	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
<i>papszOptions</i>	NULL terminated list of options (may be NULL).
<i>pfnProgress</i>	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
<i>pProgressArg</i>	argument to be passed to pfnProgress. May be NULL.

#### Returns

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

#### Note

The first geometry field is always used.

#### Since

OGR 1.10

References [OGRLayer::FromHandle\(\)](#), [OGRLayer::Intersection\(\)](#), [OGRERR\\_INVALID\\_HANDLE](#), and [VALIDATE\\_POINTER1](#).

#### 12.15.4.285 OGR\_L\_ReorderField()

```
OGRErr OGR_L_ReorderField (
    OGRLayerH hLayer,
    int iOldFieldPos,
    int iNewFieldPos )
```

Reorder an existing field on a layer.

This function is a convenience wrapper of **OGR\_L\_ReorderFields()** (p. ??) dedicated to move a single field.

You must use this to reorder existing fields on a real layer. Internally the **OGRFeatureDefn** (p. ??) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. ??) used by a layer directly.

This function should not be called while there are feature objects in existence that were obtained or created with the previous layer definition.

The field definition that was at initial position `iOldFieldPos` will be moved at position `iNewFieldPos`, and elements between will be shuffled accordingly.

For example, let suppose the fields were "0","1","2","3","4" initially. `ReorderField(1, 3)` will reorder them as "0","2","3","1","4".

Not all drivers support this function. You can query a layer to check if it supports it with the `OLCReorderFields` capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existing features of the backing file/database should be updated accordingly.

This function is the same as the C++ method **OGRLayer::ReorderField()** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer.
<i>iOldFieldPos</i>	previous position of the field to move. Must be in the range <code>[0,GetFieldCount()-1]</code> .
<i>iNewFieldPos</i>	new position of the field to move. Must be in the range <code>[0,GetFieldCount()-1]</code> .

#### Returns

`OGRERR_NONE` on success.

#### Since

OGR 1.9.0

References `OGRLayer::FromHandle()`, `OGRERR_INVALID_HANDLE`, `OGRLayer::ReorderField()`, and `VALIDATE_POINTER1`.

#### 12.15.4.286 OGR\_L\_ReorderFields()

```
OGRERR OGR_L_ReorderFields (
    OGRLayerH hLayer,
    int * panMap )
```

Reorder all the fields of a layer.

You must use this to reorder existing fields on a real layer. Internally the **OGRFeatureDefn** (p. ??) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. ??) used by a layer directly.

This function should not be called while there are feature objects in existence that were obtained or created with the previous layer definition.

panMap is such that, for each field definition at position *i* after reordering, its position before reordering was panMap[i].

For example, let suppose the fields were "0","1","2","3","4" initially. ReorderFields([0,2,3,1,4]) will reorder them as "0","2","3","1","4".

Not all drivers support this function. You can query a layer to check if it supports it with the OLCReorderFields capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existing features of the backing file/database should be updated accordingly.

This function is the same as the C++ method **OGRLayer::ReorderFields()** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer.
<i>panMap</i>	an array of GetLayerDefn()-> <b>OGRFeatureDefn::GetFieldCount()</b> (p. ??) elements which is a permutation of [0, GetLayerDefn()-> <b>OGRFeatureDefn::GetFieldCount()</b> (p. ??)-1].

#### Returns

OGRERR\_NONE on success.

#### Since

OGR 1.9.0

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::ReorderFields(), and VALIDATE\_POINTER1.

#### 12.15.4.287 OGR\_L\_ResetReading()

```
void OGR_L_ResetReading (
    OGRLayerH hLayer )
```

Reset feature reading to start on the first feature.

This affects GetNextFeature().

This function is the same as the C++ method **OGRLayer::ResetReading()** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer on which features are read.
---------------	---

References OGRLayer::FromHandle(), OGRLayer::ResetReading(), and VALIDATE\_POINTER0.

## 12.15.4.288 OGR\_L\_RollbackTransaction()

```
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()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

## Returns

OGRERR\_NONE on success.

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::RollbackTransaction(), and [VALIDATE\\_POINTER1](#).

## 12.15.4.289 OGR\_L\_SetAttributeFilter()

```
OGRERR OGR_L_SetAttributeFilter (
    OGRLayerH hLayer,
    const char * pszQuery )
```

Set a new attribute query.

This function sets the attribute query string to be used when fetching features via the **OGR\_L\_GetNextFeature()** (p. ??) function. Only features for which the query evaluates as true will be returned.

The query string should be in the format of an SQL WHERE clause. For instance "population > 1000000 and population < 5000000" where population is an attribute in the layer. The query format is a restricted form of SQL WHERE clause as defined "eq\_format=restricted\_where" about half way through this document:

<http://ogdi.sourceforge.net/prop/6.2.CapabilitiesMetadata.html>

Note that installing a query string will generally result in resetting the current reading position (ala **OGR\_L\_ResetReading()** (p. ??)).

This function is the same as the C++ method **OGRLayer::SetAttributeFilter()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer on which attribute query will be executed.
<i>pszQuery</i>	query in restricted SQL WHERE format, or NULL to clear the current query.

**Returns**

OGRERR\_NONE if successfully installed, or an error code if the query expression is in error, or some other failure occurs.

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::SetAttributeFilter(), and VALIDATE\_POINTER1.

**12.15.4.290 OGR\_L\_SetFeature()**

```
OGRERR OGR_L_SetFeature (
    OGRLayerH hLayer,
    OGRFeatureH hFeat )
```

Rewrite an existing feature.

This function will write a feature to the layer, based on the feature id within the **OGRFeature** (p. ??).

Use OGR\_L\_TestCapability(OLCRandomWrite) to establish if this layer supports random access writing via **OGR\_L\_SetFeature()** (p. ??).

This function is the same as the C++ method **OGRLayer::SetFeature()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer to write the feature.
<i>hFeat</i>	the feature to write.

**Returns**

OGRERR\_NONE if the operation works, otherwise an appropriate error code (e.g OGRERR\_NON\_EXISTING\_FEATURE if the feature does not exist).

References OGRLayer::FromHandle(), OGRFeature::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::SetFeature(), and VALIDATE\_POINTER1.

**12.15.4.291 OGR\_L\_SetIgnoredFields()**

```
OGRERR OGR_L_SetIgnoredFields (
    OGRLayerH ,
    const char ** papszFields )
```

Set which fields can be omitted when retrieving features from the layer.

If the driver supports this functionality (testable using OLCIgnoreFields capability), it will not fetch the specified fields in subsequent calls to GetFeature() / GetNextFeature() and thus save some processing time and/or bandwidth.

Besides field names of the layers, the following special fields can be passed: "OGR\_GEOMETRY" to ignore geometry and "OGR\_STYLE" to ignore layer style.

By default, no fields are ignored.

This method is the same as the C++ method **OGRLayer::SetIgnoredFields()** (p. ??)

## Parameters

<i>papszFields</i>	an array of field names terminated by NULL item. If NULL is passed, the ignored list is cleared.
--------------------	--

## Returns

OGRERR\_NONE if all field names have been resolved (even if the driver does not support this method)

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::SetIgnoredFields(), and VALI←DATE\_POINTER1.

## 12.15.4.292 OGR\_L\_SetNextByIndex()

```
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

<i>hLayer</i>	handle to the layer
<i>nIndex</i>	the index indicating how many steps into the result set to seek.

## Returns

OGRERR\_NONE on success or an error code.

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::SetNextByIndex(), and VALI←DATE\_POINTER1.

## 12.15.4.293 OGR\_L\_SetSpatialFilter()

```
void OGR_L_SetSpatialFilter (
    OGRLayerH hLayer,
    OGRGeometryH hGeom )
```

Set a new spatial filter.

This function set the geometry to be used as a spatial filter when fetching features via the **OGR\_L\_GetNextFeature()** (p. ??) function. Only features that geometrically intersect the filter geometry will be returned.

Currently this test is may be inaccurately implemented, but it is guaranteed that all features whose 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.

Starting with GDAL 2.3, features with null or empty geometries will never be considered as matching a spatial filter.

This function makes an internal copy of the passed geometry. The passed geometry remains the responsibility of the caller, and may be safely destroyed.

For the time being the passed filter geometry should be in the same SRS as the layer (as returned by **OGR\_L\_GetSpatialRef()** (p. ??)). In the future this may be generalized.

This function is the same as the C++ method **OGRLayer::SetSpatialFilter** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer on which to set the spatial filter.
<i>hGeom</i>	handle to the geometry to use as a filtering region. NULL may be passed indicating that the current spatial filter should be cleared, but no new one instituted.

References **OGRLayer::FromHandle()**, **OGRGeometry::FromHandle()**, **OGRLayer::SetSpatialFilter()**, and **VALIDATE\_POINTER0**.

#### 12.15.4.294 OGR\_L\_SetSpatialFilterEx()

```
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\_GetNextFeature()** (p. ??) function. Only features that geometrically intersect the filter geometry will be returned.

Currently this test is may be inaccurately implemented, but it is guaranteed that all features who's envelope (as returned by **OGR\_G\_GetEnvelope()** (p. ??)) overlaps the envelope of the spatial filter will be returned. This can result in more shapes being returned that should strictly be the case.

This function makes an internal copy of the passed geometry. The passed geometry remains the responsibility of the caller, and may be safely destroyed.

For the time being the passed filter geometry should be in the same SRS as the geometry field definition it corresponds to (as returned by **GetLayerDefn()->OGRFeatureDefn::GetGeomFieldDefn(iGeomField)->GetSpatialRef()**). In the future this may be generalized.

Note that only the last spatial filter set is applied, even if several successive calls are done with different *iGeomField* values.

This function is the same as the C++ method **OGRLayer::SetSpatialFilter** (p. ??).



## Parameters

<i>hLayer</i>	handle to the layer on which to set the spatial filter.
<i>iGeomField</i>	index of the geometry field on which the spatial filter operates.
<i>hGeom</i>	handle to the geometry to use as a filtering region. NULL may be passed indicating that the current spatial filter should be cleared, but no new one instituted.

## Since

GDAL 1.11

References `OGRLayer::FromHandle()`, `OGRGeometry::FromHandle()`, `OGRLayer::SetSpatialFilter()`, and `VALIDATE_POINTER0`.

## 12.15.4.295 OGR\_L\_SetSpatialFilterRect()

```
void OGR_L_SetSpatialFilterRect (
    OGRLayerH hLayer,
    double dfMinX,
    double dfMinY,
    double dfMaxX,
    double dfMaxY )
```

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the `OGR_L_GetNextFeature()` (p. ??) method. Only features that geometrically intersect the given rectangle will be returned.

The x/y values should be in the same coordinate system as the layer as a whole (as returned by `OGRLayer::GetSpatialRef()` (p. ??)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to `OGRLayer::SetSpatialFilter()` (p. ??). It exists as a convenience.

The only way to clear a spatial filter set with this method is to call `OGRLayer::SetSpatialFilter(NULL)`.

This method is the same as the C++ method `OGRLayer::SetSpatialFilterRect()` (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer on which to set the spatial filter.
<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.
<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

References `OGRLayer::FromHandle()`, `OGRLayer::SetSpatialFilterRect()`, and `VALIDATE_POINTER0`.

**12.15.4.296 OGR\_L\_SetSpatialFilterRectEx()**

```
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**

<i>hLayer</i>	handle to the layer on which to set the spatial filter.
<i>iGeomField</i>	index of the geometry field on which the spatial filter operates.
<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.
<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

Since

GDAL 1.11

References OGRLayer::FromHandle(), OGRLayer::SetSpatialFilterRect(), and VALIDATE\_POINTER0.

**12.15.4.297 OGR\_L\_SetStyleTable()**

```
void OGR_L_SetStyleTable (
    OGRLayerH ,
    OGRStyleTableH )
```

Set style table

References OGRLayer::FromHandle(), OGRLayer::SetStyleTable(), and VALIDATE\_POINTER0.

## 12.15.4.298 OGR\_L\_SetStyleTableDirectly()

```
void OGR_L_SetStyleTableDirectly (
    OGRLayerH ,
    OGRStyleTableH )
```

Set style table (and take ownership)

References OGRLayer::FromHandle(), OGRLayer::SetStyleTableDirectly(), and VALIDATE\_POINTER0.

## 12.15.4.299 OGR\_L\_StartTransaction()

```
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()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

## Returns

OGRERR\_NONE on success.

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::StartTransaction(), and VALIDATE\_POINTER1.

## 12.15.4.300 OGR\_L\_SymDifference()

```
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 or a GEOS call failed.
- **PROMOTE\_TO\_MULTI=**YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- **INPUT\_PREFIX=**string. Set a prefix for the field names that will be created from the fields of the input layer.
- **METHOD\_PREFIX=**string. Set a prefix for the field names that will be created from the fields of the method layer.

This function is the same as the C++ method **OGRLayer::SymDifference()** (p. ??).

#### Parameters

<i>pLayerInput</i>	the input layer. Should not be NULL.
<i>pLayerMethod</i>	the method layer. Should not be NULL.
<i>pLayerResult</i>	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
<i>papszOptions</i>	NULL terminated list of options (may be NULL).
<i>pfnProgress</i>	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
<i>pProgressArg</i>	argument to be passed to pfnProgress. May be NULL.

#### Returns

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

#### Note

The first geometry field is always used.

#### Since

OGR 1.10

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::SymDifference(), and VALIDATE\_POINTER1.

## 12.15.4.301 OGR\_L\_SyncToDisk()

```
OGRErr OGR_L_SyncToDisk (
    OGRLayerH hLayer )
```

Flush pending changes to disk.

This call is intended to force the layer to flush any pending writes to disk, and leave the disk file in a consistent state. It would not normally have any effect on read-only datasources.

Some layers do not implement this method, and will still return OGRERR\_NONE. The default implementation just returns OGRERR\_NONE. An error is only returned if an error occurs while attempting to flush to disk.

In any event, you should always close any opened datasource with **OGR\_DS\_Destroy()** (p. ??) that will ensure all data is correctly flushed.

This method is the same as the C++ method **OGRLayer::SyncToDisk()** (p. ??)

## Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

## Returns

OGRERR\_NONE if no error occurs (even if nothing is done) or an error code.

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::SyncToDisk(), and VALIDATE\_POINTER1.

## 12.15.4.302 OGR\_L\_TestCapability()

```
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, i.e. without counting the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastGetExtent** / "FastGetExtent": TRUE if this layer can return its data extent (via **OGR\_L\_GetExtent()** (p. ??)) efficiently, i.e. without scanning all the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastSetNextByIndex** / "FastSetNextByIndex": TRUE if this layer can perform the **SetNextByIndex()** call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **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

<i>hLayer</i>	handle to the layer to get the capability from.
<i>pszCap</i>	the name of the capability to test.

#### Returns

TRUE if the layer has the requested capability, or FALSE otherwise. **OGRLayers** will return FALSE for any unrecognized capabilities.

References **OGRLayer::FromHandle()**, **OGRLayer::TestCapability()**, and **VALIDATE\_POINTER1**.

## 12.15.4.303 OGR\_L\_Union()

```

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, in the method layer, or in both. The features in the result layer have attributes from both input and method layers. For features which represent areas that are only in the input or in the method layer the respective attributes have undefined values. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in the input and method layers.

#### Note

If the schema of the result is set by user and contains fields that have the same name as a field in input and in method layer, then the attribute in the result feature will get the value from the feature of the method layer (even if it is undefined).

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP\_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.
- USE\_PREPARED\_GEOMETRIES=YES/NO. Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- KEEP\_LOWER\_DIMENSION\_GEOMETRIES=YES/NO. Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

This function is the same as the C++ method **OGRLayer::Union()** (p. ??).

#### Parameters

<i>pLayerInput</i>	the input layer. Should not be NULL.
<i>pLayerMethod</i>	the method layer. Should not be NULL.
<i>pLayerResult</i>	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
<i>papszOptions</i>	NULL terminated list of options (may be NULL).
<i>pfnProgress</i>	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
<i>pProgressArg</i>	argument to be passed to pfnProgress. May be NULL.

**Returns**

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

**Note**

The first geometry field is always used.

**Since**

OGR 1.10

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::Union(), and VALIDATE\_POI↔  
NTER1.

**12.15.4.304 OGR\_L\_Update()**

```
OGRErr OGR_L_Update (
    OGRLayerH pLayerInput,
    OGRLayerH pLayerMethod,
    OGRLayerH pLayerResult,
    char ** ppszOptions,
    GDALProgressFunc pfnProgress,
    void * pProgressArg )
```

Update this layer with features from the update layer.

The result layer contains features whose geometries represent areas that are either in the input layer or in the method layer. The features in the result layer have areas of the features of the method layer or those areas of the features of the input layer that are not covered by the method layer. The features of the result layer get their attributes from the input layer. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in the input layer.

**Note**

If the schema of the result is set by user and contains fields that have the same name as a field in the method layer, then the attribute in the result feature the originates from the method layer will get the value from the feature of the method layer.

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP\_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE\_TO\_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD\_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

This function is the same as the C++ method **OGRLayer::Update()** (p. ??).



## Parameters

<i>pLayerInput</i>	the input layer. Should not be NULL.
<i>pLayerMethod</i>	the method layer. Should not be NULL.
<i>pLayerResult</i>	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
<i>papszOptions</i>	NULL terminated list of options (may be NULL).
<i>pfnProgress</i>	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
<i>pProgressArg</i>	argument to be passed to pfnProgress. May be NULL.

## Returns

an error code if there was an error or the execution was interrupted, OGRERR\_NONE otherwise.

## Note

The first geometry field is always used.

## Since

OGR 1.10

References OGRLayer::FromHandle(), OGRERR\_INVALID\_HANDLE, OGRLayer::Update(), and VALIDATE\_POINTER1.

## 12.15.4.305 OGR\_RawField\_IsNull()

```
int OGR_RawField_IsNull (
    const OGRField * puField )
```

Returns whether a raw field is null.

Note: this function is rather low-level and should be rarely used in client code. Use instead **OGR\_F\_IsFieldNull()** (p. ??).

## Parameters

<i>puField</i>	pointer to raw field.
----------------	-----------------------

## Since

GDAL 2.2

References OGRNullMarker.

#### 12.15.4.306 OGR\_RawField\_IsUnset()

```
int OGR_RawField_IsUnset (
    const  OGRField * puField )
```

Returns whether a raw field is unset.

Note: this function is rather low-level and should be rarely used in client code. Use instead **OGR\_F\_IsFieldSet()** (p. ??).

##### Parameters

<i>puField</i>	pointer to raw field.
----------------	-----------------------

##### Since

GDAL 2.2

References OGRUnsetMarker.

Referenced by OGRFeature::IsFieldSet().

#### 12.15.4.307 OGR\_RawField\_SetNull()

```
void OGR_RawField_SetNull (
    OGRField * puField )
```

Mark a raw field as null.

This should be called on a un-initialized field. In particular this will not free any memory dynamically allocated.

Note: this function is rather low-level and should be rarely used in client code. Use instead **OGR\_F\_SetFieldNull()** (p. ??).

##### Parameters

<i>puField</i>	pointer to raw field.
----------------	-----------------------

##### Since

GDAL 2.2

References OGRNullMarker.

Referenced by OGRFeature::SetFieldNull().

#### 12.15.4.308 OGR\_RawField\_SetUnset()

```
void OGR_RawField_SetUnset (
    OGRField * puField )
```

Mark a raw field as unset.

This should be called on a un-initialized field. In particular this will not free any memory dynamically allocated.

Note: this function is rather low-level and should be rarely used in client code. Use instead **OGR\_F\_UnsetField()** (p. ??).

##### Parameters

<i>puField</i>	pointer to raw field.
----------------	-----------------------

##### Since

GDAL 2.2

References OGRUnsetMarker.

Referenced by OGRFeature::OGRFeature(), OGRFeature::SetField(), and OGRFeature::UnsetField().

#### 12.15.4.309 OGR\_SM\_AddPart()

```
int OGR_SM_AddPart (
    OGRStyleMgrH hSM,
    OGRStyleToolH hST )
```

Add a part (style tool) to the current style.

This function is the same as the C++ method **OGRStyleMgr::AddPart()** (p. ??).

##### Parameters

<i>hSM</i>	handle to the style manager.
<i>hST</i>	the style tool defining the part to add.

##### Returns

TRUE on success, FALSE on errors.

References VALIDATE\_POINTER1.

#### 12.15.4.310 OGR\_SM\_AddStyle()

```
int OGR_SM_AddStyle (
    OGRStyleMgrH hSM,
    const char * pszStyleName,
    const char * pszStyleString )
```

Add a style to the current style table.

This function is the same as the C++ method **OGRStyleMgr::AddStyle()** (p. ??).

##### Parameters

<i>hSM</i>	handle to the style manager.
<i>pszStyleName</i>	the name of the style to add.
<i>pszStyleString</i>	the style string to use, or NULL to use the style stored in the manager.

##### Returns

TRUE on success, FALSE on errors.

References VALIDATE\_POINTER1.

#### 12.15.4.311 OGR\_SM\_Create()

```
OGRStyleMgrH OGR_SM_Create (
    OGRStyleTableH hStyleTable )
```

**OGRStyleMgr** (p. ??) factory.

This function is the same as the C++ method **OGRStyleMgr::OGRStyleMgr()** (p. ??).

##### Parameters

<i>hStyleTable</i>	pointer to <b>OGRStyleTable</b> (p. ??) or NULL if not working with a style table.
--------------------	--

##### Returns

an handle to the new style manager object.

#### 12.15.4.312 OGR\_SM\_Destroy()

```
void OGR_SM_Destroy (
    OGRStyleMgrH hSM )
```

Destroy Style Manager.

This function is the same as the C++ method **OGRStyleMgr::~OGRStyleMgr()** (p. ??).

## Parameters

<i>hSM</i>	handle to the style manager to destroy.
------------	---

## 12.15.4.313 OGR\_SM\_GetPart()

```
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 instantiates a new object that should be freed with **OGR\_ST\_Destroy()** (p. ??).

## Parameters

<i>hSM</i>	handle to the style manager.
<i>nPartId</i>	the part number (0-based index).
<i>pszStyleString</i>	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

## Returns

OGRStyleToolH of the requested part (style tools) or NULL on error.

References VALIDATE\_POINTER1.

## 12.15.4.314 OGR\_SM\_GetPartCount()

```
int OGR_SM_GetPartCount (
    OGRStyleMgrH hSM,
    const char * pszStyleString )
```

Get the number of parts in a style.

This function is the same as the C++ method **OGRStyleMgr::GetPartCount()** (p. ??).

## Parameters

<i>hSM</i>	handle to the style manager.
<i>pszStyleString</i>	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

**Returns**

the number of parts (style tools) in the style.

References `VALIDATE_POINTER1`.

**12.15.4.315 OGR\_SM\_InitFromFeature()**

```
const char* OGR_SM_InitFromFeature (
    OGRStyleMgrH hSM,
    OGRFeatureH hFeat )
```

Initialize style manager from the style string of a feature.

This function is the same as the C++ method `OGRStyleMgr::InitFromFeature()` (p. ??).

**Parameters**

<i>hSM</i>	handle to the style manager.
<i>hFeat</i>	handle to the new feature from which to read the style.

**Returns**

a reference to the style string read from the feature, or NULL in case of error.

References `VALIDATE_POINTER1`.

**12.15.4.316 OGR\_SM\_InitStyleString()**

```
int OGR_SM_InitStyleString (
    OGRStyleMgrH hSM,
    const char * pszStyleString )
```

Initialize style manager from the style string.

This function is the same as the C++ method `OGRStyleMgr::InitStyleString()` (p. ??).

**Parameters**

<i>hSM</i>	handle to the style manager.
<i>pszStyleString</i>	the style string to use (can be NULL).

**Returns**

TRUE on success, FALSE on errors.

References `VALIDATE_POINTER1`.

#### 12.15.4.317 `OGR_ST_Create()`

```
OGRStyleToolH OGR_ST_Create (
    OGRSTClassId eClassId )
```

**OGRStyleTool** (p. ??) factory.

This function is a constructor for **OGRStyleTool** (p. ??) derived classes.

##### Parameters

<i>e</i> ↔ <i>ClassId</i>	subclass of style tool to create. One of <code>OGRSTCPen</code> (1), <code>OGRSTCBrush</code> (2), <code>OGRSTCSymbol</code> (3) or <code>OGRSTCLabel</code> (4).
------------------------------	---

##### Returns

an handle to the new style tool object or `NULL` if the creation failed.

References `OGRSTCBrush`, `OGRSTCLabel`, `OGRSTCPen`, and `OGRSTCSymbol`.

#### 12.15.4.318 `OGR_ST_Destroy()`

```
void OGR_ST_Destroy (
    OGRStyleToolH hST )
```

Destroy Style Tool.

##### Parameters

<i>hST</i>	handle to the style tool to destroy.
------------	--------------------------------------

#### 12.15.4.319 `OGR_ST_GetParamDbl()`

```
double OGR_ST_GetParamDbl (
    OGRStyleToolH hST,
    int eParam,
    int * bValueIsNull )
```

Get Style Tool parameter value as a double.

Maps to the **OGRStyleTool** (p. ??) subclasses' `GetParamDbl()` methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValueIsNull</i>	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

## Returns

the parameter value as double and sets bValueIsNull.

References OGRSTCBrush, OGRSTCLabel, OGRSTCPen, OGRSTCSymbol, and VALIDATE\_POINTER1.

## 12.15.4.320 OGR\_ST\_GetParamNum()

```
int OGR_ST_GetParamNum (
    OGRStyleToolH hST,
    int eParam,
    int * bValueIsNull )
```

Get Style Tool parameter value as an integer.

Maps to the **OGRStyleTool** (p. ??) subclasses' GetParamNum() methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValueIsNull</i>	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

## Returns

the parameter value as integer and sets bValueIsNull.

References OGRSTCBrush, OGRSTCLabel, OGRSTCPen, OGRSTCSymbol, and VALIDATE\_POINTER1.

## 12.15.4.321 OGR\_ST\_GetParamStr()

```
const char* OGR_ST_GetParamStr (
    OGRStyleToolH hST,
```



```
int eParam,  
int * bValueIsNull )
```

Get Style Tool parameter value as string.

Maps to the **OGRStyleTool** (p. ??) subclasses' GetParamStr() methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValuesNull</i>	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

## Returns

the parameter value as string and sets bValuesNull.

References OGRSTCBrush, OGRSTCLabel, OGRSTCPen, OGRSTCSymbol, and VALIDATE\_POINTER1.

## 12.15.4.322 OGR\_ST\_GetRGBFromString()

```
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()** (p. ??).

## Parameters

<i>hST</i>	handle to the style tool.
<i>pszColor</i>	the color to parse
<i>pnRed</i>	pointer to an int in which the red value will be returned
<i>pnGreen</i>	pointer to an int in which the green value will be returned
<i>pnBlue</i>	pointer to an int in which the blue value will be returned
<i>pnAlpha</i>	pointer to an int in which the (optional) alpha value will be returned

## Returns

TRUE if the color could be successfully parsed, or FALSE in case of errors.

References VALIDATE\_POINTER1.

#### 12.15.4.323 OGR\_ST\_GetStyleString()

```
const char* OGR_ST_GetStyleString (
    OGRStyleToolH hST )
```

Get the style string for this Style Tool.

Maps to the **OGRStyleTool** (p. ??) subclasses' GetStyleString() methods.

##### Parameters

<i>hST</i>	handle to the style tool.
------------	---------------------------

##### Returns

the style string for this style tool or "" if the hST is invalid.

References OGRSTCBrush, OGRSTCLabel, OGRSTCPen, OGRSTCSymbol, and VALIDATE\_POINTER1.

#### 12.15.4.324 OGR\_ST\_GetType()

```
OGRSTClassId OGR_ST_GetType (
    OGRStyleToolH hST )
```

Determine type of Style Tool.

##### Parameters

<i>hST</i>	handle to the style tool.
------------	---------------------------

##### Returns

the style tool type, one of OGRSTCPen (1), OGRSTCBrush (2), OGRSTCSymbol (3) or OGRSTCLabel (4). Returns OGRSTCNone (0) if the OGRStyleToolH is invalid.

References OGRSTCNone, and VALIDATE\_POINTER1.

#### 12.15.4.325 OGR\_ST\_GetUnit()

```
OGRSTUnitId OGR_ST_GetUnit (
    OGRStyleToolH hST )
```

Get Style Tool units.

## Parameters

<i>hST</i>	handle to the style tool.
------------	---------------------------

## Returns

the style tool units.

References OGRSTUGround, and VALIDATE\_POINTER1.

## 12.15.4.326 OGR\_ST\_SetParamDbl()

```
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

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>dfValue</i>	the new parameter value

References OGRSTCBrush, OGRSTCLabel, OGRSTCPen, OGRSTCSymbol, and VALIDATE\_POINTER0.

## 12.15.4.327 OGR\_ST\_SetParamNum()

```
void OGR_ST_SetParamNum (
    OGRStyleToolH hST,
    int eParam,
    int nValue )
```

Set Style Tool parameter value from an integer.

Maps to the **OGRStyleTool** (p. ??) subclasses' SetParamNum() methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>nValue</i>	the new parameter value

References OGRSTCBrush, OGRSTCLabel, OGRSTCPen, OGRSTCSymbol, and VALIDATE\_POINTER0.

#### 12.15.4.328 OGR\_ST\_SetParamStr()

```
void OGR_ST_SetParamStr (
    OGRStyleToolH hST,
    int eParam,
    const char * pszValue )
```

Set Style Tool parameter value from a string.

Maps to the **OGRStyleTool** (p. ??) subclasses' SetParamStr() methods.

##### Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>pszValue</i>	the new parameter value

References OGRSTCBrush, OGRSTCLabel, OGRSTCPen, OGRSTCSymbol, and VALIDATE\_POINTER0.

#### 12.15.4.329 OGR\_ST\_SetUnit()

```
void OGR_ST_SetUnit (
    OGRStyleToolH hST,
    OGRSTUnitId eUnit,
    double dfGroundPaperScale )
```

Set Style Tool units.

This function is the same as **OGRStyleTool::SetUnit()** (p. ??)

##### Parameters

<i>hST</i>	handle to the style tool.
<i>eUnit</i>	the new unit.
<i>dfGroundPaperScale</i>	ground to paper scale factor.

References VALIDATE\_POINTER0.

#### 12.15.4.330 OGR\_STBL\_AddStyle()

```
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

<i>hStyleTable</i>	handle to the style table.
<i>pszName</i>	the name the style to add.
<i>pszStyleString</i>	the style string to add.

##### Returns

TRUE on success, FALSE on error

References VALIDATE\_POINTER1.

#### 12.15.4.331 OGR\_STBL\_Create()

```
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.

#### 12.15.4.332 OGR\_STBL\_Destroy()

```
void OGR_STBL_Destroy (
    OGRStyleTableH hSTBL )
```

Destroy Style Table.

##### Parameters

<i>hSTBL</i>	handle to the style table to destroy.
--------------	---------------------------------------

#### 12.15.4.333 OGR\_STBL\_Find()

```
const char* OGR_STBL_Find (
    OGRStyleTableH hStyleTable,
    const char * pszName )
```

Get a style string by name.

This function is the same as the C++ method **OGRStyleTable::Find()** (p. ??).

##### Parameters

<i>hStyleTable</i>	handle to the style table.
<i>pszName</i>	the name of the style string to find.

##### Returns

the style string matching the name or NULL if not found or error.

References VALIDATE\_POINTER1.

#### 12.15.4.334 OGR\_STBL\_GetLastStyleName()

```
const char* OGR_STBL_GetLastStyleName (
    OGRStyleTableH hStyleTable )
```

Get the style name of the last style string fetched with OGR\_STBL\_GetNextStyle.

This function is the same as the C++ method **OGRStyleTable::GetStyleName()** (p. ??).

##### Parameters

<i>hStyleTable</i>	handle to the style table.
--------------------	----------------------------

##### Returns

the Name of the last style string or NULL on error.

References VALIDATE\_POINTER1.

#### 12.15.4.335 OGR\_STBL\_GetNextStyle()

```
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()** (p. ??).

#### Parameters

<i>hStyleTable</i>	handle to the style table.
--------------------	----------------------------

#### Returns

the next style string or NULL on error.

References `VALIDATE_POINTER1`.

#### 12.15.4.336 OGR\_STBL\_LoadStyleTable()

```
int OGR_STBL_LoadStyleTable (
    OGRStyleTableH hStyleTable,
    const char * pszFilename )
```

Load a style table from a file.

This function is the same as the C++ method **OGRStyleTable::LoadStyleTable()** (p. ??).

#### Parameters

<i>hStyleTable</i>	handle to the style table.
<i>pszFilename</i>	the name of the file to load from.

#### Returns

TRUE on success, FALSE on error

References `VALIDATE_POINTER1`.

#### 12.15.4.337 OGR\_STBL\_ResetStyleStringReading()

```
void OGR_STBL_ResetStyleStringReading (
    OGRStyleTableH hStyleTable )
```

Reset the next style pointer to 0.

This function is the same as the C++ method **OGRStyleTable::ResetStyleStringReading()** (p. ??).



## Parameters

<i>hStyleTable</i>	handle to the style table.
--------------------	----------------------------

References `VALIDATE_POINTER0`.

12.15.4.338 `OGR_STBL_SaveStyleTable()`

```
int OGR_STBL_SaveStyleTable (
    OGRStyleTableH hStyleTable,
    const char * pszFilename )
```

Save a style table to a file.

This function is the same as the C++ method `OGRStyleTable::SaveStyleTable()` (p. ??).

## Parameters

<i>hStyleTable</i>	handle to the style table.
<i>pszFilename</i>	the name of the file to save to.

## Returns

TRUE on success, FALSE on error

References `VALIDATE_POINTER1`.

12.15.4.339 `OGRCleanupAll()`

```
void OGRCleanupAll (
    void )
```

Clean-up all drivers (including raster ones starting with GDAL 2.0. See `GDALDestroyDriverManager()`)

12.15.4.340 `OGRGetDriver()`

```
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**

<i>iDriver</i>	the driver index, from 0 to GetDriverCount()-1.
----------------	---

**Returns**

handle to the driver, or NULL if iDriver is out of range.

**12.15.4.341 OGRGetDriverByName()**

```
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**

<i>pszName</i>	the driver name
----------------	-----------------

**Returns**

the driver, or NULL if no driver with that name is found

**12.15.4.342 OGRGetDriverCount()**

```
int OGRGetDriverCount (
    void )
```

Fetch the number of registered drivers.

**Deprecated** Use GDALGetDriverCount() in GDAL 2.0

**Returns**

the drivers count.

## 12.15.4.343 OGRGetNonLinearGeometriesEnabledFlag()

```
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\_G↔Fid\_GetType()**, and **OGR\_L\_GetGeomType()**.

## 12.15.4.344 OGROpen()

```
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 **VSIInstallZipFile↔Handler()** (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

<i>pszName</i>	the name of the file, or data source to open.
<i>bUpdate</i>	FALSE for read-only access (the default) or TRUE for read-write access.
<i>pahDriverList</i>	if non-NULL, this argument will be updated with a pointer to the driver which was used to open the data source.

**Returns**

NULL on error or if the pass name is not supported by this driver, otherwise an handle to 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 );
```

**12.15.4.345 OGROpenShared()**

```
OGRDataSourceH OGROpenShared (
    const char * pszName,
    int bUpdate,
    OGRSFDriverH * pahDriverList )
```

Open a file / data source with one of the registered drivers if not already opened, or increment reference count of already opened data source previously opened with **OGROpenShared()** (p. ??)

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.

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**

<i>pszName</i>	the name of the file, or data source to open.
<i>bUpdate</i>	FALSE for read-only access (the default) or TRUE for read-write access.
<i>pahDriverList</i>	if non-NULL, this argument will be updated with a pointer to the driver which was used to open the data source.

### Returns

NULL on error or if the pass name is not supported by this driver, otherwise an handle to a GDALDataset.  
This GDALDataset should be closed by deleting the object when it is no longer needed.

### Example:

```
OGRDataSourceH  hDS;  
OGRSFDriverH    *pahDriver;  
  
hDS = OGROpenShared( "polygon.shp", 0, pahDriver );  
if( hDS == NULL )  
{  
    return;  
}  
  
... use the data source ...  
  
OGRReleaseDataSource( hDS );
```

#### 12.15.4.346 OGRRegisterAll()

```
int OGRRegisterAll (   
    void )
```

Register all drivers.

**Deprecated** Use GDALAllRegister() in GDAL 2.0

#### 12.15.4.347 OGRReleaseDataSource()

```
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

<i>hDS</i>	handle to the data source to release
------------	--------------------------------------

## Returns

OGRERR\_NONE on success or an error code.

Referenced by OGRGeocodeDestroySession().

## 12.15.4.348 OGRSetNonLinearGeometriesEnabledFlag()

```
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\_GFId\_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

<i>bFlag</i>	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

## 12.16 ogr\_core.h File Reference

```
#include "cpl_port.h"
#include "gdal_version.h"
```

## Classes

- union **OGRField**

## Macros

- #define **OGRERR\_NONE** 0
- #define **OGRERR\_NOT\_ENOUGH\_DATA** 1
- #define **OGRERR\_NOT\_ENOUGH\_MEMORY** 2
- #define **OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE** 3
- #define **OGRERR\_UNSUPPORTED\_OPERATION** 4
- #define **OGRERR\_CORRUPT\_DATA** 5
- #define **OGRERR\_FAILURE** 6
- #define **OGRERR\_UNSUPPORTED\_SRS** 7
- #define **OGRERR\_INVALID\_HANDLE** 8
- #define **OGRERR\_NON\_EXISTING\_FEATURE** 9
- #define **wkb25DBit** 0x80000000
- #define **wkbFlatten(x)** **OGR\_GT\_Flatten**(static\_cast< **OGRwkbGeometryType**>(x))
- #define **wkbHasZ(x)** (**OGR\_GT\_HasZ**(x) != 0)
- #define **wkbSetZ(x)** **OGR\_GT\_SetZ**(x)
- #define **wkbHasM(x)** (**OGR\_GT\_HasM**(x) != 0)
- #define **wkbSetM(x)** **OGR\_GT\_SetM**(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\_PRECISION\_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\_ALLOW\_DIFFERENT\_GEOM\_DIM** 0x00000010
- #define **OGR\_F\_VAL\_ALL** (0x7FFFFFFF & ~**OGR\_F\_VAL\_ALLOW\_DIFFERENT\_GEOM\_DIM**)
- #define **OGRNullFID** -1
- #define **OGRUnsetMarker** -21121
- #define **OGRNullMarker** -21122
- #define **OLCRandomRead** "RandomRead"
- #define **OLCSequentialWrite** "SequentialWrite"
- #define **OLCRandomWrite** "RandomWrite"
- #define **OLCFastSpatialFilter** "FastSpatialFilter"
- #define **OLCFastFeatureCount** "FastFeatureCount"
- #define **OLCFastGetExtent** "FastGetExtent"
- #define **OLCCreateField** "CreateField"
- #define **OLCDeleteField** "DeleteField"
- #define **OLCReorderFields** "ReorderFields"
- #define **OLCAlterFieldDefn** "AlterFieldDefn"
- #define **OLCTransactions** "Transactions"
- #define **OLCDeleteFeature** "DeleteFeature"
- #define **OLCFastSetNextByIndex** "FastSetNextByIndex"
- #define **OLCStringsAsUTF8** "StringsAsUTF8"
- #define **OLCIgnoreFields** "IgnoreFields"
- #define **OLCCreateGeomField** "CreateGeomField"

- `#define OLCCurveGeometries "CurveGeometries"`
- `#define OLCMeasuredGeometries "MeasuredGeometries"`
- `#define ODsCCreateLayer "CreateLayer"`
- `#define ODsCDeleteLayer "DeleteLayer"`
- `#define ODsCCreateGeomFieldAfterCreateLayer "CreateGeomFieldAfterCreateLayer"`
- `#define ODsCCurveGeometries "CurveGeometries"`
- `#define ODsCTransactions "Transactions"`
- `#define ODsCEmulatedTransactions "EmulatedTransactions"`
- `#define ODsCMeasuredGeometries "MeasuredGeometries"`
- `#define ODsCRandomLayerRead "RandomLayerRead"`
- `#define ODsCRandomLayerWrite "RandomLayerWrite "`
- `#define ODrCCreateDataSource "CreateDataSource"`
- `#define ODrCDeleteDataSource "DeleteDataSource"`
- `#define OLMD_FID64 "OLMD_FID64"`

## Typedefs

- `typedef int OGRErr`
- `typedef int OGRBoolean`
- `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, wkbCurve = 13, wkbSurface = 14, wkbPolyhedralSurface = 15,  
wkbTIN = 16, wkbTriangle = 17, wkbNone = 100, wkbLinearRing = 101,  
wkbCircularStringZ = 1008, wkbCompoundCurveZ = 1009, wkbCurvePolygonZ = 1010, wkbMultiCurveZ = 1011,  
wkbMultiSurfaceZ = 1012, wkbCurveZ = 1013, wkbSurfaceZ = 1014, wkbPolyhedralSurfaceZ = 1015,  
wkbTINZ = 1016, wkbTriangleZ = 1017, wkbPointM = 2001, wkbLineStringM = 2002,  
wkbPolygonM = 2003, wkbMultiPointM = 2004, wkbMultiLineStringM = 2005, wkbMultiPolygonM = 2006,  
wkbGeometryCollectionM = 2007, wkbCircularStringM = 2008, wkbCompoundCurveM = 2009, wkbCurvePolygonM = 2010,  
wkbMultiCurveM = 2011, wkbMultiSurfaceM = 2012, wkbCurveM = 2013, wkbSurfaceM = 2014,  
wkbPolyhedralSurfaceM = 2015, wkbTINM = 2016, wkbTriangleM = 2017, wkbPointZM = 3001,  
wkbLineStringZM = 3002, wkbPolygonZM = 3003, wkbMultiPointZM = 3004, wkbMultiLineStringZM = 3005,  
wkbMultiPolygonZM = 3006, wkbGeometryCollectionZM = 3007, wkbCircularStringZM = 3008, wkbCompoundCurveZM = 3009,  
wkbCurvePolygonZM = 3010, wkbMultiCurveZM = 3011, wkbMultiSurfaceZM = 3012, wkbCurveZM = 3013,  
wkbSurfaceZM = 3014, wkbPolyhedralSurfaceZM = 3015, wkbTINZM = 3016, wkbTriangleZM = 3017,  
wkbPoint25D = 0x80000001, wkbLineString25D = 0x80000002, wkbPolygon25D = 0x80000003, wkbMultiPoint25D = 0x80000004,  
wkbMultiLineString25D = 0x80000005, wkbMultiPolygon25D = 0x80000006, wkbGeometryCollection25D = 0x80000007 }`



- enum **OGRwkbVariant** { **wkbVariantOldOgc**, **wkbVariantIso**, **wkbVariantPostGIS1** }
- enum **OGRwkbByteOrder** { **wkbXDR** = 0, **wkbNDR** = 1 }
- 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** {  
**OGRSTCNone** = 0, **OGRSTCPen** = 1, **OGRSTCBrush** = 2, **OGRSTCSymbol** = 3,  
**OGRSTCLabel** = 4, **OGRSTCVector** = 5 }
- enum **ogr\_style\_tool\_units\_id** {  
**OGRSTUPixel** = 0, **OGRSTUPixel** = 1, **OGRSTUPoints** = 2, **OGRSTUMM** = 3,  
**OGRSTUCM** = 4, **OGRSTUInches** = 5 }
- enum **ogr\_style\_tool\_param\_pen\_id** {  
**OGRSTPenColor** = 0, **OGRSTPenWidth** = 1, **OGRSTPenPattern** = 2, **OGRSTPenId** = 3,  
**OGRSTPenPerOffset** = 4, **OGRSTPenCap** = 5, **OGRSTPenJoin** = 6, **OGRSTPenPriority** = 7 }
- enum **ogr\_style\_tool\_param\_brush\_id** {  
**OGRSTBrushFColor** = 0, **OGRSTBrushBColor** = 1, **OGRSTBrushId** = 2, **OGRSTBrushAngle** = 3,  
**OGRSTBrushSize** = 4, **OGRSTBrushDx** = 5, **OGRSTBrushDy** = 6, **OGRSTBrushPriority** = 7 }
- enum **ogr\_style\_tool\_param\_symbol\_id** {  
**OGRSTSymbolId** = 0, **OGRSTSymbolAngle** = 1, **OGRSTSymbolColor** = 2, **OGRSTSymbolSize** = 3,  
**OGRSTSymbolDx** = 4, **OGRSTSymbolDy** = 5, **OGRSTSymbolStep** = 6, **OGRSTSymbolPerp** = 7,  
**OGRSTSymbolOffset** = 8, **OGRSTSymbolPriority** = 9, **OGRSTSymbolFontName** = 10, **OGRST↵**  
**SymbolIColor** = 11 }
- enum **ogr\_style\_tool\_param\_label\_id** {  
**OGRSTLabelFontName** = 0, **OGRSTLabelSize** = 1, **OGRSTLabelTextString** = 2, **OGRSTLabelAngle** =  
3,  
**OGRSTLabelFColor** = 4, **OGRSTLabelBColor** = 5, **OGRSTLabelPlacement** = 6, **OGRSTLabelAnchor**  
= 7,  
**OGRSTLabelDx** = 8, **OGRSTLabelDy** = 9, **OGRSTLabelPerp** = 10, **OGRSTLabelBold** = 11,  
**OGRSTLabelItalic** = 12, **OGRSTLabelUnderline** = 13, **OGRSTLabelPriority** = 14, **OGRSTLabel↵**  
**Strikeout** = 15,  
**OGRSTLabelStretch** = 16, **OGRSTLabelAdjHor** = 17, **OGRSTLabelAdjVert** = 18, **OGRSTLabelHColor**  
= 19,  
**OGRSTLabelIColor** = 20 }

## 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, **OGRwkb↵**  
**GeometryType** eExtra)  
*Find common geometry type.*
- **OGRwkbGeometryType** **OGRMergeGeometryTypesEx** ( **OGRwkbGeometryType** eMain, **OGRwkb↵**  
**GeometryType** 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.*
- **OGRwkbGeometryType** **OGR\_GT\_SetM** ( **OGRwkbGeometryType** eType)  
*Returns the measured geometry type corresponding to the passed geometry type.*

- **OGRwkbGeometryType OGR\_GT\_SetModifier** ( **OGRwkbGeometryType** eType, int bSetZ, int bSetM)  
*Returns a XY, XYZ, XYM or XYZM geometry type depending on parameter.*
- int **OGR\_GT\_HasZ** ( **OGRwkbGeometryType** eType)  
*Return if the geometry type is a 3D geometry type.*
- int **OGR\_GT\_HasM** ( **OGRwkbGeometryType** eType)  
*Return if the geometry type is a measured 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 **OGR\_GET\_MS** (float fSec)

### 12.16.1 Detailed Description

Core portability services for cross-platform OGR code.

### 12.16.2 Macro Definition Documentation

#### 12.16.2.1 ALTER\_ALL\_FLAG

```
#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. ??).

#### 12.16.2.2 ALTER\_DEFAULT\_FLAG

```
#define ALTER_DEFAULT_FLAG 0x10
```

Alter field DEFAULT value. Used by **OGR\_L\_AlterFieldDefn()** (p. ??).

Since

GDAL 2.0

### 12.16.2.3 ALTER\_NAME\_FLAG

```
#define ALTER_NAME_FLAG 0x1
```

Alter field name. Used by **OGR\_L\_AlterFieldDefn()** (p. ??).

### 12.16.2.4 ALTER\_NULLABLE\_FLAG

```
#define ALTER_NULLABLE_FLAG 0x8
```

Alter field NOT NULL constraint. Used by **OGR\_L\_AlterFieldDefn()** (p. ??).

Since

GDAL 2.0

### 12.16.2.5 ALTER\_TYPE\_FLAG

```
#define ALTER_TYPE_FLAG 0x2
```

Alter field type. Used by **OGR\_L\_AlterFieldDefn()** (p. ??).

### 12.16.2.6 ALTER\_WIDTH\_PRECISION\_FLAG

```
#define ALTER_WIDTH_PRECISION_FLAG 0x4
```

Alter field width and precision. Used by **OGR\_L\_AlterFieldDefn()** (p. ??).

### 12.16.2.7 ODrCCreateDataSource

```
#define ODrCCreateDataSource "CreateDataSource"
```

Driver capability for datasource creation

### 12.16.2.8 ODrCDeleteDataSource

```
#define ODrCDeleteDataSource "DeleteDataSource"
```

Driver capability for datasource deletion

### 12.16.2.9 ODsCCreateGeomFieldAfterCreateLayer

```
#define ODsCCreateGeomFieldAfterCreateLayer "CreateGeomFieldAfterCreateLayer"
```

Dataset capability for geometry field creation support

**12.16.2.10 ODSCreateLayer**

```
#define ODSCreateLayer "CreateLayer"
```

Dataset capability for layer creation

**12.16.2.11 ODSCurveGeometries**

```
#define ODSCurveGeometries "CurveGeometries"
```

Dataset capability for curve geometries support

**12.16.2.12 ODSDeleteLayer**

```
#define ODSDeleteLayer "DeleteLayer"
```

Dataset capability for layer deletion

**12.16.2.13 ODSEmulatedTransactions**

```
#define ODSEmulatedTransactions "EmulatedTransactions"
```

Dataset capability for emulated dataset transactions

**12.16.2.14 ODSMeasuredGeometries**

```
#define ODSMeasuredGeometries "MeasuredGeometries"
```

Dataset capability for measured geometries support

**12.16.2.15 ODSRandomLayerRead**

```
#define ODSRandomLayerRead "RandomLayerRead"
```

Dataset capability for GetNextFeature() returning features from random layers

**12.16.2.16 ODSRandomLayerWrite**

```
#define ODSRandomLayerWrite "RandomLayerWrite "
```

Dataset capability for supporting CreateFeature on layer in random order

**12.16.2.17 ODSCTransactions**

```
#define ODSCTransactions "Transactions"
```

Dataset capability for dataset transctions

### 12.16.2.18 OGR\_F\_VAL\_ALL

```
#define OGR_F_VAL_ALL (0x7FFFFFFF & ~OGR_F_VAL_ALLOW_DIFFERENT_GEOM_DIM)
```

Enable all validation tests (except OGR\_F\_VAL\_ALLOW\_DIFFERENT\_GEOM\_DIM) Used by **OGR\_F\_Validate()** (p. ??).

Since

GDAL 2.0

### 12.16.2.19 OGR\_F\_VAL\_ALLOW\_DIFFERENT\_GEOM\_DIM

```
#define OGR_F_VAL_ALLOW_DIFFERENT_GEOM_DIM 0x00000010
```

Allow geometry fields to have a different coordinate dimension than their geometry column type. This flag only makes sense if OGR\_F\_VAL\_GEOM\_TYPE is set too. Used by **OGR\_F\_Validate()** (p. ??).

Since

GDAL 2.1

### 12.16.2.20 OGR\_F\_VAL\_ALLOW\_NULL\_WHEN\_DEFAULT

```
#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

### 12.16.2.21 OGR\_F\_VAL\_GEOM\_TYPE

```
#define OGR_F_VAL_GEOM_TYPE 0x00000002
```

Validate that geometries respect geometry column type. Used by **OGR\_F\_Validate()** (p. ??).

Since

GDAL 2.0

#### 12.16.2.22 OGR\_F\_VAL\_NULL

```
#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().

#### 12.16.2.23 OGR\_F\_VAL\_WIDTH

```
#define OGR_F_VAL_WIDTH 0x00000004
```

Validate that (string) fields respect field width. Used by **OGR\_F\_Validate()** (p. ??).

Since

GDAL 2.0

#### 12.16.2.24 OGRERR\_CORRUPT\_DATA

```
#define OGRERR_CORRUPT_DATA 5
```

Corrupt data

Referenced by OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromUSGS(), OGRSimpleCurve::importFromWkb(), OGRCircularString::importFromWkb(), OGRTriangle::importFromWkb(), OGRPolyhedralSurface::importFromWkb(), OGRPoint::importFromWkt(), OGRSimpleCurve::importFromWkt(), OGRCircularString::importFromWkt(), OGRMultiSurface::importFromWkt(), OGRMultiPoint::importFromWkt(), OGRSpatialReference::importFromXML(), OGRSpatialReference::SetFromUserInput(), and OGRSpatialReference::Validate().

## 12.16.2.25 OGRERR\_FAILURE

```
#define OGRERR_FAILURE 6
```

Failure

Referenced by OGRGeometryCollection::addGeometry(), OGRPolyhedralSurface::addGeometry(), OGRTriangulatedSurface::addGeometry(), OGRGeometryCollection::addGeometryDirectly(), OGRPolyhedralSurface::addGeometryDirectly(), OGRTriangle::addRingDirectly(), OGRGeometry::Centroid(), OGRSpatialReference::CopyGeogCSFrom(), OGRFeatureDefn::DeleteFieldDefn(), OGRFeatureDefn::DeleteGeomFieldDefn(), OGRSpatialReference::exportToMICoordSys(), OGRCircularString::exportToWkb(), OGRCircularString::exportToWkt(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::ImportFromESRIStatePlaneWKT(), OGRSpatialReference::ImportFromESRIWisconsinWKT(), OGRSpatialReference::importFromMICoordSys(), OGRSpatialReference::importFromWkt(), OGR\_DS\_SyncToDisk(), OGR\_F\_SetFID(), OGR\_F\_SetFrom(), OGR\_F\_SetFromWithMap(), OGR\_F\_SetGeometry(), OGR\_F\_SetGeometryDirectly(), OGR\_F\_SetGeomField(), OGR\_F\_SetGeomFieldDirectly(), OGR\_G\_Centroid(), OGR\_G\_ExportToIsoWkb(), OGR\_G\_ExportToIsoWkt(), OGR\_G\_ExportToWkb(), OGR\_G\_ExportToWkt(), OGR\_G\_ImportFromWkb(), OGR\_G\_ImportFromWkt(), OGR\_G\_RemoveGeometry(), OGR\_G\_Transform(), OGR\_G\_TransformTo(), OSRAutoIdentifyEPSG(), OSRCopyGeogCSFrom(), OSREPSGTreatsAsLatLong(), OSREPSGTreatsAsNorthingEasting(), OSRExportToERM(), OSRExportToMICoordSys(), OSRExportToPanorama(), OSRExportToPCI(), OSRExportToUSGS(), OSRExportToXML(), OSRFixup(), OSRFixupOrdering(), OSRGetTOWGS84(), OSRImportFromDict(), OSRImportFromERM(), OSRImportFromESRI(), OSRImportFromMICoordSys(), OSRImportFromOzi(), OSRImportFromPanorama(), OSRImportFromPCI(), OSRImportFromProj4(), OSRImportFromUrl(), OSRImportFromUSGS(), OSRImportFromWkt(), OSRImportFromXML(), OSRMorphFromESRI(), OSRMorphToESRI(), OSRSetACEA(), OSRSetAE(), OSRSetAngularUnits(), OSRSetAuthority(), OSRSetAxes(), OSRSetBonne(), OSRSetCEA(), OSRSetCompoundCS(), OSRSetCS(), OSRSetEC(), OSRSetEckert(), OSRSetEckertIV(), OSRSetEckertVI(), OSRSetEquirectangular(), OSRSetEquirectangular2(), OSRSetGaussSchreiberTMercator(), OSRSetGeocCS(), OSRSetGeogCS(), OSRSetGEOS(), OSRSetGH(), OSRSetGnomonic(), OSRSetGS(), OSRSetHOM(), OSRSetHOM2PNO(), OSRSetHOMAC(), OSRSetIGH(), OSRSetIWMPolyconic(), OSRSetKrovak(), OSRSetLAEA(), OSRSetLCC(), OSRSetLCC1SP(), OSRSetLCCB(), OSRSetLinearUnits(), OSRSetLinearUnitsAndUpdateParameters(), OSRSetLocalCS(), OSRSetMC(), OSRSetMercator(), OSRSetMercator2SP(), OSRSetMollweide(), OSRSetNormProjParm(), OSRSetNZMG(), OSRSetOrthographic(), OSRSetOS(), OSRSetPolyconic(), OSRSetProjCS(), OSRSetProjection(), OSRSetProjParm(), OSRSetPS(), OSRSetQSC(), OSRSetRobinson(), OSRSetSCH(), OSRSetSinusoidal(), OSRSetSOC(), OSRSetStatePlane(), OSRSetStatePlaneWithUnits(), OSRSetStereographic(), OSRSetTargetLinearUnits(), OSRSetTM(), OSRSetTMG(), OSRSetTMSO(), OSRSetTMVariant(), OSRSetTOWGS84(), OSRSetTPED(), OSRSetUTM(), OSRSetVDG(), OSRSetVertCS(), OSRSetWagner(), OSRSetWellKnownGeogCS(), OSRStripCTParms(), OSRValidate(), OGRGeometryCollection::removeGeometry(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetExtension(), OGRFeature::SetFieldsFrom(), OGRFeature::SetFrom(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetGeogCS(), OGRFeature::SetGeometry(), OGRFeature::SetGeometryDirectly(), OGRFeature::SetGeomField(), OGRFeature::SetGeomFieldDirectly(), OGRLayer::SetIgnoredFields(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetLocalCS(), OGRLayer::SetNextByIndex(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetStatePlane(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), OGRSpatialReference::SetWellKnownGeogCS(), OGRPoint::transform(), and OGRGeometryCollection::transform().

## 12.16.2.26 OGRERR\_INVALID\_HANDLE

```
#define OGRERR_INVALID_HANDLE 8
```

Invalid handle

Referenced by OGR\_DS\_DeleteLayer(), OGR\_DS\_SyncToDisk(), OGR\_L\_AlterFieldDefn(), OGR\_L\_Clip(), OGR\_L\_CommitTransaction(), OGR\_L\_CreateFeature(), OGR\_L\_CreateField(), OGR\_L\_CreateGeomField(), OGR\_L\_DeleteFeature(), OGR\_L\_DeleteField(), OGR\_L\_Erase(), OGR\_L\_GetExtent(), OGR\_L\_GetExtentEx(), OGR\_L\_Identity(), OGR\_L\_Intersection(), OGR\_L\_ReorderField(), OGR\_L\_ReorderFields(), OGR\_L\_RollbackTransaction(), OGR\_L\_SetAttributeFilter(), OGR\_L\_SetFeature(), OGR\_L\_SetIgnoredFields(), OGR\_L\_SetNextByIndex(), OGR\_L\_StartTransaction(), OGR\_L\_SymDifference(), OGR\_L\_SyncToDisk(), OGR\_L\_Union(), and OGR\_L\_Update().

### 12.16.2.27 OGRERR\_NON\_EXISTING\_FEATURE

```
#define OGRERR_NON_EXISTING_FEATURE 9
```

Non existing feature. Added in GDAL 2.0

### 12.16.2.28 OGRERR\_NONE

```
#define OGRERR_NONE 0
```

Success

Referenced by OGRCompoundCurve::addCurve(), OGRGeometryCollection::addGeometry(), OGRPolyhedralSurface::addGeometry(), OGRTriangulatedSurface::addGeometry(), OGRGeometryCollection::addGeometryDirectly(), OGRPolyhedralSurface::addGeometryDirectly(), OGRCurvePolygon::addRing(), OGR\_SRSNode::applyRemapper(), OGRSpatialReference::AutoIdentifyEPSG(), OGRCurve::CastToCompoundCurve(), OGR\_Layer::Clip(), OGRCurvePolygon::clone(), OGRGeometryCollection::clone(), OGRPolyhedralSurface::clone(), OGR\_Layer::CommitTransaction(), OGRSpatialReference::CopyGeogCSFrom(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRFeatureDefn::DeleteFieldDefn(), OGRFeatureDefn::DeleteGeomFieldDefn(), OGRGeometry::dumpReadable(), OGR\_Layer::Erase(), OGRSpatialReference::exportToERM(), OGRSpatialReference::exportToMCoordSys(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGR\_SRSNode::exportToPrettyWkt(), OGRSpatialReference::exportToPrettyWkt(), OGRSpatialReference::exportToUSGS(), OGRPoint::exportToWkb(), OGRSimpleCurve::exportToWkb(), OGRPolygon::exportToWkb(), OGRPolyhedralSurface::exportToWkb(), OGR\_SRSNode::exportToWkt(), OGRSpatialReference::exportToWkt(), OGRPoint::exportToWkt(), OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRSpatialReference::exportToXML(), OGR\_SRSNode::FixupOrdering(), OGRSpatialReference::FixupOrdering(), OGRGeometryFactory::forceTo(), OGRSpatialReference::GetEccentricity(), OGRFeature::GetFieldAsString(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetNormProjParm(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetSquaredEccentricity(), OGRSpatialReference::GetTOWGS84(), OGR\_Layer::Identity(), OGRSpatialReference::importFromDict(), OGRSpatialReference::importFromEPSG(), OGRSpatialReference::importFromERM(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::ImportFromESRIStatePlaneWKT(), OGRSpatialReference::ImportFromESRIWisconsinWKT(), OGRSpatialReference::importFromMCoordSys(), OGRSpatialReference::importFromPanorama(), OGRSpatialReference::importFromProj4(), OGRPoint::importFromWkb(), OGRSimpleCurve::importFromWkb(), OGRCircularString::importFromWkb(), OGRCompoundCurve::importFromWkb(), OGRCurvePolygon::importFromWkb(), OGRPolygon::importFromWkb(), OGRTriangle::importFromWkb(), OGRPolyhedralSurface::importFromWkb(), OGRSpatialReference::importFromWkt(), OGRPoint::importFromWkt(), OGRSimpleCurve::importFromWkt(), OGRCircularString::importFromWkt(), OGRPolygon::importFromWkt(), OGRMultiSurface::importFromWkt(), OGRPolyhedralSurface::importFromWkt(), OGRMultiPoint::importFromWkt(), OGR\_Layer::Intersection(), OGRSpatialReference::IsSameGeogCS(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGR\_DS\_SyncToDisk(), OGR\_G\_AddGeometryDirectly(), OGRTriangle::OGRTriangle(), OGRGeometryCollection::removeGeometry(), OGRFeatureDefn::ReorderFieldDefns(), OGRSpatialReference::SetACEA(), OGRSpatialReference::SetAE(), OGRSpatialReference::SetAngularUnits(), OGR\_Layer::SetAttributeFilter(), OGRSpatialReference::SetAuthority(),



OGRSpatialReference::SetAxes(), OGRSpatialReference::SetBonne(), OGRSpatialReference::SetCEA(), OGRSpatialReference::SetCS(), OGRSpatialReference::SetEC(), OGRSpatialReference::SetEckertIV(), OGRSpatialReference::SetEckertVI(), OGRSpatialReference::SetEquirectangular(), OGRSpatialReference::SetEquirectangular2(), OGRSpatialReference::SetExtension(), OGRFeature::SetFID(), OGRFeature::SetFieldsFrom(), OGRFeature::SetFrom(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGaussSchreiberTMercator(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetGeogCS(), OGRFeature::SetGeomField(), OGRFeature::SetGeomFieldDirectly(), OGRSpatialReference::SetGEOS(), OGRSpatialReference::SetGH(), OGRSpatialReference::SetGnomonic(), OGRSpatialReference::SetGS(), OGRSpatialReference::SetHOM(), OGRSpatialReference::SetHOM2PNO(), OGRSpatialReference::SetHOMAC(), OGRSpatialReference::SetIGH(), OGRLayer::SetIgnoredFields(), OGRSpatialReference::SetIWMPolyconic(), OGRSpatialReference::SetKrovak(), OGRSpatialReference::SetLAEA(), OGRSpatialReference::SetLCC(), OGRSpatialReference::SetLCC1SP(), OGRSpatialReference::SetLCCB(), OGRSpatialReference::SetLocalCS(), OGRSpatialReference::SetMC(), OGRSpatialReference::SetMercator(), OGRSpatialReference::SetMercator2SP(), OGRSpatialReference::SetMollweide(), OGRLayer::SetNextByIndex(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetNZMG(), OGRSpatialReference::SetOrthographic(), OGRSpatialReference::SetOS(), OGRSpatialReference::SetPolyconic(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetPS(), OGRSpatialReference::SetQSC(), OGRSpatialReference::SetRobinson(), OGRSpatialReference::SetSCH(), OGRSpatialReference::SetSinusoidal(), OGRSpatialReference::SetSOC(), OGRSpatialReference::SetStereographic(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTM(), OGRSpatialReference::SetTMG(), OGRSpatialReference::SetTMSO(), OGRSpatialReference::SetTMVariant(), OGRSpatialReference::SetTOWGS84(), OGRSpatialReference::SetTPED(), OGRSpatialReference::SetVDG(), OGRSpatialReference::SetVertCS(), OGRSpatialReference::SetWellKnownGeogCS(), OGRLayer::StartTransaction(), OGRSpatialReference::StripCTParms(), OGRSpatialReference::StripVertical(), OGRLayer::SymDifference(), OGRLayer::SyncToDisk(), OGRPoint::transform(), OGRGeometryCollection::transform(), OGRGeometryFactory::transformWithOptions(), OGRLayer::Union(), OGRLayer::Update(), and OGRSpatialReference::Validate().

#### 12.16.2.29 OGRERR\_NOT\_ENOUGH\_DATA

```
#define OGRERR_NOT_ENOUGH_DATA 1
```

Not enough data to deserialize

Referenced by OGRGeometryFactory::createFromWkb(), OGRSpatialReference::importFromOzi(), OGRPoint::importFromWkb(), and OGRPolyhedralSurface::importFromWkb().

#### 12.16.2.30 OGRERR\_NOT\_ENOUGH\_MEMORY

```
#define OGRERR_NOT_ENOUGH_MEMORY 2
```

Not enough memory

Referenced by OGRSimpleCurve::exportToWkt(), OGRPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRSpatialReference::importFromPanorama(), OGRSpatialReference::importFromPCI(), OGRPolyhedralSurface::importFromWkb(), and OGRSimpleCurve::transform().

### 12.16.2.31 OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE

```
#define OGRERR_UNSUPPORTED_GEOMETRY_TYPE 3
```

Unsupported geometry type

Referenced by OGRPolyhedralSurface::addGeometry(), OGRTriangulatedSurface::addGeometry(), OGRGeometryCollection::addGeometryDirectly(), OGRPolyhedralSurface::addGeometryDirectly(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGR\_G\_AddGeometry(), and OGR\_G\_AddGeometryDirectly().

### 12.16.2.32 OGRERR\_UNSUPPORTED\_OPERATION

```
#define OGRERR_UNSUPPORTED_OPERATION 4
```

Unsupported operation

Referenced by OGRLayer::Clip(), OGRLayer::DeleteFeature(), OGRLayer::Erase(), OGRLinearRing::exportToWkb(), OGRLayer::!CreateFeature(), OGRLayer::!Identity(), OGRLinearRing::importFromWkb(), OGRLayer::!Intersection(), OGRLayer::!SetFeature(), OGR\_G\_AddGeometry(), OGR\_G\_AddGeometryDirectly(), OGRLayer::!RollbackTransaction(), OGRLayer::SymDifference(), OGRLayer::Union(), and OGRLayer::Update().

### 12.16.2.33 OGRERR\_UNSUPPORTED\_SRS

```
#define OGRERR_UNSUPPORTED_SRS 7
```

Unsupported SRS

Referenced by OGRSpatialReference::AutoidentifyEPSG(), OGRSpatialReference::exportToERM(), OGRSpatialReference::exportToXML(), OGRSpatialReference::importFromDict(), and OGRSpatialReference::importFromXML().

### 12.16.2.34 OGRNullFID

```
#define OGRNullFID -1
```

Special value for a unset FID

Referenced by OGRFeature::!IsFieldSet(), and OGRFeature::SetFrom().

### 12.16.2.35 OGRNullMarker

```
#define OGRNullMarker -21122
```

Special value set in OGRField.Set.nMarker1, nMarker2 and nMarker3 for a null field. Direct use of this value is strongly discouraged. Use **OGR\_RawField\_SetNull()** (p. ??) or **OGR\_RawField\_IsNull()** (p. ??) instead.

Since

GDAL 2.2

Referenced by OGR\_RawField\_IsNull(), and OGR\_RawField\_SetNull().

### 12.16.2.36 OGRUnsetMarker

```
#define OGRUnsetMarker -21121
```

Special value set in OGRField.Set.nMarker1, nMarker2 and nMarker3 for a unset field. Direct use of this value is strongly discouraged. Use **OGR\_RawField\_SetUnset()** (p. ??) or **OGR\_RawField\_IsUnset()** (p. ??) instead.

Referenced by OGR\_RawField\_IsUnset(), and OGR\_RawField\_SetUnset().

### 12.16.2.37 OLCAAlterFieldDefn

```
#define OLCAAlterFieldDefn "AlterFieldDefn"
```

Layer capability for field alteration

### 12.16.2.38 OLCCreateField

```
#define OLCCreateField "CreateField"
```

Layer capability for field creation

### 12.16.2.39 OLCCreateGeomField

```
#define OLCCreateGeomField "CreateGeomField"
```

Layer capability for geometry field creation

### 12.16.2.40 OLCCurveGeometries

```
#define OLCCurveGeometries "CurveGeometries"
```

Layer capability for curve geometries support

**12.16.2.41 OLCDeleteFeature**

```
#define OLCDeleteFeature "DeleteFeature"
```

Layer capability for feature deletion

**12.16.2.42 OLCDeleteField**

```
#define OLCDeleteField "DeleteField"
```

Layer capability for field deletion

**12.16.2.43 OLCFastFeatureCount**

```
#define OLCFastFeatureCount "FastFeatureCount"
```

Layer capability for fast feature count retrieval

**12.16.2.44 OLCFastGetExtent**

```
#define OLCFastGetExtent "FastGetExtent"
```

Layer capability for fast extent retrieval

**12.16.2.45 OLCFastSetNextByIndex**

```
#define OLCFastSetNextByIndex "FastSetNextByIndex"
```

Layer capability for setting next feature index

**12.16.2.46 OLCFastSpatialFilter**

```
#define OLCFastSpatialFilter "FastSpatialFilter"
```

Layer capability for fast spatial filter

**12.16.2.47 OLCIgnoreFields**

```
#define OLCIgnoreFields "IgnoreFields"
```

Layer capability for field ignoring

**12.16.2.48 OLCMeasuredGeometries**

```
#define OLCMeasuredGeometries "MeasuredGeometries"
```

Layer capability for measured geometries support

**12.16.2.49 OLCRandomRead**

```
#define OLCRandomRead "RandomRead"
```

Layer capability for random read

**12.16.2.50 OLCRandomWrite**

```
#define OLCRandomWrite "RandomWrite"
```

Layer capability for random write

**12.16.2.51 OLCReorderFields**

```
#define OLCReorderFields "ReorderFields"
```

Layer capability for field reordering

**12.16.2.52 OLCSequentialWrite**

```
#define OLCSequentialWrite "SequentialWrite"
```

Layer capability for sequential write

**12.16.2.53 OLCStringsAsUTF8**

```
#define OLCStringsAsUTF8 "StringsAsUTF8"
```

Layer capability for strings returned with UTF-8 encoding

**12.16.2.54 OLCTransactions**

```
#define OLCTransactions "Transactions"
```

Layer capability for transactions

**12.16.2.55 OLMD\_FID64**

```
#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

**12.16.2.56 wkb25DBit**

```
#define wkb25DBit 0x80000000
```

**Deprecated** in GDAL 2.0. Use **wkbHasZ()** (p. ??) or **wkbSetZ()** (p. ??) instead

**12.16.2.57 wkbFlatten**

```
#define wkbFlatten(  
    x )  OGR_GT_Flatten(static_cast< OGRwkbGeometryType>(x))
```

Return the 2D geometry type corresponding to the specified geometry type

Referenced by OGRGeometryCollection::CastToGeometryCollection(), OGRGeometryCollection::closeRings(), OGRCurvePolygon::Contains(), OGRGeometryFactory::createGeometry(), OGRGeometry::dumpReadable(), OGRPoint::exportToWkb(), OGRSimpleCurve::exportToWkb(), OGRPolygon::exportToWkb(), OGRGeometryCollection::exportToWkb(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRGeometryCollection::get\_Area(), OGRGeometryCollection::get\_Length(), OGRPolygon::getCurveGeometry(), OGRGeometry::getIsoGeometryType(), OGRMultiCurve::importFromWkt(), OGRPoint::Intersects(), OGRCurvePolygon::Intersects(), OGRMultiSurface::isCompatibleSubType(), OGRMultiPolygon::isCompatibleSubType(), OGRMultiPoint::isCompatibleSubType(), OGRMultiLineString::isCompatibleSubType(), OGR\_G\_AddGeometry(), OGR\_G\_AddGeometryDirectly(), OGR\_G\_AddPoint(), OGR\_G\_AddPoint\_2D(), OGR\_G\_AddPointM(), OGR\_G\_AddPointZM(), OGR\_G\_Area(), OGR\_G\_Centroid(), OGR\_G\_GetGeometryCount(), OGR\_G\_GetGeometryRef(), OGR\_G\_GetPoint(), OGR\_G\_GetPointCount(), OGR\_G\_GetPoints(), OGR\_G\_GetPointsZM(), OGR\_G\_GetPointZM(), OGR\_G\_Length(), OGR\_G\_RemoveGeometry(), OGR\_G\_SetPoint(), OGR\_G\_SetPoint\_2D(), OGR\_G\_SetPointCount(), OGR\_G\_SetPointM(), OGR\_G\_SetPointZM(), OGR\_GT\_GetCollection(), OGR\_GT\_GetCurve(), OGR\_GT\_GetLinear(), OGR\_GT\_IsNonLinear(), OGR\_GT\_IsSubClassOf(), OGR\_GT\_SetModifier(), OGRGeometryTypeToName(), OGRMergeGeometryTypesEx(), OGRToOGCGeomType(), OGRGeometryFactory::transformWithOptions(), and OGRPoint::Within().

**12.16.2.58 wkbHasM**

```
#define wkbHasM(  
    x ) ( OGR_GT_HasM(x) != 0)
```

Return if the geometry type is a measured geometry type

Since

GDAL 2.1

Referenced by OGR\_GT\_GetCollection(), OGR\_GT\_GetCurve(), OGR\_GT\_GetLinear(), OGRGeometryTypeToName(), and OGRMergeGeometryTypesEx().

### 12.16.2.59 wkbHasZ

```
#define wkbHasZ(  
    x ) ( OGR_GT_HasZ(x) != 0)
```

Return if the geometry type is a 3D geometry type

Since

GDAL 2.0

Referenced by OGRGeometryCollection::exportToWkb(), OGR\_GT\_GetCollection(), OGR\_GT\_GetCurve(), OGR\_GT\_GetLinear(), OGRGeometryTypeToName(), and OGRMergeGeometryTypesEx().

### 12.16.2.60 wkbSetM

```
#define wkbSetM(  
    x ) OGR_GT_SetM(x)
```

Return the measured geometry type corresponding to the specified geometry type.

Since

GDAL 2.1

Referenced by OGR\_GT\_GetCollection(), OGR\_GT\_GetCurve(), OGR\_GT\_GetLinear(), and OGRFromOGCGeomType().

### 12.16.2.61 wkbSetZ

```
#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 OGRFromOGCGeomType().

## 12.16.3 Typedef Documentation

### 12.16.3.1 OGRBoolean

```
typedef int OGRBoolean
```

Type for a OGR boolean

### 12.16.3.2 OGRErr

```
typedef int OGRErr
```

Simple container for a bounding region.

Type for a OGR error

### 12.16.3.3 OGRSTBrushParam

```
typedef enum ogr_style_tool_param_brush_id OGRSTBrushParam
```

List of parameters for use with OGRStyleBrush.

### 12.16.3.4 OGRSTClassId

```
typedef enum ogr_style_tool_class_id OGRSTClassId
```

**OGRStyleTool** (p. ??) derived class types (returned by GetType()).

### 12.16.3.5 OGRSTLabelParam

```
typedef enum ogr_style_tool_param_label_id OGRSTLabelParam
```

List of parameters for use with OGRStyleLabel.

### 12.16.3.6 OGRSTPenParam

```
typedef enum ogr_style_tool_param_pen_id OGRSTPenParam
```

List of parameters for use with OGRStylePen.

### 12.16.3.7 OGRSTSymbolParam

```
typedef enum ogr_style_tool_param_symbol_id OGRSTSymbolParam
```

List of parameters for use with OGRStyleSymbol.

### 12.16.3.8 OGRSTUnitId

```
typedef enum ogr_style_tool_units_id OGRSTUnitId
```

List of units supported by OGRStyleTools.

## 12.16.4 Enumeration Type Documentation

### 12.16.4.1 ogr\_style\_tool\_class\_id

```
enum ogr_style_tool_class_id
```

**OGRStyleTool** (p. ??) derived class types (returned by GetType()).



## Enumerator

OGRSTCNone	None
OGRSTCPen	Pen
OGRSTCBrush	Brush
OGRSTCSymbol	Symbol
OGRSTCLabel	Label
OGRSTCVector	Vector

## 12.16.4.2 ogr\_style\_tool\_param\_brush\_id

```
enum ogr_style_tool_param_brush_id
```

List of parameters for use with OGRStyleBrush.

## Enumerator

OGRSTBrushFColor	Foreground color
OGRSTBrushBColor	Background color
OGRSTBrushId	Id
OGRSTBrushAngle	Angle
OGRSTBrushSize	Size
OGRSTBrushDx	Dx
OGRSTBrushDy	Dy
OGRSTBrushPriority	Priority

## 12.16.4.3 ogr\_style\_tool\_param\_label\_id

```
enum ogr_style_tool_param_label_id
```

List of parameters for use with OGRStyleLabel.

## Enumerator

OGRSTLabelFontName	Font name
OGRSTLabelSize	Size
OGRSTLabelTextString	Text string
OGRSTLabelAngle	Angle
OGRSTLabelFColor	Foreground color
OGRSTLabelBColor	Background color
OGRSTLabelPlacement	Placement
OGRSTLabelAnchor	Anchor
OGRSTLabelDx	Dx
OGRSTLabelDy	Dy

## Enumerator

OGRSTLabelPerp	Perpendicular
OGRSTLabelBold	Bold
OGRSTLabelItalic	Italic
OGRSTLabelUnderline	Underline
OGRSTLabelPriority	Priority
OGRSTLabelStrikeout	Strike out
OGRSTLabelStretch	Stretch
OGRSTLabelAdjHor	OBSOLETE; do not use
OGRSTLabelAdjVert	OBSOLETE; do not use
OGRSTLabelHColor	Highlight color
OGRSTLabelOColor	Outline color

## 12.16.4.4 ogr\_style\_tool\_param\_pen\_id

```
enum ogr_style_tool_param_pen_id
```

List of parameters for use with OGRStylePen.

## Enumerator

OGRSTPenColor	Color
OGRSTPenWidth	Width
OGRSTPenPattern	Pattern
OGRSTPenId	Id
OGRSTPenPerOffset	Perpendicular offset
OGRSTPenCap	Cap
OGRSTPenJoin	Join
OGRSTPenPriority	Priority

## 12.16.4.5 ogr\_style\_tool\_param\_symbol\_id

```
enum ogr_style_tool_param_symbol_id
```

List of parameters for use with OGRStyleSymbol.

## Enumerator

OGRSTSymbolId	Id
OGRSTSymbolAngle	Angle
OGRSTSymbolColor	Color
OGRSTSymbolSize	Size
OGRSTSymbolDx	Dx

## Enumerator

OGRSTSymbolDy	Dy
OGRSTSymbolStep	Step
OGRSTSymbolPerp	Perpendicular
OGRSTSymbolOffset	Offset
OGRSTSymbolPriority	Priority
OGRSTSymbolFontName	OBSOLETE; do not use
OGRSTSymbolOColor	Outline color

## 12.16.4.6 ogr\_style\_tool\_units\_id

```
enum ogr_style_tool_units_id
```

List of units supported by OGRStyleTools.

## Enumerator

OGRSTUGround	Ground unit
OGRSTUPixel	Pixel
OGRSTUPoints	Points
OGRSTUMM	Millimeter
OGRSTUCM	Centimeter
OGRSTUInches	Inch

## 12.16.4.7 OGRFieldSubType

```
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) floating point. Only valid for OFTReal and OFTRealList.

#### 12.16.4.8 OGRFieldType

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

#### 12.16.4.9 OGRJustification

enum **OGRJustification**

Display justification for field values.

#### 12.16.4.10 OGRwkbByteOrder

enum **OGRwkbByteOrder**

Enumeration to describe byte order

##### Enumerator

wkbXDR	MSB/Sun/Motorola: Most Significant Byte First
wkbNDR	LSB/Intel/Vax: Least Significant Byte First

## 12.16.4.11 OGRwkbGeometryType

enum **OGRwkbGeometryType**

List of well known binary geometry types. These are used within the BLOBs but are also returned from **OGRGeometry::getGeometryType()** (p. ??) to identify the type of a geometry object.

## Enumerator

wkbUnknown	unknown type, non-standard
wkbPoint	0-dimensional geometric object, standard WKB
wkbLineString	1-dimensional geometric object with linear interpolation between Points, standard WKB
wkbPolygon	planar 2-dimensional geometric object defined by 1 exterior boundary and 0 or more interior boundaries, standard WKB
wkbMultiPoint	GeometryCollection of Points, standard WKB
wkbMultiLineString	GeometryCollection of LineStrings, standard WKB
wkbMultiPolygon	GeometryCollection of Polygons, standard WKB
wkbGeometryCollection	geometric object that is a collection of 1 or more geometric objects, standard WKB
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
wkbCurve	Curve (abstract type). ISO SQL/MM Part 3. GDAL >= 2.1
wkbSurface	Surface (abstract type). ISO SQL/MM Part 3. GDAL >= 2.1
wkbPolyhedralSurface	a contiguous collection of polygons, which share common boundary segments, ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbTIN	a PolyhedralSurface consisting only of Triangle patches ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbTriangle	a Triangle. ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
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
wkbCurveZ	wkbCurve with Z component. ISO SQL/MM Part 3. GDAL >= 2.1
wkbSurfaceZ	wkbSurface with Z component. ISO SQL/MM Part 3. GDAL >= 2.1
wkbPolyhedralSurfaceZ	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbTINZ	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbTriangleZ	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbPointM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbLineStringM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbPolygonM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiPointM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiLineStringM	ISO SQL/MM Part 3. GDAL >= 2.1

## Enumerator

wkbMultiPolygonM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbGeometryCollectionM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbCircularStringM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbCompoundCurveM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbCurvePolygonM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiCurveM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiSurfaceM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbCurveM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbSurfaceM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbPolyhedralSurfaceM	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbTINM	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbTriangleM	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbPointZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbLineStringZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbPolygonZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiPointZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiLineStringZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiPolygonZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbGeometryCollectionZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbCircularStringZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbCompoundCurveZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbCurvePolygonZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiCurveZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbMultiSurfaceZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbCurveZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbSurfaceZM	ISO SQL/MM Part 3. GDAL >= 2.1
wkbPolyhedralSurfaceZM	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbTINZM	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
wkbTriangleZM	ISO SQL/MM Part 3. Reserved in GDAL >= 2.1 but not yet implemented
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

## 12.16.4.12 OGRwkbVariant

```
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 coordinates 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\_↵ ISO/IEC\_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

## 12.16.5 Function Documentation

### 12.16.5.1 OGR\_GET\_MS()

```
int OGR_GET_MS (
    float fSec ) [inline]
```

Return the number of milliseconds from a datetime with decimal seconds

Referenced by OGRFeature::GetFieldAsString().

### 12.16.5.2 OGR\_GT\_Flatten()

```
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

<i>eType</i>	Input geometry type
--------------	---------------------

#### Returns

2D geometry type corresponding to the passed geometry type.

#### Since

GDAL 2.0

### 12.16.5.3 OGR\_GT\_GetCollection()

```
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->wkbMultiLineString, wkbPolygon/wkbTriangle/wkbPolyhedralSurface/wkbTIN->wkbMultiPolygon, wkbCircularString->wkbMultiCurve, wkbCompoundCurve->wkbMultiCurve, wkbCurvePolygon->wkbMultiSurface. In other cases, wkbUnknown is returned

Passed Z, M, ZM flag is preserved.

#### Parameters

<i>eType</i>	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, wkbHasM, wkbHasZ, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbNone, wkbPoint, wkbPolygon, wkbSetM, wkbSetZ, wkbTIN, wkbTriangle, and wkbUnknown.

Referenced by OGRGeometryFactory::forceTo().

### 12.16.5.4 OGR\_GT\_GetCurve()

```
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, wkbMultiPolygon->wkbMultiSurface and wkbMultiLineString->wkbMultiCurve. In other cases, the passed geometry is returned.

Passed Z, M, ZM flag is preserved.

#### Parameters

<i>eType</i>	Input geometry type
--------------	---------------------



**Returns**

the curve type that can contain the passed geometry type

**Since**

GDAL 2.0

References wkbCompoundCurve, wkbCurvePolygon, wkbFlatten, wkbHasM, wkbHasZ, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, wkbSetM, wkbSetZ, and wkbTriangle.

Referenced by OGRGeometryCollection::getCurveGeometry().

**12.16.5.5 OGR\_GT\_GetLinear()**

```
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, wkbCompoundCurve->wkbLineString, wkbMultiSurface->wkbMultiPolygon and wkbMultiCurve->wkbMultiLineString. In other cases, the passed geometry is returned.

Passed Z, M, ZM flag is preserved.

**Parameters**

<i>eType</i>	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, wkbHasM, wkbHasZ, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, wkbSetM, and wkbSetZ.

Referenced by OGRGeometryCollection::getLinearGeometry(), OGR\_F\_GetGeometryRef(), OGR\_F\_GetGeomFieldRef(), OGR\_FD\_GetGeomType(), OGR\_GFld\_GetType(), and OGR\_L\_GetGeomType().

**12.16.5.6 OGR\_GT\_HasM()**

```
int OGR_GT_HasM (
    OGRwkbGeometryType eType )
```

Return if the geometry type is a measured type.

**Parameters**

<i>eType</i>	Input geometry type
--------------	---------------------

**Returns**

TRUE if the geometry type is a measured type.

**Since**

GDAL 2.1

Referenced by OGR\_GT\_SetM().

**12.16.5.7 OGR\_GT\_HasZ()**

```
int OGR_GT_HasZ (
    OGRwkbGeometryType eType )
```

Return if the geometry type is a 3D geometry type.

**Parameters**

<i>eType</i>	Input geometry type
--------------	---------------------

**Returns**

TRUE if the geometry type is a 3D geometry type.

**Since**

GDAL 2.0

Referenced by OGR\_GT\_SetZ().

**12.16.5.8 OGR\_GT\_IsCurve()**

```
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 Z/M/ZM variant.

## Parameters

<i>eGeomType</i>	the geometry type
------------------	-------------------

## Returns

TRUE if the geometry type is an instance of Curve

## Since

GDAL 2.0

References OGR\_GT\_IsSubClassOf(), and wkbCurve.

Referenced by OGRGeometryFactory::forceToPolygon(), OGRGeometryCollection::get\_Area(), OGRGeometryCollection::get\_Length(), OGRMultiCurve::isCompatibleSubType(), OGR\_G\_AddGeometry(), OGR\_G\_AddGeometryDirectly(), OGR\_G\_Area(), OGR\_G\_GetPointCount(), OGR\_G\_Length(), OGR\_G\_Value(), OGR\_GT\_GetCollection(), OGR\_GT\_GetLinear(), and OGRMergeGeometryTypesEx().

## 12.16.5.9 OGR\_GT\_IsNonLinear()

```
int OGR_GT_IsNonLinear (
    OGRwkbGeometryType eGeomType )
```

Return if a geometry type is a non-linear geometry type.

Such geometry type are wkbCurve, wkbCircularString, wkbCompoundCurve, wkbSurface, wkbCurvePolygon, wkbMultiCurve, wkbMultiSurface and their Z/M variants.

## Parameters

<i>eGeomType</i>	the geometry type
------------------	-------------------

## Returns

TRUE if the geometry type is a non-linear geometry type.

## Since

GDAL 2.0

References wkbCircularString, wkbCompoundCurve, wkbCurve, wkbCurvePolygon, wkbFlatten, wkbMultiCurve, wkbMultiSurface, and wkbSurface.

Referenced by OGR\_F\_GetGeometryRef(), OGR\_F\_GetGeomFieldRef(), OGR\_FD\_GetGeomType(), OGR\_G\_Fld\_GetType(), and OGR\_L\_GetGeomType().

## 12.16.5.10 OGR\_GT\_IsSubClassOf()

```
int OGR_GT_IsSubClassOf (
    OGRwkbGeometryType eType,
    OGRwkbGeometryType eSuperType )
```

Returns if a type is a subclass of another one.

## Parameters

<i>eType</i>	Type.
<i>eSuperType</i>	Super type

## Returns

TRUE if eType is a subclass of eSuperType.

## Since

GDAL 2.0

References wkbCircularString, wkbCompoundCurve, wkbCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbPolygon, wkbPolyhedralSurface, wkbSurface, wkbTIN, wkbTriangle, and wkbUnknown.

Referenced by OGRGeometryCollection::closeRings(), OGRGeometryFactory::forceTo(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), 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().

## 12.16.5.11 OGR\_GT\_IsSurface()

```
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 Z/M/ZM variant.

## Parameters

<i>eGeomType</i>	the geometry type
------------------	-------------------

## Returns

TRUE if the geometry type is an instance of Surface

Since

GDAL 2.0

References OGR\_GT\_IsSubClassOf(), and wkbSurface.

Referenced by OGRGeometryCollection::get\_Area(), OGR\_G\_Area(), OGR\_GT\_GetCollection(), and OGR\_GT↔\_GetLinear().

#### 12.16.5.12 OGR\_GT\_SetM()

```
OGRwkbGeometryType OGR_GT_SetM (
    OGRwkbGeometryType eType )
```

Returns the measured geometry type corresponding to the passed geometry type.

Parameters

<i>eType</i>	Input geometry type
--------------	---------------------

Returns

measured geometry type corresponding to the passed geometry type.

Since

GDAL 2.1

References OGR\_GT\_HasM(), and wkbNone.

Referenced by OGR\_GT\_SetModifier().

#### 12.16.5.13 OGR\_GT\_SetModifier()

```
OGRwkbGeometryType OGR_GT_SetModifier (
    OGRwkbGeometryType eType,
    int bHasZ,
    int bHasM )
```

Returns a XY, XYZ, XYM or XYZM geometry type depending on parameter.

Parameters

<i>eType</i>	Input geometry type
<i>bHasZ</i>	TRUE if the output geometry type must be 3D.
<i>bHasM</i>	TRUE if the output geometry type must be measured.

**Returns**

Output geometry type.

**Since**

GDAL 2.0

References OGR\_GT\_SetM(), OGR\_GT\_SetZ(), and wkbFlatten.

Referenced by OGRMergeGeometryTypesEx().

**12.16.5.14 OGR\_GT\_SetZ()**

```
OGRwkbGeometryType OGR_GT_SetZ (
    OGRwkbGeometryType eType )
```

Returns the 3D geometry type corresponding to the passed geometry type.

**Parameters**

<i>eType</i>	Input geometry type
--------------	---------------------

**Returns**

3D geometry type corresponding to the passed geometry type.

**Since**

GDAL 2.0

References OGR\_GT\_HasZ(), wkbGeometryCollection, and wkbNone.

Referenced by OGR\_GT\_SetModifier().

**12.16.5.15 OGRGeometryTypeToName()**

```
const char* OGRGeometryTypeToName (
    OGRwkbGeometryType eType )
```

Fetch a human readable name corresponding to an OGRwkbGeometryType value. The returned value should not be modified, or freed by the application.

This function is C callable.

## Parameters

<i>eType</i>	the geometry type.
--------------	--------------------

## Returns

internal human readable string, or NULL on failure.

References CPLPrintf(), wkbCircularString, wkbCompoundCurve, wkbCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbHasM, wkbHasZ, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbNone, wkbPoint, wkbPolygon, wkbPolyhedralSurface, wkbSurface, wkbTIN, wkbTriangle, and wkbUnknown.

## 12.16.5.16 OGRMergeGeometryTypes()

```
OGRwkbGeometryType OGRMergeGeometryTypes (
    OGRwkbGeometryType eMain,
    OGRwkbGeometryType eExtra )
```

Find common geometry type.

Given two geometry types, find the most specific common type. Normally used repeatedly with the geometries in a layer to try and establish the most specific geometry type that can be reported for the layer.

NOTE: wkbUnknown is the "worst case" indicating a mixture of geometry types with nothing in common but the base geometry type. wkbNone should be used to indicate that no geometries have been encountered yet, and means the first geometry encountered will establish the preliminary type.

## Parameters

<i>eMain</i>	the first input geometry type.
<i>eExtra</i>	the second input geometry type.

## Returns

the merged geometry type.

References OGRMergeGeometryTypesEx().

## 12.16.5.17 OGRMergeGeometryTypesEx()

```
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 encountered 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

<i>eMain</i>	the first input geometry type.
<i>eExtra</i>	the second input geometry type.
<i>bAllowPromotingToCurves</i>	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()`, `OGR_GT_SetModifier()`, `wkbCompoundCurve`, `wkbFlatten`, `wkbGeometryCollection`, `wkbHasM`, `wkbHasZ`, `wkbNone`, and `wkbUnknown`.

Referenced by `OGRMergeGeometryTypes()`.

## 12.17 ogr\_feature.h File Reference

```
#include "cpl_atomic_ops.h"
#include "ogr_featurestyle.h"
#include "ogr_geometry.h"
#include <exception>
#include <memory>
#include <string>
#include <vector>
```

#### Classes

- class **OGRFieldDefn**
- class **OGRGeomFieldDefn**
- class **OGRFeatureDefn**
- class **OGRFeature**
- class **OGRFeature::FieldValue**
- class **OGRFeature::ConstFieldIterator**
- class **OGRFeature::FieldNotFoundException**



## Typedefs

- typedef void \* **OGRFieldDefnH**
- typedef void \* **OGRFeatureDefnH**
- typedef void \* **OGRFeatureH**
- typedef void \* **OGRStyleTableH**
- typedef struct OGRGeomFieldDefnHS \* **OGRGeomFieldDefnH**
- typedef std::unique\_ptr< **OGRFeature**, OGRFeatureUniquePtrDeleter > **OGRFeatureUniquePtr**

### 12.17.1 Detailed Description

Simple feature classes.

### 12.17.2 Typedef Documentation

#### 12.17.2.1 OGRFeatureDefnH

```
typedef void* OGRFeatureDefnH
```

Opaque type for a feature definition (**OGRFeatureDefn** (p. ??))

#### 12.17.2.2 OGRFeatureH

```
typedef void* OGRFeatureH
```

Opaque type for a feature (**OGRFeature** (p. ??))

#### 12.17.2.3 OGRFeatureUniquePtr

```
typedef std::unique_ptr< OGRFeature, OGRFeatureUniquePtrDeleter> OGRFeatureUniquePtr
```

Unique pointer type for **OGRFeature** (p. ??).

Since

GDAL 2.3

#### 12.17.2.4 OGRFieldDefnH

```
typedef void* OGRFieldDefnH
```

Opaque type for a field definition (**OGRFieldDefn** (p. ??))

### 12.17.2.5 OGRGeomFieldDefnH

```
typedef struct OGRGeomFieldDefnHS* OGRGeomFieldDefnH
```

Opaque type for a geometry field definition (**OGRGeomFieldDefn** (p. ??))

### 12.17.2.6 OGRStyleTableH

```
typedef void* OGRStyleTableH
```

Opaque type for a style table (**OGRStyleTable** (p. ??))

## 12.18 ogr\_featurestyle.h File Reference

```
#include "cpl_conv.h"  
#include "cpl_string.h"  
#include "ogr_core.h"
```

### Classes

- class **OGRStyleTable**
- class **OGRStyleMgr**
- class **OGRStyleTool**

### Typedefs

- typedef enum **ogr\_style\_type** **OGRSType**

### Enumerations

- enum **ogr\_style\_type**

### 12.18.1 Detailed Description

Simple feature style classes.

### 12.18.2 Typedef Documentation

### 12.18.2.1 OGRStyleType

```
typedef enum ogr_style_type OGRStyleType
```

OGR Style type

## 12.18.3 Enumeration Type Documentation

### 12.18.3.1 ogr\_style\_type

```
enum ogr_style_type
```

OGR Style type

## 12.19 ogr\_geocoding.h File Reference

```
#include "cpl_port.h"
#include "ogr_api.h"
```

### Typedefs

- typedef struct **\_OGRGeocodingSessionHS** \* **OGRGeocodingSessionH**

### 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.*

### 12.19.1 Detailed Description

C API for geocoding client.

## 12.19.2 Typedef Documentation

### 12.19.2.1 OGRGeocodingSessionH

```
typedef struct _OGRGeocodingSessionHS* OGRGeocodingSessionH
```

Opaque type for a geocoding session

## 12.19.3 Function Documentation

### 12.19.3.1 OGRGeocode()

```
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 follow ISO 3166-1, i.e. gb for United Kingdom, de for Germany, etc.. (Known to work with OSM and MapQuest Nominatim)
- LIMIT=number: the number of records to return. Unlimited if not specified. (Known to work with OSM and MapQuest Nominatim)
- RAW\_FEATURE=YES: to specify that a 'raw' field must be added to the returned feature with the raw XML content.
- EXTRA\_QUERY\_PARAMETERS=params: additional parameters for the GET request.

## Parameters

<i>hSession</i>	the geocoding session handle.
<i>pszQuery</i>	the string to geocode.
<i>papszStructuredQuery</i>	unused for now. Must be NULL.
<i>papszOptions</i>	a list of options or NULL.

## Returns

a OGR layer with the result(s), or NULL in case of error. The returned layer must be freed with **OGRGeocodeFreeResult()** (p. ??).

## Since

GDAL 1.10

References CPLError(), and VALIDATE\_POINTER1.

## 12.19.3.2 OGRGeocodeCreateSession()

```
OGRGeocodingSessionH OGRGeocodeCreateSession (
    char ** papszOptions )
```

Creates a session handle for geocoding requests.

Available papszOptions values:

- "CACHE\_FILE" : Defaults to "ogr\_geocode\_cache.sqlite" (or otherwise "ogr\_geocode\_cache.csv" if the SQLite driver isn't available). Might be any CSV, SQLite or PostgreSQL datasource.
- "READ\_CACHE" : "TRUE" (default) or "FALSE"
- "WRITE\_CACHE" : "TRUE" (default) or "FALSE"
- "SERVICE": "OSM\_NOMINATIM" (default), "MAPQUEST\_NOMINATIM", "YAHOO", "GEONAMES", "BING" or other value. Note: "YAHOO" is no longer available as a free service.
- "EMAIL": used by OSM\_NOMINATIM. Optional, but recommended.
- "USERNAME": used by GEONAMES. Compulsory in that case.
- "KEY": used by BING. Compulsory in that case.
- "APPLICATION": used to set the User-Agent MIME header. Defaults to GDAL/OGR version string.
- "LANGUAGE": used to set the Accept-Language MIME header. Preferred language order for showing search results.
- "DELAY": minimum delay, in second, between 2 consecutive queries. Defaults to 1.0.
- "QUERY\_TEMPLATE": URL template for GET requests. Must contain one and only one occurrence of %s in it. If not specified, for SERVICE=OSM\_NOMINATIM, MAPQUEST\_NOMINATIM, YAHOO, GEONAMES or BING, the URL template is hard-coded.
- "REVERSE\_QUERY\_TEMPLATE": URL template for GET requests for reverse geocoding. Must contain one and only one occurrence of {lon} and {lat} in it. If not specified, for SERVICE=OSM\_NOMINATIM, MAPQUEST\_NOMINATIM, YAHOO, GEONAMES or BING, the URL template is hard-coded.

All the above options can also be set by defining the configuration option of the same name, prefixed by **OGR\_GEOCODE\_**. For example "OGR\_GEOCODE\_SERVICE" for the "SERVICE" option.

## Parameters

<i>papszOptions</i>	NULL, or a NULL-terminated list of string options.
---------------------	--

## Returns

an handle that should be freed with **OGRGeocodeDestroySession()** (p. ??), or NULL in case of failure.

## Since

GDAL 1.10

References CPLCalloc(), CPLError(), CPLGetExtension(), EQUAL, and STARTS\_WITH\_CI.

## 12.19.3.3 OGRGeocodeDestroySession()

```
void OGRGeocodeDestroySession (
    OGRGeocodingSessionH hSession )
```

Destroys a session handle for geocoding requests.

## Parameters

<i>hSession</i>	the handle to destroy.
-----------------	------------------------

## Since

GDAL 1.10

References CPLFree, and OGRReleaseDataSource().

## 12.19.3.4 OGRGeocodeFreeResult()

```
void OGRGeocodeFreeResult (
    OGRLayerH hLayer )
```

Destroys the result of a geocoding request.

## Parameters

<i>hLayer</i>	the layer returned by <b>OGRGeocode()</b> (p. ??) or <b>OGRGeocodeReverse()</b> (p. ??) to destroy.
---------------	---

Since

GDAL 1.10

### 12.19.3.5 OGRGeocodeReverse()

```
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**: additional parameters for the GET request for reverse geocoding.

#### Parameters

<i>hSession</i>	the geocoding session handle.
<i>dfLon</i>	the longitude.
<i>dfLat</i>	the latitude.
<i>papszOptions</i>	a list of options or NULL.

#### Returns

a OGR layer with the result(s), or NULL in case of error. The returned layer must be freed with **OGR↵ GeocodeFreeResult()** (p. ??).

Since

GDAL 1.10

References `CPL`Error(), and `VALIDATE_POINTER1`.

## 12.20 ogr\_geometry.h File Reference

```
#include "cpl_conv.h"
#include "cpl_json.h"
#include "ogr_core.h"
#include "ogr_spatialref.h"
#include <memory>
```

### Classes

- class **OGRRawPoint**
- class **IOGRGeometryVisitor**
- class **OGRDefaultGeometryVisitor**
- class **IOGRConstGeometryVisitor**
- class **OGRDefaultConstGeometryVisitor**
- class **OGRGeometry**
- class **OGRPoint**
- class **OGRPointIterator**
- class **OGRCurve**
- class **OGRSimpleCurve**
- class **OGRLineString**
- class **OGRLinearRing**
- class **OGRCircularString**
- class **OGRCompoundCurve**
- class **OGRSurface**
- class **OGRCurvePolygon**
- class **OGRPolygon**
- class **OGRTriangle**
- class **OGRGeometryCollection**
- class **OGRMultiSurface**
- class **OGRMultiPolygon**
- class **OGRPolyhedralSurface**
- class **OGRTriangulatedSurface**
- class **OGRMultiPoint**
- class **OGRMultiCurve**
- class **OGRMultiLineString**
- class **OGRGeometryFactory**

### Typedefs

- typedef struct GEOSGeom\_t \* **GEOSGeom**
- typedef struct GEOSContextHandle\_HS \* **GEOSContextHandle\_t**
- typedef void **sfcgal\_geometry\_t**
- typedef std::unique\_ptr< **OGRGeometry**, OGRGeometryUniquePtrDeleter > **OGRGeometryUniquePtr**
- typedef struct \_OGRPreparedGeometry **OGRPreparedGeometry**
- typedef std::unique\_ptr< **OGRPreparedGeometry**, OGRPreparedGeometryUniquePtrDeleter > **OGRPreparedGeometryUniquePtr**



## Functions

- **OGRwkbGeometryType** **OGRFromOGCGeomType** (const char \*pszGeomType)
- const char \* **OGRToOGCGeomType** ( **OGRwkbGeometryType** eGeomType)
- int **OGRHasPreparedGeometrySupport** ()

### 12.20.1 Detailed Description

Simple feature geometry classes.

### 12.20.2 Typedef Documentation

#### 12.20.2.1 GEOSContextHandle\_t

```
typedef struct GEOSContextHandle_HS* GEOSContextHandle_t
```

GEOS context handle type

#### 12.20.2.2 GEOSGeom

```
typedef struct GEOSGeom_t* GEOSGeom
```

GEOS geometry type

#### 12.20.2.3 OGRGeometryUniquePtr

```
typedef std::unique_ptr< OGRGeometry, OGRGeometryUniquePtrDeleter> OGRGeometryUniquePtr
```

Unique pointer type for **OGRGeometry** (p. ??).

Since

GDAL 2.3

#### 12.20.2.4 OGRPreparedGeometry

```
typedef struct _OGRPreparedGeometry OGRPreparedGeometry
```

Prepared geometry API (needs GEOS >= 3.1.0)

### 12.20.2.5 OGRPreparedGeometryUniquePtr

```
typedef std::unique_ptr< OGRPreparedGeometry, OGRPreparedGeometryUniquePtrDeleter> OGRPreparedGeometryUniquePtr
```

Unique pointer type for OGRPreparedGeometry.

Since

GDAL 2.3

### 12.20.2.6 sfcgal\_geometry\_t

```
typedef void sfcgal_geometry_t
```

SFCGAL geometry type

## 12.20.3 Function Documentation

### 12.20.3.1 OGRFromOGCGeomType()

```
OGRwkbGeometryType OGRFromOGCGeomType (
    const char * pszGeomType )
```

Map OGCgeometry format type to corresponding OGR constants.

Parameters

<i>pszGeomType</i>	POINT[ ][Z][M], LINESTRING[ ][Z][M], etc...
--------------------	---

Returns

OGR constant.

References STARTS\_WITH\_CI, wkbCircularString, wkbCompoundCurve, wkbCurve, wkbCurvePolygon, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbPoint, wkbPolygon, wkbPolyhedralSurface, wkbSetM, wkbSetZ, wkbSurface, wkbTIN, wkbTriangle, and wkbUnknown.

### 12.20.3.2 OGRHasPreparedGeometrySupport()

```
int OGRHasPreparedGeometrySupport ( )
```

Returns if GEOS has prepared geometry support.

**Returns**

TRUE or FALSE

Referenced by OGRLayer::Identity(), OGRLayer::Intersection(), and OGRLayer::Union().

**12.20.3.3 OGRToOGCGeomType()**

```
const char* OGRToOGCGeomType (
    OGRwkbGeometryType eGeomType )
```

Map OGR geometry format constants to corresponding OGC geometry type.

**Parameters**

<i>eGeomType</i>	OGR geometry type
------------------	-------------------

**Returns**

string with OGC geometry type (without dimensionality)

References wkbCircularString, wkbCompoundCurve, wkbCurve, wkbCurvePolygon, wkbFlatten, wkbGeometryCollection, wkbLineString, wkbMultiCurve, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbMultiSurface, wkbPoint, wkbPolygon, wkbPolyhedralSurface, wkbSurface, wkbTIN, wkbTriangle, and wkbUnknown.

**12.21 ogr\_spatialref.h File Reference**

```
#include "cpl_string.h"
#include "ogr_srs_api.h"
#include <cstddef>
#include <map>
#include <vector>
```

**Classes**

- class **OGR\_SRSNode**
- class **OGRSpatialReference**
- class **OGRCoordinateTransformation**

**Macros**

- #define **USGS\_ANGLE\_DECIMALDEGREES** 0
- #define **USGS\_ANGLE\_PACKEDDMS** TRUE
- #define **USGS\_ANGLE\_RADIANS** 2

## Functions

- **OGRCoordinateTransformation** \* **OGRCreateCoordinateTransformation** ( **OGRSpatialReference** \*poSource, **OGRSpatialReference** \*poTarget)

### 12.21.1 Detailed Description

Coordinate systems services.

### 12.21.2 Macro Definition Documentation

#### 12.21.2.1 USGS\_ANGLE\_DECIMALDEGREES

```
#define USGS_ANGLE_DECIMALDEGREES 0
```

Angle is in decimal degrees.

Referenced by OGRSpatialReference::importFromUSGS().

#### 12.21.2.2 USGS\_ANGLE\_PACKEDDMS

```
#define USGS_ANGLE_PACKEDDMS TRUE
```

Angle is in packed degree minute second.

#### 12.21.2.3 USGS\_ANGLE\_RADIANS

```
#define USGS_ANGLE_RADIANS 2
```

Angle is in radians.

Referenced by OGRSpatialReference::importFromUSGS().

### 12.21.3 Function Documentation

#### 12.21.3.1 OGRCreateCoordinateTransformation()

```
OGRCoordinateTransformation* OGRCreateCoordinateTransformation (
    OGRSpatialReference * poSource,
    OGRSpatialReference * poTarget )
```

Create transformation object.

This is the same as the C function **OCTNewCoordinateTransformation()** (p. ??).

Input spatial reference system objects are assigned by copy (calling clone() method) and no ownership transfer occurs.

The delete operator, or **OCTDestroyCoordinateTransformation()** (p. ??) should be used to destroy transformation objects.

The PROJ.4 library must be available at run-time.

## Parameters

<i>poSource</i>	source spatial reference system.
<i>poTarget</i>	target spatial reference system.

## Returns

NULL on failure or a ready to use transformation object.

References CPLError().

Referenced by OCTNewCoordinateTransformation(), and OGRGeometryFactory::transformWithOptions().

## 12.22 ogr\_srs\_api.h File Reference

```
#include "ogr_core.h"
```

## Macros

- #define **SRS\_WKT\_WGS84** "GEOGCS[\"WGS 84\", DATUM[\"WGS\_1984\", SPHEROID[\"WGS 84\", 6378137, 298.↵  
257223563, AUTHORITY[\"EPSG\", \"7030\"], AUTHORITY[\"EPSG\", \"6326\"], PRIMEM[\"Greenwich\", 0, A↵  
UTHORITY[\"EPSG\", \"8901\"], UNIT[\"degree\", 0.0174532925199433, AUTHORITY[\"EPSG\", \"9122\"], A↵  
UTHORITY[\"EPSG\", \"4326\"]]"
- #define **SRS\_PT\_ALBERS\_CONIC\_EQUAL\_AREA** "Albers\_Conic\_Equal\_Area"
- #define **SRS\_PT\_AZIMUTHAL\_EQUIDISTANT** "Azimuthal\_Equidistant"
- #define **SRS\_PT\_CASSINI\_SOLDNER** "Cassini\_Soldner"
- #define **SRS\_PT\_CYLINDRICAL\_EQUAL\_AREA** "Cylindrical\_Equal\_Area"
- #define **SRS\_PT\_BONNE** "Bonne"
- #define **SRS\_PT\_ECKERT\_I** "Eckert\_I"
- #define **SRS\_PT\_ECKERT\_II** "Eckert\_II"
- #define **SRS\_PT\_ECKERT\_III** "Eckert\_III"
- #define **SRS\_PT\_ECKERT\_IV** "Eckert\_IV"
- #define **SRS\_PT\_ECKERT\_V** "Eckert\_V"
- #define **SRS\_PT\_ECKERT\_VI** "Eckert\_VI"
- #define **SRS\_PT\_EQUIDISTANT\_CONIC** "Equidistant\_Conic"
- #define **SRS\_PT\_EQUIRECTANGULAR** "Equirectangular"
- #define **SRS\_PT\_GALL\_STEREOGRAPHIC** "Gall\_Stereographic"
- #define **SRS\_PT\_GAUSSSCHREIBERTMERCATOR** "Gauss\_Schreiber\_Transverse\_Mercator"
- #define **SRS\_PT\_GEOSTATIONARY\_SATELLITE** "Geostationary\_Satellite"
- #define **SRS\_PT\_GOODE\_HOMOLOGINE** "Goode\_Homolosine"
- #define **SRS\_PT\_IGH** "Interrupted\_Goode\_Homolosine"
- #define **SRS\_PT\_GNOMONIC** "Gnomonic"
- #define **SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR\_AZIMUTH\_CENTER** "Hotine\_Oblique\_Mercator\_↵  
Azimuth\_Center"
- #define **SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR** "Hotine\_Oblique\_Mercator"
- #define **SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR\_TWO\_POINT\_NATURAL\_ORIGIN** "Hotine\_↵  
Oblique\_Mercator\_Two\_Point\_Natural-Origin"
- #define **SRS\_PT\_LABORDE\_OBLIQUE\_MERCATOR** "Laborde\_Oblique\_Mercator"
- #define **SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP** "Lambert\_Conformal\_Conic\_1SP"

- #define **SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP** "Lambert\_Conformal\_Conic\_2SP"
- #define **SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP\_BELGIUM** "Lambert\_Conformal\_Conic\_2SP↔\_Belgium"
- #define **SRS\_PT\_LAMBERT\_AZIMUTHAL\_EQUAL\_AREA** "Lambert\_Azimuthal\_Equal\_Area"
- #define **SRS\_PT\_MERCATOR\_1SP** "Mercator\_1SP"
- #define **SRS\_PT\_MERCATOR\_2SP** "Mercator\_2SP"
- #define **SRS\_PT\_MERCATOR\_AUXILIARY\_SPHERE** "Mercator\_Auxiliary\_Sphere"
- #define **SRS\_PT\_MILLER\_CYLINDRICAL** "Miller\_Cylindrical"
- #define **SRS\_PT\_MOLLWEIDE** "Mollweide"
- #define **SRS\_PT\_NEW\_ZEALAND\_MAP\_GRID** "New\_Zealand\_Map\_Grid"
- #define **SRS\_PT\_OBLIQUE\_STEREOGRAPHIC** "Oblique\_Stereographic"
- #define **SRS\_PT\_ORTHOGRAPHIC** "Orthographic"
- #define **SRS\_PT\_POLAR\_STEREOGRAPHIC** "Polar\_Stereographic"
- #define **SRS\_PT\_POLYCONIC** "Polyconic"
- #define **SRS\_PT\_ROBINSON** "Robinson"
- #define **SRS\_PT\_SINUSOIDAL** "Sinusoidal"
- #define **SRS\_PT\_STEREOGRAPHIC** "Stereographic"
- #define **SRS\_PT\_SWISS\_OBLIQUE\_CYLINDRICAL** "Swiss\_Oblique\_Cylindrical"
- #define **SRS\_PT\_TRANSVERSE\_MERCATOR** "Transverse\_Mercator"
- #define **SRS\_PT\_TRANSVERSE\_MERCATOR\_SOUTH\_ORIENTED** "Transverse\_Mercator\_South\_↔Orientated"
- #define **SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_21** "Transverse\_Mercator\_MapInfo\_21"
- #define **SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_22** "Transverse\_Mercator\_MapInfo\_22"
- #define **SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_23** "Transverse\_Mercator\_MapInfo\_23"
- #define **SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_24** "Transverse\_Mercator\_MapInfo\_24"
- #define **SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_25** "Transverse\_Mercator\_MapInfo\_25"
- #define **SRS\_PT\_TUNISIA\_MINING\_GRID** "Tunisia\_Mining\_Grid"
- #define **SRS\_PT\_TWO\_POINT\_EQUIDISTANT** "Two\_Point\_Equidistant"
- #define **SRS\_PT\_VANDERGRINTEN** "VanDerGrinten"
- #define **SRS\_PT\_KROVAK** "Krovak"
- #define **SRS\_PT\_IMW\_POLYCONIC** "International\_Map\_of\_the\_World\_Polyconic"
- #define **SRS\_PT\_WAGNER\_I** "Wagner\_I"
- #define **SRS\_PT\_WAGNER\_II** "Wagner\_II"
- #define **SRS\_PT\_WAGNER\_III** "Wagner\_III"
- #define **SRS\_PT\_WAGNER\_IV** "Wagner\_IV"
- #define **SRS\_PT\_WAGNER\_V** "Wagner\_V"
- #define **SRS\_PT\_WAGNER\_VI** "Wagner\_VI"
- #define **SRS\_PT\_WAGNER\_VII** "Wagner\_VII"
- #define **SRS\_PT\_QSC** "Quadrilateralized\_Spherical\_Cube"
- #define **SRS\_PT\_AITOFF** "Aitoff"
- #define **SRS\_PT\_WINKEL\_I** "Winkel\_I"
- #define **SRS\_PT\_WINKEL\_II** "Winkel\_II"
- #define **SRS\_PT\_WINKEL\_TRIPEL** "Winkel\_Tripel"
- #define **SRS\_PT\_CRASTER\_PARABOLIC** "Craster\_Parabolic"
- #define **SRS\_PT\_LOXIMUTHAL** "Loximuthal"
- #define **SRS\_PT\_QUARTIC\_AUTHALIC** "Quartic\_Authalic"
- #define **SRS\_PT\_SCH** "Spherical\_Cross\_Track\_Height"
- #define **SRS\_PP\_CENTRAL\_MERIDIAN** "central\_meridian"
- #define **SRS\_PP\_SCALE\_FACTOR** "scale\_factor"
- #define **SRS\_PP\_STANDARD\_PARALLEL\_1** "standard\_parallel\_1"
- #define **SRS\_PP\_STANDARD\_PARALLEL\_2** "standard\_parallel\_2"
- #define **SRS\_PP\_PSEUDO\_STD\_PARALLEL\_1** "pseudo\_standard\_parallel\_1"
- #define **SRS\_PP\_LONGITUDE\_OF\_CENTER** "longitude\_of\_center"
- #define **SRS\_PP\_LATITUDE\_OF\_CENTER** "latitude\_of\_center"
- #define **SRS\_PP\_LONGITUDE\_OF\_ORIGIN** "longitude\_of\_origin"

- #define SRS\_PP\_LATITUDE\_OF\_ORIGIN "latitude\_of\_origin"
- #define SRS\_PP\_FALSE\_EASTING "false\_easting"
- #define SRS\_PP\_FALSE\_NORTHING "false\_northing"
- #define SRS\_PP\_AZIMUTH "azimuth"
- #define SRS\_PP\_LONGITUDE\_OF\_POINT\_1 "longitude\_of\_point\_1"
- #define SRS\_PP\_LATITUDE\_OF\_POINT\_1 "latitude\_of\_point\_1"
- #define SRS\_PP\_LONGITUDE\_OF\_POINT\_2 "longitude\_of\_point\_2"
- #define SRS\_PP\_LATITUDE\_OF\_POINT\_2 "latitude\_of\_point\_2"
- #define SRS\_PP\_LONGITUDE\_OF\_POINT\_3 "longitude\_of\_point\_3"
- #define SRS\_PP\_LATITUDE\_OF\_POINT\_3 "latitude\_of\_point\_3"
- #define SRS\_PP\_RECTIFIED\_GRID\_ANGLE "rectified\_grid\_angle"
- #define SRS\_PP\_LANDSAT\_NUMBER "landsat\_number"
- #define SRS\_PP\_PATH\_NUMBER "path\_number"
- #define SRS\_PP\_PERSPECTIVE\_POINT\_HEIGHT "perspective\_point\_height"
- #define SRS\_PP\_SATELLITE\_HEIGHT "satellite\_height"
- #define SRS\_PP\_FIPZONE "fipszone"
- #define SRS\_PP\_ZONE "zone"
- #define SRS\_PP\_LATITUDE\_OF\_1ST\_POINT "Latitude\_Of\_1st\_Point"
- #define SRS\_PP\_LONGITUDE\_OF\_1ST\_POINT "Longitude\_Of\_1st\_Point"
- #define SRS\_PP\_LATITUDE\_OF\_2ND\_POINT "Latitude\_Of\_2nd\_Point"
- #define SRS\_PP\_LONGITUDE\_OF\_2ND\_POINT "Longitude\_Of\_2nd\_Point"
- #define SRS\_PP\_PEG\_POINT\_LATITUDE "peg\_point\_latitude"
- #define SRS\_PP\_PEG\_POINT\_LONGITUDE "peg\_point\_longitude"
- #define SRS\_PP\_PEG\_POINT\_HEADING "peg\_point\_heading"
- #define SRS\_PP\_PEG\_POINT\_HEIGHT "peg\_point\_height"
- #define SRS\_UL\_METER "Meter"
- #define SRS\_UL\_FOOT "Foot (International)" /\* or just "FOOT"? \*/
- #define SRS\_UL\_FOOT\_CONV "0.3048"
- #define SRS\_UL\_US\_FOOT "Foot\_US" /\* or "US survey foot" from EPSG \*/
- #define SRS\_UL\_US\_FOOT\_CONV "0.3048006096012192"
- #define SRS\_UL\_NAUTICAL\_MILE "Nautical Mile"
- #define SRS\_UL\_NAUTICAL\_MILE\_CONV "1852.0"
- #define SRS\_UL\_LINK "Link" /\* Based on US Foot \*/
- #define SRS\_UL\_LINK\_CONV "0.20116684023368047"
- #define SRS\_UL\_CHAIN "Chain" /\* based on US Foot \*/
- #define SRS\_UL\_CHAIN\_CONV "20.116684023368047"
- #define SRS\_UL\_ROD "Rod" /\* based on US Foot \*/
- #define SRS\_UL\_ROD\_CONV "5.02921005842012"
- #define SRS\_UL\_LINK\_Clarke "Link\_Clarke"
- #define SRS\_UL\_LINK\_Clarke\_CONV "0.2011661949"
- #define SRS\_UL\_KILOMETER "Kilometer"
- #define SRS\_UL\_KILOMETER\_CONV "1000."
- #define SRS\_UL\_DECIMETER "Decimeter"
- #define SRS\_UL\_DECIMETER\_CONV "0.1"
- #define SRS\_UL\_CENTIMETER "Centimeter"
- #define SRS\_UL\_CENTIMETER\_CONV "0.01"
- #define SRS\_UL\_MILLIMETER "Millimeter"
- #define SRS\_UL\_MILLIMETER\_CONV "0.001"
- #define SRS\_UL\_INTL\_NAUT\_MILE "Nautical\_Mile\_International"
- #define SRS\_UL\_INTL\_NAUT\_MILE\_CONV "1852.0"
- #define SRS\_UL\_INTL\_INCH "Inch\_International"
- #define SRS\_UL\_INTL\_INCH\_CONV "0.0254"
- #define SRS\_UL\_INTL\_FOOT "Foot\_International"
- #define SRS\_UL\_INTL\_FOOT\_CONV "0.3048"
- #define SRS\_UL\_INTL\_YARD "Yard\_International"

- `#define SRS_UL_INTL_YARD_CONV "0.9144"`
- `#define SRS_UL_INTL_STAT_MILE "Statute_Mile_International"`
- `#define SRS_UL_INTL_STAT_MILE_CONV "1609.344"`
- `#define SRS_UL_INTL_FATHOM "Fathom_International"`
- `#define SRS_UL_INTL_FATHOM_CONV "1.8288"`
- `#define SRS_UL_INTL_CHAIN "Chain_International"`
- `#define SRS_UL_INTL_CHAIN_CONV "20.1168"`
- `#define SRS_UL_INTL_LINK "Link_International"`
- `#define SRS_UL_INTL_LINK_CONV "0.201168"`
- `#define SRS_UL_US_INCH "Inch_US_Surveyor"`
- `#define SRS_UL_US_INCH_CONV "0.025400050800101603"`
- `#define SRS_UL_US_YARD "Yard_US_Surveyor"`
- `#define SRS_UL_US_YARD_CONV "0.914401828803658"`
- `#define SRS_UL_US_CHAIN "Chain_US_Surveyor"`
- `#define SRS_UL_US_CHAIN_CONV "20.11684023368047"`
- `#define SRS_UL_US_STAT_MILE "Statute_Mile_US_Surveyor"`
- `#define SRS_UL_US_STAT_MILE_CONV "1609.347218694437"`
- `#define SRS_UL_INDIAN_YARD "Yard_Indian"`
- `#define SRS_UL_INDIAN_YARD_CONV "0.91439523"`
- `#define SRS_UL_INDIAN_FOOT "Foot_Indian"`
- `#define SRS_UL_INDIAN_FOOT_CONV "0.30479841"`
- `#define SRS_UL_INDIAN_CHAIN "Chain_Indian"`
- `#define SRS_UL_INDIAN_CHAIN_CONV "20.11669506"`
- `#define SRS_UA_DEGREE "degree"`
- `#define SRS_UA_DEGREE_CONV "0.0174532925199433"`
- `#define SRS_UA_RADIAN "radian"`
- `#define SRS_PM_GREENWICH "Greenwich"`
- `#define SRS_DN_NAD27 "North_American_Datum_1927"`
- `#define SRS_DN_NAD83 "North_American_Datum_1983"`
- `#define SRS_DN_WGS72 "WGS_1972"`
- `#define SRS_DN_WGS84 "WGS_1984"`
- `#define SRS_WGS84_SEMIMAJOR 6378137.0`
- `#define SRS_WGS84_INVFLATTENING 298.257223563`

## Typedefs

- `typedef void * OGRSpatialReferenceH`
- `typedef void * OGRCoordinateTransformationH`

## Enumerations

- `enum OGRAxisOrientation {`  
`OAO_Other =0, OAO_North =1, OAO_South =2, OAO_East =3,`  
`OAO_West =4, OAO_Up =5, OAO_Down =6 }`



## 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** **OSRImportFromDict** ( **OGRSpatialReferenceH**, const char \*, const char \*)
- **OGRErr** **OSRImportFromPanorama** ( **OGRSpatialReferenceH**, long, long, long, double \*)
- **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 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 OSRExportToPanorama ( OGRSpatialReferenceH, long \*, long \*, long \*, long \*, double \*)**
- **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.*
- **OGRSpatialReferenceH OSRConvertToOtherProjection ( OGRSpatialReferenceH hSRS, const char \*pszTargetProjection, const char \*const \*papszOptions)**  
*Convert to another equivalent projection.*
- **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, const **OGRSpatialReferenceH** hSrcSRS)

*Copy GEOGCS from another **OGRSpatialReference** (p. ??).*
- **OGRERR** **OSRSetTOWGS84** ( **OGRSpatialReferenceH** hSRS, double, double, double, double, double, double, double)

*Set the Bursa-Wolf conversion to WGS84.*
- **OGRERR** **OSRGetTOWGS84** ( **OGRSpatialReferenceH** hSRS, double \*, int)

*Fetch TOWGS84 parameters, if available.*
- **OGRERR** **OSRSetCompoundCS** ( **OGRSpatialReferenceH** hSRS, const char \*pszName, **OGRSpatialReferenceH** hHorizSRS, **OGRSpatialReferenceH** hVertSRS)

*Setup a compound coordinate system.*
- **OGRERR** **OSRSetGeogCS** ( **OGRSpatialReferenceH** hSRS, const char \*pszGeogName, const char \*pszDatumName, const char \*pszEllipsoidName, double dfSemiMajor, double dfInvFlattening, const char \*pszPMName, double dfPMOffset, const char \*pszUnits, double dfConvertToRadians)

*Set geographic coordinate system.*
- **OGRERR** **OSRSetVertCS** ( **OGRSpatialReferenceH** hSRS, const char \*pszVertCSName, const char \*pszVertDatumName, int nVertDatumType)

*Setup the vertical coordinate system.*
- double **OSRGetSemiMajor** ( **OGRSpatialReferenceH**, **OGRERR** \*)

*Get spheroid semi major axis.*
- double **OSRGetSemiMinor** ( **OGRSpatialReferenceH**, **OGRERR** \*)

*Get spheroid semi minor axis.*
- double **OSRGetInvFlattening** ( **OGRSpatialReferenceH**, **OGRERR** \*)

*Get spheroid inverse flattening.*
- **OGRERR** **OSRSetAuthority** ( **OGRSpatialReferenceH** hSRS, const char \*pszTargetKey, const char \*pszAuthority, int nCode)

*Set the authority for a node.*
- const char \* **OSRGetAuthorityCode** ( **OGRSpatialReferenceH** hSRS, const char \*pszTargetKey)

*Get the authority code for a node.*

- **const char \* OSRGetAuthorityName ( OGRSpatialReferenceH hSRS, const char \*pszTargetKey)**  
*Get the authority name for a node.*
- **OGRERR OSRSetProjection ( OGRSpatialReferenceH, const char \*)**  
*Set a projection name.*
- **OGRERR OSRSetProjParm ( OGRSpatialReferenceH, const char \*, double)**  
*Set a projection parameter value.*
- **double OSRGetProjParm ( OGRSpatialReferenceH hSRS, const char \*pszParmName, double dfDefault, OGRERR \*)**  
*Fetch a projection parameter value.*
- **OGRERR OSRSetNormProjParm ( OGRSpatialReferenceH, const char \*, double)**  
*Set a projection parameter with a normalized value.*
- **double OSRGetNormProjParm ( OGRSpatialReferenceH hSRS, const char \*pszParmName, double dfDefault, OGRERR \*)**  
*This function is the same as OGRSpatialReference (p. ??)::*
- **OGRERR OSRSetUTM ( OGRSpatialReferenceH hSRS, int nZone, int bNorth)**  
*Set UTM projection definition.*
- **int OSRGetUTMZone ( OGRSpatialReferenceH hSRS, int \*pbNorth)**  
*Get utm zone information.*
- **OGRERR OSRSetStatePlane ( OGRSpatialReferenceH hSRS, int nZone, int bNAD83)**  
*Set State Plane projection definition.*
- **OGRERR OSRSetStatePlaneWithUnits ( OGRSpatialReferenceH hSRS, int nZone, int bNAD83, const char \*pszOverrideUnitName, double dfOverrideUnit)**  
*Set State Plane projection definition.*
- **OGRERR OSRAutoIdentifyEPSG ( OGRSpatialReferenceH hSRS)**  
*Set EPSG authority info if possible.*
- **OGRSpatialReferenceH \* OSRFindMatches ( OGRSpatialReferenceH hSRS, char \*\*papszOptions, int \*pnEntries, int \*\*ppanMatchConfidence)**  
*Try to identify a match between the passed SRS and a related SRS in a catalog (currently EPSG only)*
- **void OSRFreeSRSArray ( OGRSpatialReferenceH \*pahSRS)**  
*Free return of OSRIdentifyMatches()*
- **int OSREPSGTreatsAsLatLong ( OGRSpatialReferenceH hSRS)**  
*This function returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.*
- **int OSREPSGTreatsAsNorthingEasting ( OGRSpatialReferenceH hSRS)**  
*This function returns TRUE if EPSG feels this geographic coordinate system should be treated as having northing/easting coordinate ordering.*
- **const char \* OSRGetAxis ( OGRSpatialReferenceH hSRS, const char \*pszTargetKey, int iAxis, OGRAxisOrientation \*peOrientation)**  
*Fetch the orientation of one axis.*
- **OGRERR OSRSetAxes ( OGRSpatialReferenceH hSRS, const char \*pszTargetKey, const char \*pszXAxisName, OGRAxisOrientation eXAxisOrientation, const char \*pszYAxisName, OGRAxisOrientation eYAxisOrientation)**  
*Set the axes for a coordinate system.*
- **OGRERR OSRSetACEA ( OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)**
- **OGRERR OSRSetAE ( OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)**
- **OGRERR OSRSetBonne ( OGRSpatialReferenceH hSRS, double dfStandardParallel, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)**
- **OGRERR OSRSetCEA ( OGRSpatialReferenceH hSRS, double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)**
- **OGRERR OSRSetCS ( OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)**

- **OGRErr OSRSetEC** ( **OGRSpatialReferenceH** hSRS, double dfStdP1, double dfStdP2, double dfCenter↵ Lat, 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 dfFalse↵ Easting, double dfFalseNorthing)
- **OGRErr OSRSetEckertVI** ( **OGRSpatialReferenceH** hSRS, double dfCentralMeridian, double dfFalse↵ Easting, double dfFalseNorthing)
- **OGRErr OSRSetEquirectangular** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenter↵ Long, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr OSRSetEquirectangular2** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double df↵ CenterLong, 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 dfSatellite↵ Height, 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 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 OSRSetHOMAC** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)  
*Set an Oblique Mercator projection using azimuth angle.*
- **OGRErr OSRSetHOM2PNO** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing)  
*Set a Hotine Oblique Mercator projection using two points on projection centerline.*
- **OGRErr OSRSetIWMPolyconic** ( **OGRSpatialReferenceH** hSRS, double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr OSRSetKrovak** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalse↵ Northing)
- **OGRErr OSRSetLAEA** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr OSRSetLCC** ( **OGRSpatialReferenceH** hSRS, double dfStdP1, double dfStdP2, double df↵ CenterLat, 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 df↵ CenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr OSRSetMC** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr OSRSetMercator** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRErr OSRSetMercator2SP** ( **OGRSpatialReferenceH** hSRS, double dfStdP1, double dfCenterLat, double dfCenterLong, 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 dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetOrthographic** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetPolyconic** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetPS** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetRobinson** ( **OGRSpatialReferenceH** hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetSinusoidal** ( **OGRSpatialReferenceH** hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetStereographic** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetSOC** ( **OGRSpatialReferenceH** hSRS, double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetTM** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetTMVariant** ( **OGRSpatialReferenceH** hSRS, const char \*pszVariantName, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetTMG** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetTMSO** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetTPED** ( **OGRSpatialReferenceH** hSRS, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetVDG** ( **OGRSpatialReferenceH** hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetWagner** ( **OGRSpatialReferenceH** hSRS, int nVariation, double dfCenterLat, double dfFalseEasting, double dfFalseNorthing)
- **OGRERR OSRSetQSC** ( **OGRSpatialReferenceH** hSRS, double dfCenterLat, double dfCenterLong)
- **OGRERR OSRSetSCH** ( **OGRSpatialReferenceH** hSRS, double dfPegLat, double dfPegLong, double dfPegHeading, double dfPegHgt)
- 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** ( **OGRSpatialReferenceH** hSourceSRS, **OGRSpatialReferenceH** hTargetSRS)
- void CPL\_STDCALL **OCTDestroyCoordinateTransformation** ( **OGRCoordinateTransformationH**)  
*OGRCoordinateTransformation (p. ??) destructor.*
- int CPL\_STDCALL **OCTTransform** ( **OGRCoordinateTransformationH** hCT, int nCount, double \*x, double \*y, double \*z)
- int CPL\_STDCALL **OCTTransformEx** ( **OGRCoordinateTransformationH** hCT, int nCount, double \*x, double \*y, double \*z, int \*pabSuccess)
- char \*\* **OPTGetProjectionMethods** (void)
- char \*\* **OPTGetParameterList** (const char \*pszProjectionMethod, char \*\*ppszUserName)

### 12.22.1 Detailed Description

C spatial reference system services and defines.

See also: **ogr\_spatialref.h** (p. ??)

## 12.22.2 Macro Definition Documentation

### 12.22.2.1 SRS\_DN\_NAD27

```
#define SRS_DN_NAD27 "North_American_Datum_1927"
```

North\_American\_Datum\_1927 datum name

Referenced by `OGRSpatialReference::exportToPCI()`, and `OGRSpatialReference::exportToUSGS()`.

### 12.22.2.2 SRS\_DN\_NAD83

```
#define SRS_DN_NAD83 "North_American_Datum_1983"
```

North\_American\_Datum\_1983 datum name

Referenced by `OGRSpatialReference::exportToPCI()`, and `OGRSpatialReference::exportToUSGS()`.

### 12.22.2.3 SRS\_DN\_WGS72

```
#define SRS_DN_WGS72 "WGS_1972"
```

WGS\_1972 datum name

### 12.22.2.4 SRS\_DN\_WGS84

```
#define SRS_DN_WGS84 "WGS_1984"
```

WGS\_1984 datum name

Referenced by `OGRSpatialReference::exportToPanorama()`, `OGRSpatialReference::exportToPCI()`, and `OGRSpatialReference::exportToUSGS()`.

### 12.22.2.5 SRS\_PM\_GREENWICH

```
#define SRS_PM_GREENWICH "Greenwich"
```

Prime meridian Greenwich

Referenced by `OGRSpatialReference::GetPrimeMeridian()`, and `OGRSpatialReference::SetGeogCS()`.

### 12.22.2.6 SRS\_PP\_AZIMUTH

```
#define SRS_PP_AZIMUTH "azimuth"
```

azimuth projection parameter

Referenced by `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, `OGRSpatialReference::IsAngularParameter()`, `OGRSpatialReference::IsSame()`, `OGRSpatialReference::morphFromESRI()`, `OGRSpatialReference::morphToESRI()`, `OGRSpatialReference::SetHOM()`, `OGRSpatialReference::SetHOMAC()`, and `OGRSpatialReference::SetKrovak()`.

### 12.22.2.7 SRS\_PP\_CENTRAL\_MERIDIAN

```
#define SRS_PP_CENTRAL_MERIDIAN "central_meridian"
```

central\_meridian projection parameter

Referenced by `OGRSpatialReference::AutoIdentifyEPSG()`, `OGRSpatialReference::convertToOtherProjection()`, `OGRSpatialReference::exportToPanorama()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, `OGRSpatialReference::FindProjParm()`, `OGRSpatialReference::GetUTMZone()`, `OGRSpatialReference::IsAngularParameter()`, `OGRSpatialReference::IsLongitudeParameter()`, `OGRSpatialReference::SetBonne()`, `OGRSpatialReference::SetCEA()`, `OGRSpatialReference::SetCS()`, `OGRSpatialReference::SetEckertIV()`, `OGRSpatialReference::SetEckertVI()`, `OGRSpatialReference::SetEquirectangular()`, `OGRSpatialReference::SetEquirectangular2()`, `OGRSpatialReference::SetGaussSchreiberTMercator()`, `OGRSpatialReference::SetGEOS()`, `OGRSpatialReference::SetGH()`, `OGRSpatialReference::SetGnomonic()`, `OGRSpatialReference::SetGS()`, `OGRSpatialReference::SetIWMPolyconic()`, `OGRSpatialReference::SetLCC()`, `OGRSpatialReference::SetLCC1SP()`, `OGRSpatialReference::SetLCCB()`, `OGRSpatialReference::SetMercator()`, `OGRSpatialReference::SetMercator2SP()`, `OGRSpatialReference::SetMollweide()`, `OGRSpatialReference::SetNZMG()`, `OGRSpatialReference::SetOrthographic()`, `OGRSpatialReference::SetOS()`, `OGRSpatialReference::SetPolyconic()`, `OGRSpatialReference::SetPS()`, `OGRSpatialReference::SetQSC()`, `OGRSpatialReference::SetSOC()`, `OGRSpatialReference::SetStereographic()`, `OGRSpatialReference::SetTM()`, `OGRSpatialReference::SetTMG()`, `OGRSpatialReference::SetTMSO()`, `OGRSpatialReference::SetTMVariant()`, and `OGRSpatialReference::SetVDG()`.

### 12.22.2.8 SRS\_PP\_FALSE\_EASTING

```
#define SRS_PP_FALSE_EASTING "false_easting"
```

false\_easting projection parameter

Referenced by `OGRSpatialReference::AutoIdentifyEPSG()`, `OGRSpatialReference::convertToOtherProjection()`, `OGRSpatialReference::exportToPanorama()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, `OGRSpatialReference::GetUTMZone()`, `OGRSpatialReference::SetACEA()`, `OGRSpatialReference::SetAE()`, `OGRSpatialReference::SetBonne()`, `OGRSpatialReference::SetCEA()`, `OGRSpatialReference::SetCS()`, `OGRSpatialReference::SetEC()`, `OGRSpatialReference::SetEckertIV()`, `OGRSpatialReference::SetEckertVI()`, `OGRSpatialReference::SetEquirectangular()`, `OGRSpatialReference::SetEquirectangular2()`, `OGRSpatialReference::SetGaussSchreiberTMercator()`, `OGRSpatialReference::SetGEOS()`, `OGRSpatialReference::SetGH()`, `OGRSpatialReference::SetGnomonic()`, `OGRSpatialReference::SetGS()`, `OGRSpatialReference::SetHOM()`, `OGRSpatialReference::SetHOM2PNO()`, `OGRSpatialReference::SetHOMAC()`, `OGRSpatialReference::SetIWMPolyconic()`, `OGRSpatialReference::SetKrovak()`, `OGRSpatialReference::SetLAEA()`, `OGRSpatialReference::SetLCC()`, `OGRSpatialReference::SetLCC1SP()`, `OGRSpatialReference::SetLCCB()`, `OGRSpatialReference::SetMC()`, `OGRSpatialReference::SetMercator()`, `OGRSpatialReference::SetMercator2SP()`, `OGRSpatialReference::SetMollweide()`, `OGRSpatialReference::SetNZMG()`, `OGRSpatialReference::SetOrthographic()`, `OGRSpatialReference::SetOS()`, `OGRSpatialReference::SetPolyconic()`, `OGRSpatialReference::SetPS()`, `OGRSpatialReference::SetRobinson()`, `OGRSpatialReference::SetSinusoidal()`, `OGRSpatialReference::SetSOC()`, `OGRSpatialReference::SetStereographic()`, `OGRSpatialReference::SetTM()`, `OGRSpatialReference::SetTMG()`, `OGRSpatialReference::SetTMSO()`, `OGRSpatialReference::SetTMVariant()`, `OGRSpatialReference::SetTPED()`, and `OGRSpatialReference::SetVDG()`.



## 12.22.2.9 SRS\_PP\_FALSE\_NORTHING

```
#define SRS_PP_FALSE_NORTHING "false_northing"
```

false\_northing projection parameter

Referenced by OGRSpatialReference::AutIdentifyEPSG(), OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::GetUTMZone(), OGRSpatialReference::SetACEA(), OGRSpatialReference::SetAE(), OGRSpatialReference::SetBonne(), OGRSpatialReference::SetCEA(), OGRSpatialReference::SetCS(), OGRSpatialReference::SetEC(), OGRSpatialReference::SetEckertIV(), OGRSpatialReference::SetEckertVI(), OGRSpatialReference::SetEquirectangular(), OGRSpatialReference::SetEquirectangular2(), OGRSpatialReference::SetGaussSchreiberTMercator(), OGRSpatialReference::SetGEOS(), OGRSpatialReference::SetGH(), OGRSpatialReference::SetGnomonic(), OGRSpatialReference::SetGS(), OGRSpatialReference::SetHOM(), OGRSpatialReference::SetHOM2PNO(), OGRSpatialReference::SetHOMAC(), OGRSpatialReference::SetIWMPolyconic(), OGRSpatialReference::SetKrovak(), OGRSpatialReference::SetLAEA(), OGRSpatialReference::SetLCC(), OGRSpatialReference::SetLCC1SP(), OGRSpatialReference::SetLCCB(), OGRSpatialReference::SetMC(), OGRSpatialReference::SetMercator(), OGRSpatialReference::SetMercator2SP(), OGRSpatialReference::SetMollweide(), OGRSpatialReference::SetNZMG(), OGRSpatialReference::SetOrthographic(), OGRSpatialReference::SetOS(), OGRSpatialReference::SetPolyconic(), OGRSpatialReference::SetPS(), OGRSpatialReference::SetRobinson(), OGRSpatialReference::SetSinusoidal(), OGRSpatialReference::SetSOC(), OGRSpatialReference::SetStereographic(), OGRSpatialReference::SetTM(), OGRSpatialReference::SetTMG(), OGRSpatialReference::SetTMSO(), OGRSpatialReference::SetTMVariant(), OGRSpatialReference::SetTPED(), and OGRSpatialReference::SetVDG().

## 12.22.2.10 SRS\_PP\_FIPZONE

```
#define SRS_PP_FIPZONE "fipszone"
```

fipszone projection parameter

## 12.22.2.11 SRS\_PP\_LANDSAT\_NUMBER

```
#define SRS_PP_LANDSAT_NUMBER "landsat_number"
```

landsat\_number projection parameter

## 12.22.2.12 SRS\_PP\_LATITUDE\_OF\_1ST\_POINT

```
#define SRS_PP_LATITUDE_OF_1ST_POINT "Latitude_Of_1st_Point"
```

Latitude\_Of\_1st\_Point projection parameter

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::SetIWMPolyconic(), and OGRSpatialReference::SetTPED().

**12.22.2.13 SRS\_PP\_LATITUDE\_OF\_2ND\_POINT**

```
#define SRS_PP_LATITUDE_OF_2ND_POINT "Latitude_Of_2nd_Point"
```

Latitude\_Of\_2nd\_Point projection parameter

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::SetIWMPolyconic(), and OGRSpatialReference::SetTPED().

**12.22.2.14 SRS\_PP\_LATITUDE\_OF\_CENTER**

```
#define SRS_PP_LATITUDE_OF_CENTER "latitude_of_center"
```

latitude\_of\_center projection parameter

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::FindProjParm(), OGRSpatialReference::SetACEA(), OGRSpatialReference::SetAE(), OGRSpatialReference::SetEC(), OGRSpatialReference::SetHOM(), OGRSpatialReference::SetHOM2PNO(), OGRSpatialReference::SetHOMAC(), OGRSpatialReference::SetKrovak(), OGRSpatialReference::SetLAEA(), OGRSpatialReference::SetMC(), and OGRSpatialReference::SetSOC().

**12.22.2.15 SRS\_PP\_LATITUDE\_OF\_ORIGIN**

```
#define SRS_PP_LATITUDE_OF_ORIGIN "latitude_of_origin"
```

latitude\_of\_origin projection parameter

Referenced by OGRSpatialReference::AutoIdentifyEPSG(), OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::FindProjParm(), OGRSpatialReference::GetUTMZone(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetCS(), OGRSpatialReference::SetEquirectangular(), OGRSpatialReference::SetEquirectangular2(), OGRSpatialReference::SetGaussSchreiberTMercator(), OGRSpatialReference::SetGnomonic(), OGRSpatialReference::SetLCC(), OGRSpatialReference::SetLCC1SP(), OGRSpatialReference::SetLCCB(), OGRSpatialReference::SetMercator(), OGRSpatialReference::SetMercator2SP(), OGRSpatialReference::SetNZMG(), OGRSpatialReference::SetOrthographic(), OGRSpatialReference::SetOS(), OGRSpatialReference::SetPolyconic(), OGRSpatialReference::SetPS(), OGRSpatialReference::SetQSC(), OGRSpatialReference::SetStereographic(), OGRSpatialReference::SetTM(), OGRSpatialReference::SetTMG(), OGRSpatialReference::SetTMSO(), OGRSpatialReference::SetTMVariant(), and OGRSpatialReference::SetWagner().

**12.22.2.16 SRS\_PP\_LATITUDE\_OF\_POINT\_1**

```
#define SRS_PP_LATITUDE_OF_POINT_1 "latitude_of_point_1"
```

latitude\_of\_point\_1 projection parameter

Referenced by OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetHOM2PNO().

### 12.22.2.17 SRS\_PP\_LATITUDE\_OF\_POINT\_2

```
#define SRS_PP_LATITUDE_OF_POINT_2 "latitude_of_point_2"
```

latitude\_of\_point\_2 projection parameter

Referenced by OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetHOM2PNO().

### 12.22.2.18 SRS\_PP\_LATITUDE\_OF\_POINT\_3

```
#define SRS_PP_LATITUDE_OF_POINT_3 "latitude_of_point_3"
```

latitude\_of\_point\_3 projection parameter

### 12.22.2.19 SRS\_PP\_LONGITUDE\_OF\_1ST\_POINT

```
#define SRS_PP_LONGITUDE_OF_1ST_POINT "Longitude_Of_1st_Point"
```

Longitude\_Of\_1st\_Point projection parameter

Referenced by OGRSpatialReference::SetTPED().

### 12.22.2.20 SRS\_PP\_LONGITUDE\_OF\_2ND\_POINT

```
#define SRS_PP_LONGITUDE_OF_2ND_POINT "Longitude_Of_2nd_Point"
```

Longitude\_Of\_2nd\_Point projection parameter

Referenced by OGRSpatialReference::SetTPED().

### 12.22.2.21 SRS\_PP\_LONGITUDE\_OF\_CENTER

```
#define SRS_PP_LONGITUDE_OF_CENTER "longitude_of_center"
```

longitude\_of\_center projection parameter

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::FindProjParm(), OGRSpatialReference::SetACEA(), OGRSpatialReference::SetAE(), OGRSpatialReference::SetEC(), OGRSpatialReference::SetHOM(), OGRSpatialReference::SetHOMAC(), OGRSpatialReference::SetKrovak(), OGRSpatialReference::SetLAEA(), OGRSpatialReference::SetMC(), OGRSpatialReference::SetRobinson(), and OGRSpatialReference::SetSinusoidal().

**12.22.2.22 SRS\_PP\_LONGITUDE\_OF\_ORIGIN**

```
#define SRS_PP_LONGITUDE_OF_ORIGIN "longitude_of_origin"
```

longitude\_of\_origin projection parameter

Referenced by OGRSpatialReference::FindProjParm().

**12.22.2.23 SRS\_PP\_LONGITUDE\_OF\_POINT\_1**

```
#define SRS_PP_LONGITUDE_OF_POINT_1 "longitude_of_point_1"
```

longitude\_of\_point\_1 projection parameter

Referenced by OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetHOM2PNO().

**12.22.2.24 SRS\_PP\_LONGITUDE\_OF\_POINT\_2**

```
#define SRS_PP_LONGITUDE_OF_POINT_2 "longitude_of_point_2"
```

longitude\_of\_point\_2 projection parameter

Referenced by OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetHOM2PNO().

**12.22.2.25 SRS\_PP\_LONGITUDE\_OF\_POINT\_3**

```
#define SRS_PP_LONGITUDE_OF_POINT_3 "longitude_of_point_3"
```

longitude\_of\_point\_3 projection parameter

**12.22.2.26 SRS\_PP\_PATH\_NUMBER**

```
#define SRS_PP_PATH_NUMBER "path_number"
```

path\_number projection parameter

**12.22.2.27 SRS\_PP\_PEG\_POINT\_HEADING**

```
#define SRS_PP_PEG_POINT_HEADING "peg_point_heading"
```

peg\_point\_heading projection parameter

Referenced by OGRSpatialReference::SetSCH().

**12.22.2.28 SRS\_PP\_PEG\_POINT\_HEIGHT**

```
#define SRS_PP_PEG_POINT_HEIGHT "peg_point_height"
```

peg\_point\_height projection parameter

Referenced by OGRSpatialReference::SetSCH().

**12.22.2.29 SRS\_PP\_PEG\_POINT\_LATITUDE**

```
#define SRS_PP_PEG_POINT_LATITUDE "peg_point_latitude"
```

peg\_point\_latitude projection parameter

Referenced by OGRSpatialReference::SetSCH().

**12.22.2.30 SRS\_PP\_PEG\_POINT\_LONGITUDE**

```
#define SRS_PP_PEG_POINT_LONGITUDE "peg_point_longitude"
```

peg\_point\_longitude projection parameter

Referenced by OGRSpatialReference::SetSCH().

**12.22.2.31 SRS\_PP\_PERSPECTIVE\_POINT\_HEIGHT**

```
#define SRS_PP_PERSPECTIVE_POINT_HEIGHT "perspective_point_height"
```

perspective\_point\_height projection parameter

**12.22.2.32 SRS\_PP\_PSEUDO\_STD\_PARALLEL\_1**

```
#define SRS_PP_PSEUDO_STD_PARALLEL_1 "pseudo_standard_parallel_1"
```

pseudo\_standard\_parallel\_1 projection parameter

Referenced by OGRSpatialReference::SetKrovak().

**12.22.2.33 SRS\_PP\_RECTIFIED\_GRID\_ANGLE**

```
#define SRS_PP_RECTIFIED_GRID_ANGLE "rectified_grid_angle"
```

rectified\_grid\_angle projection parameter

Referenced by OGRSpatialReference::IsAngularParameter(), OGRSpatialReference::IsSame(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetHOM(), and OGRSpatialReference::SetHOMAC().

**12.22.2.34 SRS\_PP\_SATELLITE\_HEIGHT**

```
#define SRS_PP_SATELLITE_HEIGHT "satellite_height"
```

satellite\_height projection parameter

Referenced by OGRSpatialReference::IsLinearParameter(), and OGRSpatialReference::SetGEOS().

**12.22.2.35 SRS\_PP\_SCALE\_FACTOR**

```
#define SRS_PP_SCALE_FACTOR "scale_factor"
```

scale\_factor projection parameter

Referenced by OGRSpatialReference::AutoidentifyEPSG(), OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::GetUTMZone(), OGRSpatialReference::SetGaussSchreiberTMercator(), OGRSpatialReference::SetHOM(), OGRSpatialReference::SetHOM2PNO(), OGRSpatialReference::SetHOMAC(), OGRSpatialReference::SetKrovak(), OGRSpatialReference::SetLCC1SP(), OGRSpatialReference::SetMercator(), OGRSpatialReference::SetOS(), OGRSpatialReference::SetPS(), OGRSpatialReference::SetStereographic(), OGRSpatialReference::SetTM(), OGRSpatialReference::SetTMSO(), and OGRSpatialReference::SetTMVariant().

**12.22.2.36 SRS\_PP\_STANDARD\_PARALLEL\_1**

```
#define SRS_PP_STANDARD_PARALLEL_1 "standard_parallel_1"
```

standard\_parallel\_1 projection parameter

Referenced by OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::IsSame(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::SetACEA(), OGRSpatialReference::SetBonne(), OGRSpatialReference::SetCEA(), OGRSpatialReference::SetEC(), OGRSpatialReference::SetEquirectangular2(), OGRSpatialReference::SetLCC(), OGRSpatialReference::SetLCCB(), and OGRSpatialReference::SetMercator2SP().

### 12.22.2.37 SRS\_PP\_STANDARD\_PARALLEL\_2

```
#define SRS_PP_STANDARD_PARALLEL_2 "standard_parallel_2"
```

standard\_parallel\_2 projection parameter

Referenced by OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::IsSame(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::SetACEA(), OGRSpatialReference::SetEC(), OGRSpatialReference::SetLCC(), and OGRSpatialReference::SetLCCB().

### 12.22.2.38 SRS\_PP\_ZONE

```
#define SRS_PP_ZONE "zone"
```

zone projection parameter

### 12.22.2.39 SRS\_PT\_AITOFF

```
#define SRS_PT_AITOFF "Aitoff"
```

Aitoff projection

### 12.22.2.40 SRS\_PT\_ALBERS\_CONIC\_EQUAL\_AREA

```
#define SRS_PT_ALBERS_CONIC_EQUAL_AREA "Albers_Conic_Equal_Area"
```

Albers\_Conic\_Equal\_Area projection

Referenced by OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetACEA().

### 12.22.2.41 SRS\_PT\_AZIMUTHAL\_EQUIDISTANT

```
#define SRS_PT_AZIMUTHAL_EQUIDISTANT "Azimuthal_Equidistant"
```

Azimuthal\_Equidistant projection

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::morphFromESRI(), and OGRSpatialReference::SetAE().

**12.22.2.42 SRS\_PT\_BONNE**

```
#define SRS_PT_BONNE "Bonne"
```

Cylindrical\_Equal\_Area projection

Referenced by OGRSpatialReference::SetBonne().

**12.22.2.43 SRS\_PT\_CASSINI\_SOLDNER**

```
#define SRS_PT_CASSINI_SOLDNER "Cassini_Soldner"
```

Cassini\_Soldner projection

Referenced by OGRSpatialReference::exportToPCI(), and OGRSpatialReference::SetCS().

**12.22.2.44 SRS\_PT\_CRASTER\_PARABOLIC**

```
#define SRS_PT_CRASTER_PARABOLIC "Craster_Parabolic"
```

Craster\_Parabolic projection

**12.22.2.45 SRS\_PT\_CYLINDRICAL\_EQUAL\_AREA**

```
#define SRS_PT_CYLINDRICAL_EQUAL_AREA "Cylindrical_Equal_Area"
```

Cylindrical\_Equal\_Area projection

Referenced by OGRSpatialReference::exportToPanorama(), and OGRSpatialReference::SetCEA().

**12.22.2.46 SRS\_PT\_ECKERT\_I**

```
#define SRS_PT_ECKERT_I "Eckert_I"
```

Eckert\_I projection

Referenced by OGRSpatialReference::SetEckert().



**12.22.2.47 SRS\_PT\_ECKERT\_II**

```
#define SRS_PT_ECKERT_II "Eckert_II"
```

Eckert\_II projection

Referenced by OGRSpatialReference::SetEckert().

**12.22.2.48 SRS\_PT\_ECKERT\_III**

```
#define SRS_PT_ECKERT_III "Eckert_III"
```

Eckert\_III projection

Referenced by OGRSpatialReference::SetEckert().

**12.22.2.49 SRS\_PT\_ECKERT\_IV**

```
#define SRS_PT_ECKERT_IV "Eckert_IV"
```

Eckert\_IV projection

Referenced by OGRSpatialReference::SetEckert(), and OGRSpatialReference::SetEckertIV().

**12.22.2.50 SRS\_PT\_ECKERT\_V**

```
#define SRS_PT_ECKERT_V "Eckert_V"
```

Eckert\_V projection

Referenced by OGRSpatialReference::SetEckert().

**12.22.2.51 SRS\_PT\_ECKERT\_VI**

```
#define SRS_PT_ECKERT_VI "Eckert_VI"
```

Eckert\_VI projection

Referenced by OGRSpatialReference::SetEckert(), and OGRSpatialReference::SetEckertVI().

**12.22.2.52 SRS\_PT\_EQUIDISTANT\_CONIC**

```
#define SRS_PT_EQUIDISTANT_CONIC "Equidistant_Conic"
```

Equidistant\_Conic projection

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::morphFromESRI(), and OGRSpatialReference::SetEC().

**12.22.2.53 SRS\_PT\_EQUIRECTANGULAR**

```
#define SRS_PT_EQUIRECTANGULAR "Equirectangular"
```

Equirectangular projection

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::SetEquirectangular(), and OGRSpatialReference::SetEquirectangular2().

**12.22.2.54 SRS\_PT\_GALL\_STEREOGRAPHIC**

```
#define SRS_PT_GALL_STEREOGRAPHIC "Gall_Stereographic"
```

Gall\_Stereographic projection

Referenced by OGRSpatialReference::SetGS().

**12.22.2.55 SRS\_PT\_GAUSSSCHREIBERTMERCATOR**

```
#define SRS_PT_GAUSSSCHREIBERTMERCATOR "Gauss_Schreiber_Transverse_Mercator"
```

Gauss\_Schreiber\_Transverse\_Mercator projection

Referenced by OGRSpatialReference::SetGaussSchreiberTMercator().

**12.22.2.56 SRS\_PT\_GEOSTATIONARY\_SATELLITE**

```
#define SRS_PT_GEOSTATIONARY_SATELLITE "Geostationary_Satellite"
```

Geostationary\_Satellite projection

Referenced by OGRSpatialReference::SetGEOS().

### 12.22.2.57 SRS\_PT\_GNOMONIC

```
#define SRS_PT_GNOMONIC "Gnomonic"
```

Gnomonic projection

Referenced by `OGRSpatialReference::exportToPanorama()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, and `OGRSpatialReference::SetGnomonic()`.

### 12.22.2.58 SRS\_PT\_GOODE\_HOMOLOGOSINE

```
#define SRS_PT_GOODE_HOMOLOGOSINE "Goode_Homolosine"
```

Goode\_Homolosine projection

Referenced by `OGRSpatialReference::SetGH()`.

### 12.22.2.59 SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR

```
#define SRS_PT_HOTINE_OBLIQUE_MERCATOR "Hotine_Oblique_Mercator"
```

Hotine\_Oblique\_Mercator projection

Referenced by `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, `OGRSpatialReference::IsSame()`, `OGRSpatialReference::morphToESRI()`, and `OGRSpatialReference::SetHOM()`.

### 12.22.2.60 SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR\_AZIMUTH\_CENTER

```
#define SRS_PT_HOTINE_OBLIQUE_MERCATOR_AZIMUTH_CENTER "Hotine_Oblique_Mercator_Azimuth_Center"
```

Hotine\_Oblique\_Mercator\_Azimuth\_Center projection

Referenced by `OGRSpatialReference::SetHOMAC()`.

### 12.22.2.61 SRS\_PT\_HOTINE\_OBLIQUE\_MERCATOR\_TWO\_POINT\_NATURAL\_ORIGIN

```
#define SRS_PT_HOTINE_OBLIQUE_MERCATOR_TWO_POINT_NATURAL_ORIGIN "Hotine_Oblique_Mercator_Two_Point_Natural-Origin"
```

Hotine\_Oblique\_Mercator\_Two\_Point\_Natural-Origin projection

Referenced by `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, and `OGRSpatialReference::SetHOM2PNO()`.

**12.22.2.62 SRS\_PT\_IGH**

```
#define SRS_PT_IGH "Interrupted_Goode_Homolosine"
```

Interrupted\_Goode\_Homolosine projection

Referenced by OGRSpatialReference::SetIGH().

**12.22.2.63 SRS\_PT\_IMW\_POLYCONIC**

```
#define SRS_PT_IMW_POLYCONIC "International_Map_of_the_World_Polyconic"
```

International\_Map\_of\_the\_World\_Polyconic projection

Referenced by OGRSpatialReference::exportToPanorama(), and OGRSpatialReference::SetIWMPolyconic().

**12.22.2.64 SRS\_PT\_KROVAK**

```
#define SRS_PT_KROVAK "Krovak"
```

Krovak projection

Referenced by OGRSpatialReference::SetKrovak().

**12.22.2.65 SRS\_PT\_LABORDE\_OBLIQUE\_MERCATOR**

```
#define SRS_PT_LABORDE_OBLIQUE_MERCATOR "Laborde_Oblique_Mercator"
```

Laborde\_Oblique\_Mercator projection

**12.22.2.66 SRS\_PT\_LAMBERT\_AZIMUTHAL\_EQUAL\_AREA**

```
#define SRS_PT_LAMBERT_AZIMUTHAL_EQUAL_AREA "Lambert_Azimuthal_Equal_Area"
```

Lambert\_Azimuthal\_Equal\_Area projection

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::morphFromESRI(), and OGRSpatialReference::SetLAEA().

**12.22.2.67 SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP**

```
#define SRS_PT_LAMBERT_CONFORMAL_CONIC_1SP "Lambert_Conformal_Conic_1SP"
```

Lambert\_Conformal\_Conic\_1SP projection

Referenced by OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::morphFromESRI(), and OGRSpatialReference::SetLCC1SP().

**12.22.2.68 SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP**

```
#define SRS_PT_LAMBERT_CONFORMAL_CONIC_2SP "Lambert_Conformal_Conic_2SP"
```

Lambert\_Conformal\_Conic\_2SP projection

Referenced by OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::IsSame(), OGRSpatialReference::morphFromESRI(), and OGRSpatialReference::SetLCC().

**12.22.2.69 SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP\_BELGIUM**

```
#define SRS_PT_LAMBERT_CONFORMAL_CONIC_2SP_BELGIUM "Lambert_Conformal_Conic_2SP_Belgium"
```

Lambert\_Conformal\_Conic\_2SP\_Belgium projection

Referenced by OGRSpatialReference::SetLCCB().

**12.22.2.70 SRS\_PT\_LOXIMUTHAL**

```
#define SRS_PT_LOXIMUTHAL "Loximuthal"
```

Loximuthal projection

**12.22.2.71 SRS\_PT\_MERCATOR\_1SP**

```
#define SRS_PT_MERCATOR_1SP "Mercator_1SP"
```

Mercator\_1SP projection

Referenced by OGRSpatialReference::convertToOtherProjection(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::morphToESRI(), and OGRSpatialReference::SetMercator().

#### 12.22.2.72 SRS\_PT\_MERCATOR\_2SP

```
#define SRS_PT_MERCATOR_2SP "Mercator_2SP"
```

Mercator\_2SP projection

Referenced by `OGRSpatialReference::convertToOtherProjection()`, `OGRSpatialReference::morphFromESRI()`, `OGRSpatialReference::morphToESRI()`, and `OGRSpatialReference::SetMercator2SP()`.

#### 12.22.2.73 SRS\_PT\_MERCATOR\_AUXILIARY\_SPHERE

```
#define SRS_PT_MERCATOR_AUXILIARY_SPHERE "Mercator_Auxiliary_Sphere"
```

Mercator\_Auxiliary\_Sphere is used by ESRI to mean EPSG:3875

Referenced by `OGRSpatialReference::morphFromESRI()`.

#### 12.22.2.74 SRS\_PT\_MILLER\_CYLINDRICAL

```
#define SRS_PT_MILLER_CYLINDRICAL "Miller_Cylindrical"
```

Miller\_Cylindrical projection

Referenced by `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, and `OGRSpatialReference::SetMC()`.

#### 12.22.2.75 SRS\_PT\_MOLLWEIDE

```
#define SRS_PT_MOLLWEIDE "Mollweide"
```

Mollweide projection

Referenced by `OGRSpatialReference::exportToPanorama()`, `OGRSpatialReference::exportToUSGS()`, and `OGRSpatialReference::SetMollweide()`.

#### 12.22.2.76 SRS\_PT\_NEW\_ZEALAND\_MAP\_GRID

```
#define SRS_PT_NEW_ZEALAND_MAP_GRID "New_Zealand_Map_Grid"
```

New\_Zealand\_Map\_Grid projection

Referenced by `OGRSpatialReference::SetNZMG()`.

### 12.22.2.77 SRS\_PT\_OBLIQUE\_STEREOGRAPHIC

```
#define SRS_PT_OBLIQUE_STEREOGRAPHIC "Oblique_Stereographic"
```

Oblique\_Stereographic projection

Referenced by OGRSpatialReference::exportToPCI(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), and OGRSpatialReference::SetOS().

### 12.22.2.78 SRS\_PT\_ORTHOGRAPHIC

```
#define SRS_PT_ORTHOGRAPHIC "Orthographic"
```

Orthographic projection

Referenced by OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetOrthographic().

### 12.22.2.79 SRS\_PT\_POLAR\_STEREOGRAPHIC

```
#define SRS_PT_POLAR_STEREOGRAPHIC "Polar_Stereographic"
```

Polar\_Stereographic projection

Referenced by OGRSpatialReference::AutolIdentifyEPSG(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), and OGRSpatialReference::SetPS().

### 12.22.2.80 SRS\_PT\_POLYCONIC

```
#define SRS_PT_POLYCONIC "Polyconic"
```

Polyconic projection

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetPolyconic().

### 12.22.2.81 SRS\_PT\_QSC

```
#define SRS_PT_QSC "Quadrilateralized_Spherical_Cube"
```

Quadrilateralized\_Spherical\_Cube projection

Referenced by OGRSpatialReference::SetQSC().

**12.22.2.82 SRS\_PT\_QUARTIC\_AUTHALIC**

```
#define SRS_PT_QUARTIC_AUTHALIC "Quartic_Authalic"
```

Quartic\_Authalic projection

**12.22.2.83 SRS\_PT\_ROBINSON**

```
#define SRS_PT_ROBINSON "Robinson"
```

Robinson projection

Referenced by `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, `OGRSpatialReference::morphFromESRI()`, and `OGRSpatialReference::SetRobinson()`.

**12.22.2.84 SRS\_PT\_SCH**

```
#define SRS_PT_SCH "Spherical_Cross_Track_Height"
```

Spherical\_Cross\_Track\_Height projection

Referenced by `OGRSpatialReference::SetSCH()`.

**12.22.2.85 SRS\_PT\_SINUSOIDAL**

```
#define SRS_PT_SINUSOIDAL "Sinusoidal"
```

Sinusoidal projection

Referenced by `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, `OGRSpatialReference::morphFromESRI()`, and `OGRSpatialReference::SetSinusoidal()`.

**12.22.2.86 SRS\_PT\_STEREOGRAPHIC**

```
#define SRS_PT_STEREOGRAPHIC "Stereographic"
```

Stereographic projection

Referenced by `OGRSpatialReference::exportToPanorama()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, and `OGRSpatialReference::SetStereographic()`.



**12.22.2.87 SRS\_PT\_SWISS\_OBLIQUE\_CYLINDRICAL**

```
#define SRS_PT_SWISS_OBLIQUE_CYLINDRICAL "Swiss_Oblique_Cylindrical"
```

Swiss\_Oblique\_Cylindrical projection

Referenced by OGRSpatialReference::SetSOC().

**12.22.2.88 SRS\_PT\_TRANSVERSE\_MERCATOR**

```
#define SRS_PT_TRANSVERSE_MERCATOR "Transverse_Mercator"
```

Transverse\_Mercator projection

Referenced by OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRSpatialReference::GetUTMZone(), OGRSpatialReference::ImportFromESRI(), OGRSpatialReference::ImportFromWKT(), and OGRSpatialReference::SetTM().

**12.22.2.89 SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_21**

```
#define SRS_PT_TRANSVERSE_MERCATOR_MI_21 "Transverse_Mercator_MapInfo_21"
```

Transverse\_Mercator\_MapInfo\_21 projection

**12.22.2.90 SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_22**

```
#define SRS_PT_TRANSVERSE_MERCATOR_MI_22 "Transverse_Mercator_MapInfo_22"
```

Transverse\_Mercator\_MapInfo\_22 projection

**12.22.2.91 SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_23**

```
#define SRS_PT_TRANSVERSE_MERCATOR_MI_23 "Transverse_Mercator_MapInfo_23"
```

Transverse\_Mercator\_MapInfo\_23 projection

**12.22.2.92 SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_24**

```
#define SRS_PT_TRANSVERSE_MERCATOR_MI_24 "Transverse_Mercator_MapInfo_24"
```

Transverse\_Mercator\_MapInfo\_24 projection

**12.22.2.93 SRS\_PT\_TRANSVERSE\_MERCATOR\_MI\_25**

```
#define SRS_PT_TRANSVERSE_MERCATOR_MI_25 "Transverse_Mercator_MapInfo_25"
```

Transverse\_Mercator\_MapInfo\_25 projection

**12.22.2.94 SRS\_PT\_TRANSVERSE\_MERCATOR\_SOUTH\_ORIENTED**

```
#define SRS_PT_TRANSVERSE_MERCATOR_SOUTH_ORIENTED "Transverse_Mercator_South_Orientated"
```

Transverse\_Mercator\_South\_Orientated projection

Referenced by OGRSpatialReference::SetTMSO().

**12.22.2.95 SRS\_PT\_TUNISIA\_MINING\_GRID**

```
#define SRS_PT_TUNISIA_MINING_GRID "Tunisia_Mining_Grid"
```

Tunisia\_Mining\_Grid projection

Referenced by OGRSpatialReference::SetTMG().

**12.22.2.96 SRS\_PT\_TWO\_POINT\_EQUIDISTANT**

```
#define SRS_PT_TWO_POINT_EQUIDISTANT "Two_Point_Equidistant"
```

Two\_Point\_Equidistant projection

Referenced by OGRSpatialReference::SetTPED().

**12.22.2.97 SRS\_PT\_VANDERGRINTEN**

```
#define SRS_PT_VANDERGRINTEN "VanDerGrinten"
```

VanDerGrinten projection

Referenced by OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetVDG().

**12.22.2.98 SRS\_PT\_WAGNER\_I**

```
#define SRS_PT_WAGNER_I "Wagner_I"
```

Wagner\_I projection

Referenced by `OGRSpatialReference::exportToPanorama()`, and `OGRSpatialReference::SetWagner()`.

**12.22.2.99 SRS\_PT\_WAGNER\_II**

```
#define SRS_PT_WAGNER_II "Wagner_II"
```

Wagner\_II projection

Referenced by `OGRSpatialReference::SetWagner()`.

**12.22.2.100 SRS\_PT\_WAGNER\_III**

```
#define SRS_PT_WAGNER_III "Wagner_III"
```

Wagner\_III projection

Referenced by `OGRSpatialReference::SetWagner()`.

**12.22.2.101 SRS\_PT\_WAGNER\_IV**

```
#define SRS_PT_WAGNER_IV "Wagner_IV"
```

Wagner\_IV projection

Referenced by `OGRSpatialReference::exportToUSGS()`, and `OGRSpatialReference::SetWagner()`.

**12.22.2.102 SRS\_PT\_WAGNER\_V**

```
#define SRS_PT_WAGNER_V "Wagner_V"
```

Wagner\_V projection

Referenced by `OGRSpatialReference::SetWagner()`.

**12.22.2.103 SRS\_PT\_WAGNER\_VI**

```
#define SRS_PT_WAGNER_VI "Wagner_VI"
```

Wagner\_VI projection

Referenced by OGRSpatialReference::SetWagner().

**12.22.2.104 SRS\_PT\_WAGNER\_VII**

```
#define SRS_PT_WAGNER_VII "Wagner_VII"
```

Wagner\_VII projection

Referenced by OGRSpatialReference::exportToUSGS(), and OGRSpatialReference::SetWagner().

**12.22.2.105 SRS\_PT\_WINKEL\_I**

```
#define SRS_PT_WINKEL_I "Winkel_I"
```

Winkel\_I projection

**12.22.2.106 SRS\_PT\_WINKEL\_II**

```
#define SRS_PT_WINKEL_II "Winkel_II"
```

Winkel\_II projection

**12.22.2.107 SRS\_PT\_WINKEL\_TRIPEL**

```
#define SRS_PT_WINKEL_TRIPEL "Winkel_Tripel"
```

Winkel\_Tripel projection

**12.22.2.108 SRS\_UA\_DEGREE**

```
#define SRS_UA_DEGREE "degree"
```

Angular unit degree

Referenced by OGRSpatialReference::Fixup(), and OGRSpatialReference::SetGeogCS().

**12.22.2.109 SRS\_UA\_DEGREE\_CONV**

```
#define SRS_UA_DEGREE_CONV "0.0174532925199433"
```

Angular unit degree conversion factor to radians

Referenced by OGRSpatialReference::CloneGeogCS(), OGRSpatialReference::Fixup(), OGRSpatialReference::↔GetAngularUnits(), OGRSpatialReference::IsSameGeogCS(), and OGRSpatialReference::SetGeogCS().

**12.22.2.110 SRS\_UA\_RADIAN**

```
#define SRS_UA_RADIAN "radian"
```

Angular unit radian

**12.22.2.111 SRS\_UL\_CENTIMETER**

```
#define SRS_UL_CENTIMETER "Centimeter"
```

Linear unit Decimeter

**12.22.2.112 SRS\_UL\_CENTIMETER\_CONV**

```
#define SRS_UL_CENTIMETER_CONV "0.01"
```

Linear unit Decimeter conversion factor to meter

**12.22.2.113 SRS\_UL\_CHAIN**

```
#define SRS_UL_CHAIN "Chain" /* based on US Foot */
```

Linear unit Chain

**12.22.2.114 SRS\_UL\_CHAIN\_CONV**

```
#define SRS_UL_CHAIN_CONV "20.116684023368047"
```

Linear unit Chain conversion factor to meter

**12.22.2.115 SRS\_UL\_DECIMETER**

```
#define SRS_UL_DECIMETER "Decimeter"
```

Linear unit Decimeter

**12.22.2.116 SRS\_UL\_DECIMETER\_CONV**

```
#define SRS_UL_DECIMETER_CONV "0.1"
```

Linear unit Decimeter conversion factor to meter

**12.22.2.117 SRS\_UL\_FOOT**

```
#define SRS_UL_FOOT "Foot (International)" /* or just "FOOT"? */
```

Linear unit Foot (International)

Referenced by OGRSpatialReference::importFromPCI().

**12.22.2.118 SRS\_UL\_FOOT\_CONV**

```
#define SRS_UL_FOOT_CONV "0.3048"
```

Linear unit Foot (International) conversion factor to meter

Referenced by OGRSpatialReference::importFromPCI().

**12.22.2.119 SRS\_UL\_INDIAN\_CHAIN**

```
#define SRS_UL_INDIAN_CHAIN "Chain_Indian"
```

Linear unit Chain\_Indian

**12.22.2.120 SRS\_UL\_INDIAN\_CHAIN\_CONV**

```
#define SRS_UL_INDIAN_CHAIN_CONV "20.11669506"
```

Linear unit Chain\_Indian conversion factor to meter

**12.22.2.121 SRS\_UL\_INDIAN\_FOOT**

```
#define SRS_UL_INDIAN_FOOT "Foot_Indian"
```

Linear unit Foot\_Indian

**12.22.2.122 SRS\_UL\_INDIAN\_FOOT\_CONV**

```
#define SRS_UL_INDIAN_FOOT_CONV "0.30479841"
```

Linear unit Foot\_Indian conversion factor to meter

**12.22.2.123 SRS\_UL\_INDIAN\_YARD**

```
#define SRS_UL_INDIAN_YARD "Yard_Indian"
```

Linear unit Yard\_Indian

**12.22.2.124 SRS\_UL\_INDIAN\_YARD\_CONV**

```
#define SRS_UL_INDIAN_YARD_CONV "0.91439523"
```

Linear unit Yard\_Indian conversion factor to meter

**12.22.2.125 SRS\_UL\_INTL\_CHAIN**

```
#define SRS_UL_INTL_CHAIN "Chain_International"
```

Linear unit Chain\_International

**12.22.2.126 SRS\_UL\_INTL\_CHAIN\_CONV**

```
#define SRS_UL_INTL_CHAIN_CONV "20.1168"
```

Linear unit Chain\_International conversion factor to meter

**12.22.2.127 SRS\_UL\_INTL\_FATHOM**

```
#define SRS_UL_INTL_FATHOM "Fathom_International"
```

Linear unit Fathom\_International

**12.22.2.128 SRS\_UL\_INTL\_FATHOM\_CONV**

```
#define SRS_UL_INTL_FATHOM_CONV "1.8288"
```

Linear unit Fathom\_International conversion factor to meter

**12.22.2.129 SRS\_UL\_INTL\_FOOT**

```
#define SRS_UL_INTL_FOOT "Foot_International"
```

Linear unit Foot\_International

**12.22.2.130 SRS\_UL\_INTL\_FOOT\_CONV**

```
#define SRS_UL_INTL_FOOT_CONV "0.3048"
```

Linear unit Foot\_International conversion factor to meter

**12.22.2.131 SRS\_UL\_INTL\_INCH**

```
#define SRS_UL_INTL_INCH "Inch_International"
```

Linear unit Inch\_International

**12.22.2.132 SRS\_UL\_INTL\_INCH\_CONV**

```
#define SRS_UL_INTL_INCH_CONV "0.0254"
```

Linear unit Inch\_International conversion factor to meter

**12.22.2.133 SRS\_UL\_INTL\_LINK**

```
#define SRS_UL_INTL_LINK "Link_International"
```

Linear unit Link\_International

**12.22.2.134 SRS\_UL\_INTL\_LINK\_CONV**

```
#define SRS_UL_INTL_LINK_CONV "0.201168"
```

Linear unit Link\_International conversion factor to meter

**12.22.2.135 SRS\_UL\_INTL\_NAUT\_MILE**

```
#define SRS_UL_INTL_NAUT_MILE "Nautical_Mile_International"
```

Linear unit Nautical\_Mile\_International

**12.22.2.136 SRS\_UL\_INTL\_NAUT\_MILE\_CONV**

```
#define SRS_UL_INTL_NAUT_MILE_CONV "1852.0"
```

Linear unit Nautical\_Mile\_International conversion factor to meter

**12.22.2.137 SRS\_UL\_INTL\_STAT\_MILE**

```
#define SRS_UL_INTL_STAT_MILE "Statute_Mile_International"
```

Linear unit Statute\_Mile\_International

**12.22.2.138 SRS\_UL\_INTL\_STAT\_MILE\_CONV**

```
#define SRS_UL_INTL_STAT_MILE_CONV "1609.344"
```

Linear unit Statute\_Mile\_International conversion factor to meter



**12.22.2.139 SRS\_UL\_INTL\_YARD**

```
#define SRS_UL_INTL_YARD "Yard_International"
```

Linear unit Yard\_International

**12.22.2.140 SRS\_UL\_INTL\_YARD\_CONV**

```
#define SRS_UL_INTL_YARD_CONV "0.9144"
```

Linear unit Yard\_International conversion factor to meter

**12.22.2.141 SRS\_UL\_KILOMETER**

```
#define SRS_UL_KILOMETER "Kilometer"
```

Linear unit Kilometer

**12.22.2.142 SRS\_UL\_KILOMETER\_CONV**

```
#define SRS_UL_KILOMETER_CONV "1000."
```

Linear unit Kilometer conversion factor to meter

**12.22.2.143 SRS\_UL\_LINK**

```
#define SRS_UL_LINK "Link" /* Based on US Foot */
```

Linear unit Link

**12.22.2.144 SRS\_UL\_LINK\_Clarke**

```
#define SRS_UL_LINK_Clarke "Link_Clarke"
```

Linear unit Link\_Clarke

**12.22.2.145 SRS\_UL\_LINK\_Clarke\_CONV**

```
#define SRS_UL_LINK_Clarke_CONV "0.2011661949"
```

Linear unit Link\_Clarke conversion factor to meter

**12.22.2.146 SRS\_UL\_LINK\_CONV**

```
#define SRS_UL_LINK_CONV "0.20116684023368047"
```

Linear unit Link conversion factor to meter

**12.22.2.147 SRS\_UL\_METER**

```
#define SRS_UL_METER "Meter"
```

Linear unit Meter

Referenced by `OGRSpatialReference::Fixup()`, `OGRSpatialReference::importFromERM()`, and `OGRSpatialReference::importFromPCI()`.

**12.22.2.148 SRS\_UL\_MILLIMETER**

```
#define SRS_UL_MILLIMETER "Millimeter"
```

Linear unit Millimeter

**12.22.2.149 SRS\_UL\_MILLIMETER\_CONV**

```
#define SRS_UL_MILLIMETER_CONV "0.001"
```

Linear unit Millimeter conversion factor to meter

**12.22.2.150 SRS\_UL\_NAUTICAL\_MILE**

```
#define SRS_UL_NAUTICAL_MILE "Nautical Mile"
```

Linear unit Nautical Mile

**12.22.2.151 SRS\_UL\_NAUTICAL\_MILE\_CONV**

```
#define SRS_UL_NAUTICAL_MILE_CONV "1852.0"
```

Linear unit Nautical Mile conversion factor to meter

**12.22.2.152 SRS\_UL\_ROD**

```
#define SRS_UL_ROD "Rod" /* based on US Foot */
```

Linear unit Rod

**12.22.2.153 SRS\_UL\_ROD\_CONV**

```
#define SRS_UL_ROD_CONV "5.02921005842012"
```

Linear unit Rod conversion factor to meter

**12.22.2.154 SRS\_UL\_US\_CHAIN**

```
#define SRS_UL_US_CHAIN "Chain_US_Surveyor"
```

Linear unit Chain\_US\_Surveyor

**12.22.2.155 SRS\_UL\_US\_CHAIN\_CONV**

```
#define SRS_UL_US_CHAIN_CONV "20.11684023368047"
```

Linear unit Chain\_US\_Surveyor conversion factor to meter

**12.22.2.156 SRS\_UL\_US\_FOOT**

```
#define SRS_UL_US_FOOT "Foot_US" /* or "US survey foot" from EPSG */
```

Linear unit Foot

Referenced by OGRSpatialReference::importFromERM(), and OGRSpatialReference::importFromPCI().

**12.22.2.157 SRS\_UL\_US\_FOOT\_CONV**

```
#define SRS_UL_US_FOOT_CONV "0.3048006096012192"
```

Linear unit Foot conversion factor to meter

Referenced by OGRSpatialReference::importFromERM(), and OGRSpatialReference::importFromPCI().

**12.22.2.158 SRS\_UL\_US\_INCH**

```
#define SRS_UL_US_INCH "Inch_US_Surveyor"
```

Linear unit Inch\_US\_Surveyor

**12.22.2.159 SRS\_UL\_US\_INCH\_CONV**

```
#define SRS_UL_US_INCH_CONV "0.025400050800101603"
```

Linear unit Inch\_US\_Surveyor conversion factor to meter

**12.22.2.160 SRS\_UL\_US\_STAT\_MILE**

```
#define SRS_UL_US_STAT_MILE "Statute_Mile_US_Surveyor"
```

Linear unit Statute\_Mile\_US\_Surveyor

**12.22.2.161 SRS\_UL\_US\_STAT\_MILE\_CONV**

```
#define SRS_UL_US_STAT_MILE_CONV "1609.347218694437"
```

Linear unit Statute\_Mile\_US\_Surveyor conversion factor to meter

**12.22.2.162 SRS\_UL\_US\_YARD**

```
#define SRS_UL_US_YARD "Yard_US_Surveyor"
```

Linear unit Yard\_US\_Surveyor

**12.22.2.163 SRS\_UL\_US\_YARD\_CONV**

```
#define SRS_UL_US_YARD_CONV "0.914401828803658"
```

Linear unit Yard\_US\_Surveyor conversion factor to meter

**12.22.2.164 SRS\_WGS84\_INVFLATTENING**

```
#define SRS_WGS84_INVFLATTENING 298.257223563
```

Inverse flattening of the WGS84 ellipsoid

Referenced by OGRSpatialReference::GetInvFlattening().

**12.22.2.165 SRS\_WGS84\_SEMIMAJOR**

```
#define SRS_WGS84_SEMIMAJOR 6378137.0
```

Semi-major axis of the WGS84 ellipsoid

Referenced by OGRSpatialReference::GetSemiMajor().

**12.22.2.166 SRS\_WKT\_WGS84**

```
#define SRS_WKT_WGS84 "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\", \"7030\"]],AUTHORITY[\"EPSG\", \"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\", \"8901\"]],UNIT[\"degree\",0.0174532925199433,AUTHORITY[\"EPSG\", \"9122\"]],AUTHORITY[\"EPSG\", \"4326\"]]"
```

WGS 84 geodetic (long/lat) WKT / EPSG:4326 with long,lat ordering

Referenced by OGRSpatialReference::GetWGS84SRS(), and OGRSpatialReference::SetWellKnownGeogCS().

### 12.22.3 Typedef Documentation

#### 12.22.3.1 OGRCoordinateTransformationH

```
typedef void* OGRCoordinateTransformationH
```

Opaque type for a coordinate transformation object

#### 12.22.3.2 OGRSpatialReferenceH

```
typedef void* OGRSpatialReferenceH
```

Opaque type for a Spatial Reference object

### 12.22.4 Enumeration Type Documentation

#### 12.22.4.1 OGRAxisOrientation

```
enum OGRAxisOrientation
```

Axis orientations (corresponds to CS\_AxisOrientationEnum).

##### Enumerator

OA_Other	Other
OA_North	North
OA_South	South
OA_East	East
OA_West	West
OA_Up	Up (to space)
OA_Down	Down (to Earth center)

### 12.22.5 Function Documentation

#### 12.22.5.1 OCTDestroyCoordinateTransformation()

```
void CPL_STDCALL OCTDestroyCoordinateTransformation (
    OGRCoordinateTransformationH hCT )
```

**OGRCoordinateTransformation** (p. ??) destructor.

This function is the same as **OGRCoordinateTransformation::DestroyCT()** (p. ??)

#### Parameters

<i>hCT</i>	the object to delete
------------	----------------------

References OGRCoordinateTransformation::FromHandle().

#### 12.22.5.2 OCTNewCoordinateTransformation()

```
OGRCoordinateTransformationH CPL_STDCALL OCTNewCoordinateTransformation (
    OGRSpatialReferenceH hSourceSRS,
    OGRSpatialReferenceH hTargetSRS )
```

Create transformation object.

This is the same as the C++ function **OGRCreateCoordinateTransformation()** (p. ??).

Input spatial reference system objects are assigned by copy (calling clone() method) and no ownership transfer occurs.

**OCTDestroyCoordinateTransformation()** (p. ??) should be used to destroy transformation objects.

The PROJ.4 library must be available at run-time.

#### Parameters

<i>hSourceSRS</i>	source spatial reference system.
<i>hTargetSRS</i>	target spatial reference system.

#### Returns

NULL on failure or a ready to use transformation object.

References OGRCreateCoordinateTransformation().

#### 12.22.5.3 OCTTransform()

```
int CPL_STDCALL OCTTransform (
    OGRCoordinateTransformationH hTransform,
    int nCount,
    double * x,
    double * y,
    double * z )
```

Transform an array of points

## Parameters

<i>hTransform</i>	Transformation object
<i>nCount</i>	Number of points
<i>x</i>	Array of nCount x values.
<i>y</i>	Array of nCount y values.
<i>z</i>	Array of nCount z values.

## Returns

TRUE or FALSE

## 12.22.5.4 OCTTransformEx()

```
int CPL_STDCALL OCTTransformEx (
    OGRCoordinateTransformationH hTransform,
    int nCount,
    double * x,
    double * y,
    double * z,
    int * pabSuccess )
```

Transform an array of points

## Parameters

<i>hTransform</i>	Transformation object
<i>nCount</i>	Number of points
<i>x</i>	Array of nCount x values.
<i>y</i>	Array of nCount y values.
<i>z</i>	Array of nCount z values.
<i>pabSuccess</i>	Output array of nCount value that will be set to TRUE/FALSE

## Returns

TRUE or FALSE

## 12.22.5.5 OPTGetParameterList()

```
char** OPTGetParameterList (
    const char * pszProjectionMethod,
    char ** ppszUserName )
```

Fetch the parameters for a given projection method.

## Parameters

<i>pszProjectionMethod</i>	internal name of projection methods to fetch the parameters for, such as "Transverse_Mercator" (SRS_PT_TRANSVERSE_MERCATOR).
<i>ppszUserName</i>	pointer in which to return a user visible name for the projection name. The returned string should not be modified or freed by the caller. Legal to pass in NULL if user name not required.

## Returns

returns a NULL terminated list of internal parameter names that should be freed by the caller when no longer needed. Returns NULL if projection method is unknown.

References CPLCalloc(), CSLAddString(), and EQUAL.

## 12.22.5.6 OPTGetProjectionMethods()

```
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.

References CSLAddString(), and EQUAL.

## 12.22.5.7 OSRAutoIdentifyEPSG()

```
OGRERR OSRAutoIdentifyEPSG (
    OGRSpatialReferenceH hSRS )
```

Set EPSG authority info if possible.

This function is the same as **OGRSpatialReference::AutoIdentifyEPSG()** (p. ??).

Since GDAL 2.3, the **OSRFindMatches()** (p. ??) function can also be used for improved matching by researching the EPSG catalog. < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.



### 12.22.5.8 OSRAxisEnumToName()

```
const char* OSRAxisEnumToName (
    OGRAxisOrientation eOrientation )
```

Return the string representation for the OGRAxisOrientation enumeration.

For example "NORTH" for OAO\_North.

#### Returns

an internal string

References OAO\_Down, OAO\_East, OAO\_North, OAO\_Other, OAO\_South, OAO\_Up, and OAO\_West.

Referenced by OGRSpatialReference::SetAxes().

### 12.22.5.9 OSRCalcInvFlattening()

```
double OSRCalcInvFlattening (
    double dfSemiMajor,
    double dfSemiMinor )
```

Compute inverse flattening from semi-major and semi-minor axis.

#### Parameters

<i>dfSemiMajor</i>	Semi-major axis length.
<i>dfSemiMinor</i>	Semi-minor axis length.

#### Returns

inverse flattening, or 0 if both axis are equal.

#### Since

GDAL 2.0

References CPLError().

### 12.22.5.10 OSRCalcSemiMinorFromInvFlattening()

```
double OSRCalcSemiMinorFromInvFlattening (
    double dfSemiMajor,
    double dfInvFlattening )
```

Compute semi-minor axis from semi-major axis and inverse flattening.

**Parameters**

<i>dfSemiMajor</i>	Semi-major axis length.
<i>dfInvFlattening</i>	Inverse flattening or 0 for sphere.

**Returns**

semi-minor axis

**Since**

GDAL 2.0

References CPLError().

Referenced by OGRSpatialReference::exportToPCI(), and OGRSpatialReference::GetSemiMinor().

**12.22.5.11 OSRCleanup()**

```
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.

**12.22.5.12 OSRClone()**

```
OGRSpatialReferenceH CPL_STDCALL OSRClone (
    OGRSpatialReferenceH hSRS )
```

Make a duplicate of this **OGRSpatialReference** (p. ??).

This function is the same as **OGRSpatialReference::Clone()** (p. ??)

**12.22.5.13 OSRCloneGeogCS()**

```
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. ??).

#### 12.22.5.14 OSRConvertToOtherProjection()

```
OGRSpatialReferenceH OSRConvertToOtherProjection (
    OGRSpatialReferenceH hSRS,
    const char * pszTargetProjection,
    const char *const * papszOptions )
```

Convert to another equivalent projection.

Currently implemented:

- SRS\_PT\_MERCATOR\_1SP to SRS\_PT\_MERCATOR\_2SP
- SRS\_PT\_MERCATOR\_2SP to SRS\_PT\_MERCATOR\_1SP
- SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP to SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP
- SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_2SP to SRS\_PT\_LAMBERT\_CONFORMAL\_CONIC\_1SP

## Parameters

<i>hSRS</i>	source SRS
<i>pszTargetProjection</i>	target projection.
<i>papszOptions</i>	lists of options. None supported currently.

## Returns

a new SRS, or NULL in case of error.

## Since

GDAL 2.3

References VALIDATE\_POINTER1.

## 12.22.5.15 OSRCopyGeogCSFrom()

```
OGRERR OSRCopyGeogCSFrom (
    OGRSpatialReferenceH hSRS,
    const OGRSpatialReferenceH hSrcSRS )
```

Copy GEOGCS from another **OGRSpatialReference** (p. ??).

This function is the same as **OGRSpatialReference::CopyGeogCSFrom()** (p. ??) < Failure  
< Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.16 OSRDereference()

```
int OSRDereference (
    OGRSpatialReferenceH hSRS )
```

Decrements the reference count by one.

This function is the same as **OGRSpatialReference::Dereference()** (p. ??)

References VALIDATE\_POINTER1.

## 12.22.5.17 OSRDestroySpatialReference()

```
void CPL_STDCALL OSRDestroySpatialReference (
    OGRSpatialReferenceH hSRS )
```

**OGRSpatialReference** (p. ??) destructor.

This function is the same as **OGRSpatialReference::~OGRSpatialReference()** (p. ??) and **OGRSpatialReference::DestroySpatialReference()** (p. ??)

## Parameters

<i>hSRS</i>	the object to delete
-------------	----------------------

## 12.22.5.18 OSREPSGTreatsAsLatLong()

```
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()`. < Failure

References `OGRERR_FAILURE`, and `VALIDATE_POINTER1`.

## 12.22.5.19 OSREPSGTreatsAsNorthingEasting()

```
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

< Failure

References `OGRERR_FAILURE`, and `VALIDATE_POINTER1`.

## 12.22.5.20 OSRExportToERM()

```
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. ??). < Failure

References `OGRERR_FAILURE`, and `VALIDATE_POINTER1`.

## 12.22.5.21 OSRExportToMCoordSys()

```

OGRERR OSRExportToMCoordSys (
    OGRSpatialReferenceH hSRS,
    char ** ppszReturn )

```

Export coordinate system in Mapinfo style CoordSys format.

This method is the equivalent of the C++ method **OGRSpatialReference::exportToMCoordSys** (p. ??) < Failure  
References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.22 OSRExportToPanorama()

```

OGRERR OSRExportToPanorama (
    OGRSpatialReferenceH hSRS,
    long * piProjSys,
    long * piDatum,
    long * piEllips,
    long * piZone,
    double * padfPrjParams )

```

Export coordinate system in "Panorama" GIS projection definition.

See **OGRSpatialReference::exportToPanorama()** (p. ??) < Failure

< Failure

< Failure

< Failure

< Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.23 OSRExportToPCI()

```

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. ??). < Failure

References OGRSpatialReference::exportToPCI(), OGRSpatialReference::FromHandle(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.24 OSRExportToPrettyWkt()

```
OGRErr CPL_STDCALL OSRExportToPrettyWkt (
    OGRSpatialReferenceH hSRS,
    char ** ppszReturn,
    int bSimplify )
```

Convert this SRS into a nicely formatted WKT string for display to a person.

This function is the same as **OGRSpatialReference::exportToPrettyWkt()** (p. ??). < Failure

## 12.22.5.25 OSRExportToProj4()

```
OGRErr CPL_STDCALL OSRExportToProj4 (
    OGRSpatialReferenceH hSRS,
    char ** ppszReturn )
```

Export coordinate system in PROJ.4 format.

This function is the same as **OGRSpatialReference::exportToProj4()** (p. ??). < Failure

## 12.22.5.26 OSRExportToUSGS()

```
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. ??). < Failure

References **OGRSpatialReference::exportToUSGS()**, **OGRSpatialReference::FromHandle()**, **OGRErr\_FAILURE**, and **VALIDATE\_POINTER1**.

## 12.22.5.27 OSRExportToWkt()

```
OGRErr CPL_STDCALL OSRExportToWkt (
    OGRSpatialReferenceH hSRS,
    char ** ppszReturn )
```

Convert this SRS into WKT format.

Consult also the [OGC WKT Coordinate System Issues](#) page for implementation details of WKT in OGR.

This function is the same as **OGRSpatialReference::exportToWkt()** (p. ??). < Failure

### 12.22.5.28 OSRExportToXML()

```
OGRErr OSRExportToXML (
    OGRSpatialReferenceH hSRS,
    char ** ppszRawXML,
    const char * pszDialect )
```

Export coordinate system in XML format.

This function is the same as **OGRSpatialReference::exportToXML()** (p. ??). < Failure

References **OGRSpatialReference::exportToXML()**, **OGRSpatialReference::FromHandle()**, **OGRERR\_FAILURE**, and **VALIDATE\_POINTER1**.

### 12.22.5.29 OSRFindMatches()

```
OGRSpatialReferenceH* OSRFindMatches (
    OGRSpatialReferenceH hSRS,
    char ** ppszOptions,
    int * pnEntries,
    int ** ppanMatchConfidence )
```

Try to identify a match between the passed SRS and a related SRS in a catalog (currently EPSG only)

Matching may be partial, or may fail. Returned entries will be sorted by decreasing match confidence (first entry has the highest match confidence).

The exact way matching is done may change in future versions.

The current algorithm is:

- try first AutoidentifyEPSG(). If it succeeds, return the corresponding SRS
- otherwise iterate over all SRS from the EPSG catalog (as found in GDAL pcs.csv and gcs.csv files+esri\_↵ extra.wkt), and find those that match the input SRS using the IsSame() function (ignoring TOWGS84 clauses)
- if there is a single match using IsSame() or one of the matches has the same SRS name, return it with 100% confidence
- if a SRS has the same SRS name, but does not pass the IsSame() criteria, return it with 50% confidence.
- otherwise return all candidate SRS that pass the IsSame() criteria with a 90% confidence.

A pre-built SRS cache in `~/gdal/X.Y/srs_cache` will be used if existing, otherwise it will be built at the first run of this function.

This function is the same as **OGRSpatialReference::FindMatches()** (p. ??).

#### Parameters

<i>hSRS</i>	SRS to match
<i>ppszOptions</i>	NULL terminated list of options or NULL
<i>pnEntries</i>	Output parameter. Number of values in the returned array.
<i>ppanMatchConfidence</i>	Output parameter (or NULL). *ppanMatchConfidence will be allocated to an array of *pnEntries whose values between 0 and 100 indicate the confidence in the match. 100 is the highest confidence level. The array must be freed with <b>CPLFree()</b> (p. ??).



**Returns**

an array of SRS that match the passed SRS, or NULL. Must be freed with **OSRFreeSRSArray()** (p. ??)

**Since**

GDAL 2.3

References **OGRSpatialReference::FindMatches()**, and **VALIDATE\_POINTER1**.

**12.22.5.30 OSRFixup()**

```
OGRERR OSRFixup (
    OGRSpatialReferenceH hSRS )
```

Fixup as needed.

This function is the same as **OGRSpatialReference::Fixup()** (p. ??). < Failure

References **OGRERR\_FAILURE**, and **VALIDATE\_POINTER1**.

**12.22.5.31 OSRFixupOrdering()**

```
OGRERR OSRFixupOrdering (
    OGRSpatialReferenceH hSRS )
```

Correct parameter ordering to match CT Specification.

This function is the same as **OGRSpatialReference::FixupOrdering()** (p. ??). < Failure

References **OGRERR\_FAILURE**, and **VALIDATE\_POINTER1**.

**12.22.5.32 OSRFreeSRSArray()**

```
void OSRFreeSRSArray (
    OGRSpatialReferenceH * pahSRS )
```

Free return of OSRIdentifyMatches()

**Parameters**

<i>pahSRS</i>	array of SRS (must be NULL terminated)
---------------	--

Since

GDAL 2.3

References CPLFree, and OSRRelease().

#### 12.22.5.33 OSRGetAngularUnits()

```
double OSRGetAngularUnits (
    OGRSpatialReferenceH hSRS,
    char ** ppszName )
```

Fetch angular geographic coordinate system units.

This function is the same as **OGRSpatialReference::GetAngularUnits()** (p. ??)

References VALIDATE\_POINTER1.

#### 12.22.5.34 OSRGetAttrValue()

```
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. ??)

#### 12.22.5.35 OSRGetAuthorityCode()

```
const char* OSRGetAuthorityCode (
    OGRSpatialReferenceH hSRS,
    const char * pszTargetKey )
```

Get the authority code for a node.

This function is the same as **OGRSpatialReference::GetAuthorityCode()** (p. ??).

References VALIDATE\_POINTER1.

#### 12.22.5.36 OSRGetAuthorityName()

```
const char* OSRGetAuthorityName (
    OGRSpatialReferenceH hSRS,
    const char * pszTargetKey )
```

Get the authority name for a node.

This function is the same as **OGRSpatialReference::GetAuthorityName()** (p. ??).

References `VALIDATE_POINTER1`.

#### 12.22.5.37 OSRGetAxis()

```
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. ??)

References `VALIDATE_POINTER1`.

#### 12.22.5.38 OSRGetInvFlattening()

```
double OSRGetInvFlattening (
    OGRSpatialReferenceH hSRS,
    OGRERR * pnErr )
```

Get spheroid inverse flattening.

This function is the same as **OGRSpatialReference::GetInvFlattening()** (p. ??)

References `VALIDATE_POINTER1`.

#### 12.22.5.39 OSRGetLinearUnits()

```
double OSRGetLinearUnits (
    OGRSpatialReferenceH hSRS,
    char ** ppszName )
```

Fetch linear projection units.

This function is the same as **OGRSpatialReference::GetLinearUnits()** (p. ??)

References `VALIDATE_POINTER1`.

#### 12.22.5.40 OSRGetNormProjParm()

```
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. ??)

References `VALIDATE_POINTER1`.

#### 12.22.5.41 OSRGetPrimeMeridian()

```
double OSRGetPrimeMeridian (
    OGRSpatialReferenceH hSRS,
    char ** ppszName )
```

Fetch prime meridian info.

This function is the same as **OGRSpatialReference::GetPrimeMeridian()** (p. ??)

References `VALIDATE_POINTER1`.

#### 12.22.5.42 OSRGetProjParm()

```
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. ??)

References `VALIDATE_POINTER1`.

#### 12.22.5.43 OSRGetSemiMajor()

```
double OSRGetSemiMajor (
    OGRSpatialReferenceH hSRS,
    OGRErr * pnErr )
```

Get spheroid semi major axis.

This function is the same as **OGRSpatialReference::GetSemiMajor()** (p. ??)

References `VALIDATE_POINTER1`.

#### 12.22.5.44 OSRGetSemiMinor()

```
double OSRGetSemiMinor (
    OGRSpatialReferenceH hSRS,
    OGRERR * pnErr )
```

Get spheroid semi minor axis.

This function is the same as **OGRSpatialReference::GetSemiMinor()** (p. ??)

References VALIDATE\_POINTER1.

#### 12.22.5.45 OSRGetTargetLinearUnits()

```
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

References VALIDATE\_POINTER1.

#### 12.22.5.46 OSRGetTOWGS84()

```
OGRERR OSRGetTOWGS84 (
    OGRSpatialReferenceH hSRS,
    double * padfCoeff,
    int nCoeffCount )
```

Fetch TOWGS84 parameters, if available.

This function is the same as **OGRSpatialReference::GetTOWGS84()** (p. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.47 OSRGetUTMZone()

```
int OSRGetUTMZone (
    OGRSpatialReferenceH hSRS,
    int * pbNorth )
```

Get utm zone information.

This is the same as the C++ method **OGRSpatialReference::GetUTMZone()** (p. ??)

References `VALIDATE_POINTER1`.

## 12.22.5.48 OSRImportFromDict()

```
OGRERR OSRImportFromDict (
    OGRSpatialReferenceH hSRS,
    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()** (p. ??). Normally this results in searching `/usr/local/share/gdal` or somewhere similar.

This method is the same as the C++ method **OGRSpatialReference::importFromDict()** (p. ??).

## Parameters

<i>hSRS</i>	spatial reference system handle.
<i>pszDictFile</i>	the name of the dictionary file to load.
<i>pszCode</i>	the code to lookup in the dictionary.

## Returns

`OGRERR_NONE` on success, or `OGRERR_SRS_UNSUPPORTED` if the code isn't found, and `OGRERR_↔SRS_FAILURE` if something more dramatic goes wrong.

< Failure

References `OGRERR_FAILURE`, and `VALIDATE_POINTER1`.

#### 12.22.5.49 OSRImportFromEPSG()

```
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. ??). < Failure

#### 12.22.5.50 OSRImportFromEPSGA()

```
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. ??). < Failure

#### 12.22.5.51 OSRImportFromERM()

```
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. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.52 OSRImportFromESRI()

```
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. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.53 OSRImportFromMICoordSys()

```
OGRERR OSRImportFromMICoordSys (
    OGRSpatialReferenceH hSRS,
    const char * pszCoordSys )
```

Import Mapinfo style CoordSys definition.

This method is the equivalent of the C++ method **OGRSpatialReference::importFromMICoordSys** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.54 OSRImportFromOzi()

```
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

<i>hSRS</i>	spatial reference object.
<i>papszLines</i>	Map file lines. This is an array of strings containing the whole OziExplorer .MAP file. The array is terminated by a NULL pointer.

#### Returns

OGRERR\_NONE on success or an error code in case of failure.

#### Since

OGR 2.0

< Failure

References OGRSpatialReference::FromHandle(), OGRSpatialReference::importFromOzi(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.



#### 12.22.5.55 OSRImportFromPanorama()

```
OGRERR OSRImportFromPanorama (
    OGRSpatialReferenceH hSRS,
    long iProjSys,
    long iDatum,
    long iEllips,
    double * padfPrjParams )
```

Import coordinate system from "Panorama" GIS projection definition.

See **OGRSpatialReference::importFromPanorama()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.56 OSRImportFromPCI()

```
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. ??). < Failure

References OGRSpatialReference::FromHandle(), OGRSpatialReference::importFromPCI(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.57 OSRImportFromProj4()

```
OGRERR OSRImportFromProj4 (
    OGRSpatialReferenceH hSRS,
    const char * pszProj4 )
```

Import PROJ.4 coordinate string.

This function is the same as **OGRSpatialReference::importFromProj4()** (p. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.58 OSRImportFromUrl()**

```

OGRERR OSRImportFromUrl (
    OGRSpatialReferenceH hSRS,
    const char * pszUrl )

```

Set spatial reference from a URL.

This function is the same as **OGRSpatialReference::importFromUrl()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.59 OSRImportFromUSGS()**

```

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. ??). < Failure

References OGRSpatialReference::FromHandle(), OGRSpatialReference::importFromUSGS(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.60 OSRImportFromWkt()**

```

OGRERR OSRImportFromWkt (
    OGRSpatialReferenceH hSRS,
    char ** ppszInput )

```

Import from WKT string.

Consult also the [OGC WKT Coordinate System Issues](#) page for implementation details of WKT in OGR.

This function is the same as **OGRSpatialReference::importFromWkt()** (p. ??). < Failure

References OGRGeometry::importFromWkt(), OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.61 OSRImportFromXML()

```
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. ??). < Failure  
< Failure

References **OGRSpatialReference::FromHandle()**, **OGRSpatialReference::importFromXML()**, **OGRERR\_FAILURE**, and **VALIDATE\_POINTER1**.

### 12.22.5.62 OSRIsCompound()

```
int OSRIsCompound (
    OGRSpatialReferenceH hSRS )
```

Check if the coordinate system is compound.

This function is the same as **OGRSpatialReference::IsCompound()** (p. ??).

References **VALIDATE\_POINTER1**.

### 12.22.5.63 OSRIsGeocentric()

```
int OSRIsGeocentric (
    OGRSpatialReferenceH hSRS )
```

Check if geocentric coordinate system.

This function is the same as **OGRSpatialReference::IsGeocentric()** (p. ??).

Since

OGR 1.9.0

References **VALIDATE\_POINTER1**.

### 12.22.5.64 OSRIsGeographic()

```
int OSRIsGeographic (
    OGRSpatialReferenceH hSRS )
```

Check if geographic coordinate system.

This function is the same as **OGRSpatialReference::IsGeographic()** (p. ??).

References **VALIDATE\_POINTER1**.

#### 12.22.5.65 OSRIsLocal()

```
int OSRIsLocal (
    OGRSpatialReferenceH hSRS )
```

Check if local coordinate system.

This function is the same as **OGRSpatialReference::IsLocal()** (p. ??).

References VALIDATE\_POINTER1.

#### 12.22.5.66 OSRIsProjected()

```
int OSRIsProjected (
    OGRSpatialReferenceH hSRS )
```

Check if projected coordinate system.

This function is the same as **OGRSpatialReference::IsProjected()** (p. ??).

References VALIDATE\_POINTER1.

#### 12.22.5.67 OSRIsSame()

```
int OSRIsSame (
    OGRSpatialReferenceH hSRS1,
    OGRSpatialReferenceH hSRS2 )
```

Do these two spatial references describe the same system ?

This function is the same as **OGRSpatialReference::IsSame()** (p. ??).

References VALIDATE\_POINTER1.

#### 12.22.5.68 OSRIsSameGeogCS()

```
int OSRIsSameGeogCS (
    OGRSpatialReferenceH hSRS1,
    OGRSpatialReferenceH hSRS2 )
```

Do the GeogCS'es match?

This function is the same as **OGRSpatialReference::IsSameGeogCS()** (p. ??).

References VALIDATE\_POINTER1.

### 12.22.5.69 OSRIsSameVertCS()

```
int OSRIsSameVertCS (
    OGRSpatialReferenceH hSRS1,
    OGRSpatialReferenceH hSRS2 )
```

Do the VertCS'es match?

This function is the same as **OGRSpatialReference::IsSameVertCS()** (p. ??).

References VALIDATE\_POINTER1.

### 12.22.5.70 OSRIsVertical()

```
int OSRIsVertical (
    OGRSpatialReferenceH hSRS )
```

Check if vertical coordinate system.

This function is the same as **OGRSpatialReference::IsVertical()** (p. ??).

Since

OGR 1.8.0

References VALIDATE\_POINTER1.

### 12.22.5.71 OSRMorphFromESRI()

```
OGRERR OSRMorphFromESRI (
    OGRSpatialReferenceH hSRS )
```

Convert in place from ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphFromESRI()** (p. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.72 OSRMorphToESRI()

```
OGRERR OSRMorphToESRI (
    OGRSpatialReferenceH hSRS )
```

Convert in place to ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphToESRI()** (p. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.73 OSRNewSpatialReference()**

```
OGRSpatialReferenceH CPL_STDCALL OSRNewSpatialReference (
    const char * pszWKT )
```

Constructor.

This function is the same as **OGRSpatialReference::OGRSpatialReference()** (p. ??) < Success

**12.22.5.74 OSRReference()**

```
int OSRReference (
    OGRSpatialReferenceH hSRS )
```

Increments the reference count by one.

This function is the same as **OGRSpatialReference::Reference()** (p. ??)

References VALIDATE\_POINTER1.

**12.22.5.75 OSRRelease()**

```
void OSRRelease (
    OGRSpatialReferenceH hSRS )
```

Decrements the reference count by one, and destroy if zero.

This function is the same as **OGRSpatialReference::Release()** (p. ??)

References VALIDATE\_POINTER0.

Referenced by OSRFreeSRSArray().

**12.22.5.76 OSRSetACEA()**

```
OGRERR OSRSetACEA (
    OGRSpatialReferenceH hSRS,
    double dfStdP1,
    double dfStdP2,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Albers Conic Equal Area < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.77 OSRSetAE()

```

OGRERR OSRSetAE (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Azimuthal Equidistant < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.78 OSRSetAngularUnits()

```

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. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.79 OSRSetAttrValue()

```

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. ??) < Failure

## 12.22.5.80 OSRSetAuthority()

```

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. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.81 OSRSetAxes()

```
OGRErr OSRSetAxes (
    OGRSpatialReferenceH hSRS,
    const char * pszTargetKey,
    const char * pszXAxisName,
    OGRAxisOrientation eXAxisOrientation,
    const char * pszYAxisName,
    OGRAxisOrientation eYAxisOrientation )
```

Set the axes for a coordinate system.

This method is the equivalent of the C++ method **OGRSpatialReference::SetAxes** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.82 OSRSetBonne()

```
OGRErr OSRSetBonne (
    OGRSpatialReferenceH hSRS,
    double dfStandardParallel,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Bonne < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.83 OSRSetCEA()

```
OGRErr OSRSetCEA (
    OGRSpatialReferenceH hSRS,
    double dfStdPl,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Cylindrical Equal Area < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.



## 12.22.5.84 OSRSetCompoundCS()

```
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. ??) < Failure

< Failure

< Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.85 OSRSetCS()

```
OGRERR OSRSetCS (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Cassini-Soldner < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.86 OSRSetEC()

```
OGRERR OSRSetEC (
    OGRSpatialReferenceH hSRS,
    double dfStdP1,
    double dfStdP2,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Equidistant Conic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.87 OSRSetEckert()

```
OGRErr OSRSetEckert (
    OGRSpatialReferenceH hSRS,
    int nVariation,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Eckert I-VI < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.88 OSRSetEckertIV()

```
OGRErr OSRSetEckertIV (
    OGRSpatialReferenceH hSRS,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Eckert IV < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.89 OSRSetEckertVI()

```
OGRErr OSRSetEckertVI (
    OGRSpatialReferenceH hSRS,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Eckert VI < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.90 OSRSetEquirectangular()

```
OGRErr OSRSetEquirectangular (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Equirectangular < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.91 OSRSetEquiangular2()

```

OGRERR OSRSetEquiangular2 (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfPseudoStdParallel,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Equiangular generalized form < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.92 OSRSetFromUserInput()

```

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. ??) < Failure

## 12.22.5.93 OSRSetGaussSchreiberTMercator()

```

OGRERR OSRSetGaussSchreiberTMercator (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Gauss Schreiber Transverse Mercator < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.94 OSRSetGeocCS()

```

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

< Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.95 OSRSetGeogCS()**

```

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. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.96 OSRSetGEOS()**

```

OGRERR OSRSetGEOS (
    OGRSpatialReferenceH hSRS,
    double dfCentralMeridian,
    double dfSatelliteHeight,
    double dfFalseEasting,
    double dfFalseNorthing )

```

GEOS - Geostationary Satellite View < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.97 OSRSetGH()**

```

OGRERR OSRSetGH (
    OGRSpatialReferenceH hSRS,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Goode Homolosine < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.98 OSRSetGnomonic()

```
OGRERR OSRSetGnomonic (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Gnomonic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.99 OSRSetGS()

```
OGRERR OSRSetGS (
    OGRSpatialReferenceH hSRS,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Gall Stereographic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.100 OSRSetHOM()

```
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. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.101 OSRSetHOM2PNO()**

```

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. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.102 OSRSetHOMAC()**

```

OGRERR OSRSetHOMAC (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfAzimuth,
    double dfRectToSkew,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Set an Oblique Mercator projection using azimuth angle.

This is the same as the C++ method **OGRSpatialReference::SetHOMAC()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

**12.22.5.103 OSRSetIGH()**

```

OGRERR OSRSetIGH (
    OGRSpatialReferenceH hSRS )

```

Interrupted Goode Homolosine < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.104 OSRSetIWMPolyconic()

```
OGRERR OSRSetIWMPolyconic (
    OGRSpatialReferenceH hSRS,
    double dfLat1,
    double dfLat2,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

International Map of the World Polyconic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.105 OSRSetKrovak()

```
OGRERR OSRSetKrovak (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfAzimuth,
    double dfPseudoStdParallelLat,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Krovak Oblique Conic Conformal < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.106 OSRSetLAEA()

```
OGRERR OSRSetLAEA (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Lambert Azimuthal Equal-Area < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.107 OSRSetLCC()

```
OGRERR OSRSetLCC (
    OGRSpatialReferenceH hSRS,
    double dfStdP1,
    double dfStdP2,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Lambert Conformal Conic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.108 OSRSetLCC1SP()

```
OGRERR OSRSetLCC1SP (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Lambert Conformal Conic 1SP < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.109 OSRSetLCCB()

```
OGRERR OSRSetLCCB (
    OGRSpatialReferenceH hSRS,
    double dfStdP1,
    double dfStdP2,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Lambert Conformal Conic (Belgium) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.



#### 12.22.5.110 OSRSetLinearUnits()

```
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. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.111 OSRSetLinearUnitsAndUpdateParameters()

```
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. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.112 OSRSetLocalCS()

```
OGRERR OSRSetLocalCS (
    OGRSpatialReferenceH hSRS,
    const char * pszName )
```

Set the user visible LOCAL\_CS name.

This function is the same as **OGRSpatialReference::SetLocalCS()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.113 OSRSetMC()

```
OGRERR OSRSetMC (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Miller Cylindrical < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.114 OSRSetMercator()

```
OGRErr OSRSetMercator (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Mercator < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.115 OSRSetMercator2SP()

```
OGRErr OSRSetMercator2SP (
    OGRSpatialReferenceH hSRS,
    double dfStdP1,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Mercator 2SP < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.116 OSRSetMollweide()

```
OGRErr OSRSetMollweide (
    OGRSpatialReferenceH hSRS,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Mollweide < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.117 OSRSetNormProjParm()

```
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. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.118 OSRSetNZMG()

```
OGRErr OSRSetNZMG (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

New Zealand Map Grid < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.119 OSRSetOrthographic()

```
OGRErr OSRSetOrthographic (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Orthographic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.120 OSRSetOS()

```
OGRErr OSRSetOS (
    OGRSpatialReferenceH hSRS,
    double dfOriginLat,
    double dfCMeridian,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Oblique Stereographic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.121 OSRSetPolyconic()

```
OGRErr OSRSetPolyconic (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Polyconic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.122 OSRSetProjCS()

```
OGRERR OSRSetProjCS (
    OGRSpatialReferenceH hSRS,
    const char * pszName )
```

Set the user visible PROJCS name.

This function is the same as **OGRSpatialReference::SetProjCS()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.123 OSRSetProjection()

```
OGRERR OSRSetProjection (
    OGRSpatialReferenceH hSRS,
    const char * pszProjection )
```

Set a projection name.

This function is the same as **OGRSpatialReference::SetProjection()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.124 OSRSetProjParm()

```
OGRERR OSRSetProjParm (
    OGRSpatialReferenceH hSRS,
    const char * pszParmName,
    double dfValue )
```

Set a projection parameter value.

This function is the same as **OGRSpatialReference::SetProjParm()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.125 OSRSetPS()

```
OGRERR OSRSetPS (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Polar Stereographic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.126 OSRSetQSC()

```
OGRErr OSRSetQSC (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong )
```

Quadrilateralized Spherical Cube < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.127 OSRSetRobinson()

```
OGRErr OSRSetRobinson (
    OGRSpatialReferenceH hSRS,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Robinson < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.128 OSRSetSCH()

```
OGRErr OSRSetSCH (
    OGRSpatialReferenceH hSRS,
    double dfPegLat,
    double dfPegLong,
    double dfPegHeading,
    double dfPegHgt )
```

Spherical, Cross-track, Height < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.129 OSRSetSinusoidal()

```
OGRErr OSRSetSinusoidal (
    OGRSpatialReferenceH hSRS,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Sinusoidal < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.130 OSRSetSOC()

```
OGRErr OSRSetSOC (
    OGRSpatialReferenceH hSRS,
    double dfLatitudeOfOrigin,
    double dfCentralMeridian,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Swiss Oblique Cylindrical < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.131 OSRSetStatePlane()

```
OGRErr OSRSetStatePlane (
    OGRSpatialReferenceH hSRS,
    int nZone,
    int bNAD83 )
```

Set State Plane projection definition.

This function is the same as **OGRSpatialReference::SetStatePlane()** (p. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.132 OSRSetStatePlaneWithUnits()

```
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. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.133 OSRSetStereographic()

```

OGRErr OSRSetStereographic (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )

```

Stereographic < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.134 OSRSetTargetLinearUnits()

```

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

< Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.22.5.135 OSRSetTM()

```

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. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.136 OSRSetTMG()

```
OGRERR OSRSetTMG (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Tunesia Mining Grid < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.137 OSRSetTMSO()

```
OGRERR OSRSetTMSO (
    OGRSpatialReferenceH hSRS,
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Transverse Mercator (South Oriented) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.138 OSRSetTMVariant()

```
OGRERR OSRSetTMVariant (
    OGRSpatialReferenceH hSRS,
    const char * pszVariantName,
    double dfCenterLat,
    double dfCenterLong,
    double dfScale,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Transverse Mercator variant < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.



### 12.22.5.139 OSRSetTOWGS84()

```
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. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.140 OSRSetTPED()

```
OGRErr OSRSetTPED (
    OGRSpatialReferenceH hSRS,
    double dfLat1,
    double dfLong1,
    double dfLat2,
    double dfLong2,
    double dfFalseEasting,
    double dfFalseNorthing )
```

TPED (Two Point Equi Distant) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

### 12.22.5.141 OSRSetUTM()

```
OGRErr OSRSetUTM (
    OGRSpatialReferenceH hSRS,
    int nZone,
    int bNorth )
```

Set UTM projection definition.

This is the same as the C++ method **OGRSpatialReference::SetUTM()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.142 OSRSetVDG()

```
OGRErr OSRSetVDG (
    OGRSpatialReferenceH hSRS,
    double dfCenterLong,
    double dfFalseEasting,
    double dfFalseNorthing )
```

VanDerGrinten < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.143 OSRSetVertCS()

```
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

< Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.144 OSRSetWagner()

```
OGRErr OSRSetWagner (
    OGRSpatialReferenceH hSRS,
    int nVariation,
    double dfCenterLat,
    double dfFalseEasting,
    double dfFalseNorthing )
```

Wagner I – VII < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.145 OSRSetWellKnownGeogCS()

```
OGRERR OSRSetWellKnownGeogCS (
    OGRSpatialReferenceH hSRS,
    const char * pszName )
```

Set a GeogCS based on well known name.

This function is the same as **OGRSpatialReference::SetWellKnownGeogCS()** (p. ??) < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.146 OSRStripCTParms()

```
OGRERR OSRStripCTParms (
    OGRSpatialReferenceH hSRS )
```

Strip OGC CT Parameters.

This function is the same as **OGRSpatialReference::StripCTParms()** (p. ??). < Failure

References OGRERR\_FAILURE, and VALIDATE\_POINTER1.

#### 12.22.5.147 OSRValidate()

```
OGRERR OSRValidate (
    OGRSpatialReferenceH hSRS )
```

Validate SRS tokens.

This function is the same as the C++ method **OGRSpatialReference::Validate()**. < Failure

References **OGRSpatialReference::FromHandle()**, OGRERR\_FAILURE, and VALIDATE\_POINTER1.

## 12.23 ograpispy.h File Reference

```
#include "gdal.h"
```

### 12.23.1 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 snapshotting relies on if the dataset correctly implements GetFileList() (for multi-file datasources)

Since

GDAL 2.0

## 12.24 ogrsf\_frmts.h File Reference

```
#include "cpl_progress.h"
#include "ogr_feature.h"
#include "ogr_featurestyle.h"
#include "gdal_priv.h"
#include <memory>
```

### Classes

- class **OGRLayer**
- class **OGRDataSource**
- class **OGRSFDriver**
- class **OGRSFDriverRegistrar**

### Functions

- OGRLayer::FeatureIterator **begin** ( OGRLayer \*poLayer)
- OGRLayer::FeatureIterator **end** ( OGRLayer \*poLayer)

### 12.24.1 Detailed Description

Classes related to registration of format support, and opening datasets.

## 12.24.2 Function Documentation

### 12.24.2.1 begin()

```
OGRLayer::FeatureIterator begin (
    OGRLayer * poLayer ) [inline]
```

Return begin of feature iterator.

Using this iterator for standard range-based loops is safe, but due to implementation limitations, you shouldn't try to access (dereference) more than one iterator step at a time, since the `std::unique_ptr<OGRFeature (p. ??)>` reference is reused.

Only one iterator per layer can be active at a time.

Since

GDAL 2.3

See also

**OGRLayer::begin()** (p. ??)

References OGRLayer::begin.

### 12.24.2.2 end()

```
OGRLayer::FeatureIterator end (
    OGRLayer * poLayer ) [inline]
```

Return end of feature iterator.

See also

**OGRLayer::end()** (p. ??)

References OGRLayer::end.

Referenced by CPLvsprintf().

