OGR

Contents

Chapter 1

OGR Simple Feature Library

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

OGR is a part of the GDAL library.

Resources

- OGR Supported Formats
- OGR Utility Programs
- OGR Class Documentation
- OGR C++ API Read/Write Tutorial
- OGR Driver Implementation Tutorial
- ogr_api.h: OGR C API
- OGR Projections Tutorial
- OGR Architecture
- OGR SQL
- OGR - Feature Style Specification
- SFC (OLE DB client side API) Tutorial
- Adam's 2.5 D Simple Features Proposal (OGC 99-402r2)
- Adam's SRS WKT Clarification Proposal in html or doc format.

Download

Ready to Use Executables

The best way to get OGR utilities in ready-to-use form is to download the latest FWTools kit for your platform. While large, these include builds of the OGR utilities with lots of optional components built-in. Once downloaded follow the included instructions to setup your path and other environment variables

correctly, and then you can use the various OGR utilities from the command line. The kits also include OpenEV, a viewer that will display OGR supported vector files.

Source

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

Bug Reporting

GDAL/OGR bugs can be reported, and can be listed using Trac.

Mailing Lists

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

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

Alternative Bindings for the OGR API

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

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

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

Chapter 2

OGR API Tutorial

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

2.1 Reading From OGR

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

Initially it is necessary to register all the format drivers that are desired. This is normally accomplished by calling **OGRRegisterAll()** (p. 408) which registers all format drivers built into GDAL/OGR.

```
#include "ogrsf_frmts.h"

int main()

{
        OGRRegisterAll();
```

Next we need to open the input OGR datasource. Datasources can be files, RDBMSes, directories full of files, or even remote web services depending on the driver being used. However, the datasource name is always a single string. In this case we are hardcoded to open a particular shapefile. The second argument (FALSE) tells the **OGRSFDriverRegistrar::Open()** (p. 221) method that we don't require update access. On failure NULL is returned, and we report an error.

```
    OGRDataSource       *poDS;

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

An **OGRDataSource** (p. 92) can potentially have many layers associated with it. The number of layers available can be queried with **OGRDataSource::GetLayerCount()** (p. 95) and individual layers fetched by index using **OGRDataSource::GetLayer()** (p. 95). However, we wil just fetch the layer by name.

```
    OGRLayer  *poLayer;

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

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

While it isn't strictly necessary in this circumstance since we are starting fresh with the layer, it is often wise to call **OGRLayer::ResetReading()** (p. 166) to ensure we are starting at the beginning of the layer. We iterate through all the features in the layer using **OGRLayer::GetNextFeature()** (p. 164). It will return NULL when we run out of features.

```
    OGRFeature *poFeature;

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

In order to dump all the attribute fields of the feature, it is helpful to get the **OGRFeatureDefn** (p. 116). 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.

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

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

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

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. 107) method. In fact we could shorten the above by using **OGRFeature::GetFieldAsString()** (p. 107) 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. 127) 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.

```
        OGRGeometry *poGeometry;

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

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

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

Note that **OGRFeature::GetGeometryRef()** (p. 108) returns a pointer to the internal geometry owned by the **OGRFeature** (p. 101). There we don't actually deleted the return geometry. However, the **OGRLayer::GetNextFeature()** (p. 164) 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.

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

The **OGRLayer** (p. 160) returned by **OGRDataSource::GetLayerByName()** (p. 95) is also a reference to an internal layer owned by the **OGRDataSource** (p. 92) so we don't need to delete it. But we do need to

delete the datasource in order to close the input file. Once again we do this with a custom delete method to avoid special win32 heap issus.

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

All together our program looks like this.

```
#include "ogrsf_frmts.h"

int main()

{
    OGRRegisterAll();

    OGRDataSource       *poDS;

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

    OGRLayer  *poLayer;

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

    OGRFeature *poFeature;

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

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

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

        OGRGeometry *poGeometry;

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

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

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

2.2 Writing To OGR

As an example of writing through OGR, we will do roughly the opposite of the above. A short program that reads comma seperated 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.

```
#include "ogrsf_frmts.h"

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

    OGRRegisterAll();

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

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

```
    OGRDataSource *poDS;

    poDS = poDriver->CreateDataSource( "point_out.shp", NULL );
    if( poDS == 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.

```
    OGRLayer *poLayer;

    poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
    if( poLayer == 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. 121) 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. 121) we pass to CreateField() is copied internally. We retain ownership of the object.

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

    oField.SetWidth(32);

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

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

\code
    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. 101), 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. 116) associated with the layer it will be written to.

```
        OGRFeature *poFeature;

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

We create a local geometry object, and assign its copy (indirectly) to the feature. The **OGRFeature::SetGeometryDirectly()** (p. 114) differs from **OGRFeature::SetGeometry()** (p. 113) 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.

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

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

Now we create a feature in the file. The **OGRLayer::CreateFeature()** (p. 160) does not take ownership of our feature so we clean it up when done with it.

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

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

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.

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

The same program all in one block looks like this:

```
#include "ogrsf_frmts.h"

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

    OGRRegisterAll();

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

    OGRDataSource *poDS;

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

    OGRLayer *poLayer;

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

    OGRFieldDefn oField( "Name", OFTString );

    oField.SetWidth(32);

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

    double x, y;
    char szName[33];

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

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

        OGRPoint pt;
```

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

        poFeature->SetGeometry( &pt );

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

         OGRFeature::DestroyFeature( poFeature );
    }

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

Chapter 3

OGR Architecture

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

Because OGR is modelled on the OpenGIS simple features data model, it is very helpful to review the SFCOM, or other simple features interface specifications which can be retrieved from the `Open GIS Consortium` web site. Data types, and method names are modelled on those from the interface specifications.

3.1 Class Overview

- **Geometry** (`ogr_geometry.h`): The geometry classes (**OGRGeometry** (p. 127), 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. 224) encapsulates the definition of a projection and datum.
- **Feature** (`ogr_feature.h`): The **OGRFeature** (p. 101) encapsulate the definition of a whole feature, that is a geometry and a set of attributes.
- **Feature Class Definition** (`ogr_feature.h`): The **OGRFeatureDefn** (p. 116) class capsures the schema (set of field definitions) for a group of related features (normally a whole layer).
- **Layer** (`ogrsf_frmts.h`): **OGRLayer** (p. 160) is an abstract base class represent a layer of features in an **OGRDataSource** (p. 92).
- **Data Source** (`ogrsf_frmts.h`): An **OGRDataSource** (p. 92) is an abstract base class representing a file or database containing one or more **OGRLayer** (p. 160) objects.
- **Drivers** (`ogrsf_frmts.h`): An **OGRSFDriver** (p. 217) represents a translator for a specific format, opening **OGRDataSource** (p. 92) objects. All available drivers are managed by the **OGRSFDriverRegistrar** (p. 220).

3.2 Geometry

The geometry classes are represent various kinds of vector geometry. All the geometry classes derived from **OGRGeometry** (p. 127) which defines the common services of all geometries. Types of geometry include **OGRPoint** (p. 198), **OGRLineString** (p. 175), **OGRPolygon** (p. 206), **OGRGeometryCollection** (p. 144), **OGRMultiPolygon** (p. 194), **OGRMultiPoint** (p. 191), and **OGRMultiLineString** (p. 188).

Additional intermediate abstract base classes contain functionality that could eventually be implemented by other geometry types. These include **OGRCurve** (p. 90) (base class for **OGRLineString** (p. 175)) and **OGRSurface** (p. 272) (base class for **OGRPolygon** (p. 206)). Some intermediate interfaces modelled in the simple features abstract model and SFCOM are not modelled in OGR at this time. In most cases the methods are aggregated into other classes. This may change.

The **OGRGeometryFactory** (p. 154) 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. 127) includes a reference to an **OGRSpatialReference** (p. 224) 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. 127) 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. 127).

While it is theoretically possible to derive other or more specific geometry classes from the existing **OGRGeometry** (p. 127) classes, this isn't as aspect that has been well thought out. In particular, it would be possible to create specialized classes using the **OGRGeometryFactory** (p. 154) without modifying it.

3.3 Spatial Reference

The **OGRSpatialReference** (p. 224) class is intended to store an OpenGIS Spatial Reference System definition. Currently local, geographic and projected coordinate systems are supported. Vertical coordinate systems, geocentric coordinate systems, and compound (horizontal + vertical) coordinate systems are not supported.

The spatial coordinate system data model is inherited from the OpenGIS **Well Known Text** format. A simple form of this is defined in the Simple Features specifications. A more sophisticated form is found in the Coordinate Transformation specification. The **OGRSpatialReference** (p. 224) 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. 88) 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. 224) class.

3.4 Feature / Feature Definition

The **OGRGeometry** (p. 127) captures the geometry of a vector feature ... the spatial position/region of a feature. The **OGRFeature** (p. 101) contains this geometry, and adds feature attributes, feature id, and a feature class identify.

The set of attributes, their types, names and so forth is represented via the **OGRFeatureDefn** (p. 116) class. One **OGRFeatureDefn** (p. 116) normally exists for a layer of features. The same definition is shared in a reference counted manner by the feature of that type (or feature class).

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

The feature class also contains an indicator of the types of geometry allowed for that feature class (returned as an OGRwkbGeometryType from **OGRFeatureDefn::GetGeomType()** (p. 118)). If this is wkbUnknown then any type of geometry is allowed. This implies that features in a given layer can potentially be of different geometry types though they will always share a common attribute schema.

The **OGRFeatureDefn** (p. 116) also contains a concept of default spatial reference system for all features of that type and a feature class name (normally used as a layer name).

3.5 Layer

An **OGRLayer** (p. 160) represents a layer of features within a data source. All features in an **OGRLayer** (p. 160) share a common schema and are of the same **OGRFeatureDefn** (p. 116). An **OGRLayer** (p. 160) class also contains methods for reading features from the data source. The **OGRLayer** (p. 160) 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. 160) represents a spatial table.

The **OGRLayer** (p. 160) includes methods for sequential and random reading and writing. Read access (via the **OGRLayer::GetNextFeature()** (p. 164) 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. 160) (via the **OGRLayer::SetSpatialFilter()** (p. 167) method).

One flaw in the current OGR architecture is that the spatial filter is set directly on the **OGRLayer** (p. 160) 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 introduct an OGRLayerView class or something similar.

Another question that might arise is why the **OGRLayer** (p. 160) and **OGRFeatureDefn** (p. 116) classes are distinct. An **OGRLayer** (p. 160) always has a one-to-one relationship to an **OGRFeatureDefn** (p. 116), so why not amalgamate the classes. There are two reasons:

1. As defined now **OGRFeature** (p. 101) and **OGRFeatureDefn** (p. 116) don't depend on **OGRLayer** (p. 160), 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. 160) class is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. OGRLayers are normally owned directly by their **OGRDataSource** (p. 92), and aren't instantiated or destroyed directly.

3.6 Data Source

An **OGRDataSource** (p. 92) represents a set of **OGRLayer** (p. 160) objects. This usually represents a single file, set of files, database or gateway. An **OGRDataSource** (p. 92) has a list of OGRLayer's which it owns but can return references to.

OGRDataSource (p. 92) is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. **OGRDataSource** (p. 92) objects are not normally instantiated directly but rather with the assistance of an **OGRSFDriver** (p. 217). Deleting an **OGRDataSource** (p. 92) closes access to the underlying persistent data source, but does not normally result in deletion of that file.

An **OGRDataSource** (p. 92) has a name (usually a filename) that can be used to reopen the data source with an **OGRSFDriver** (p. 217).

The **OGRDataSource** (p. 92) also has support for executing a datasource specific command, normally a form of SQL. This is accomplished via the **OGRDataSource::ExecuteSQL()** (p. 94) method. While some datasources (such as PostGIS and Oracle) pass the SQL through to an underlying database, OGR also includes support for evaluating a subset of the SQL SELECT statement against any datasource.

3.7 Drivers

An **OGRSFDriver** (p. 217) object is instantiated for each file format supported. The **OGRSFDriver** (p. 217) objects are registered with the **OGRSFDriverRegistrar** (p. 220), a singleton class that is normally used to open new data sources.

It is intended that a new **OGRSFDriver** (p. 217) derived class be implemented for each file format to be supported (along with a file format specific **OGRDataSource** (p. 92), and **OGRLayer** (p. 160) classes).

On application startup registration functions are normally called for each desired file format. These functions instantiate the appropriate **OGRSFDriver** (p. 217) objects, and register them with the **OGRSFDriverRegistrar** (p. 220). When a data source is to be opened, the registrar will normally try each **OGRSFDriver** (p. 217) in turn, until one succeeds, returning an **OGRDataSource** (p. 92) object.

It is not intended that the **OGRSFDriverRegistrar** (p. 220) be derived from.

Chapter 4

OGR Driver Implementation Tutorial

4.1 Overall Approach

In general new formats are added to OGR by implementing format specific drivers with subclasses of **OGRSFDriver** (p. 217), **OGRDataSource** (p. 92) and **OGRLayer** (p. 160). The **OGRSFDriver** (p. 217) subclass is registered with the **OGRSFDriverRegistrar** (p. 220) at runtime.

Before following this tutorial to implement an OGR driver, please review the `OGR Architecture` document carefully.

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

4.2 Contents

4.3 Implementing OGRSFDriver

The format specific driver class is implemented as a subclass of **OGRSFDriver** (p. 217). One instance of the driver will normally be created, and registered with the OGRSFDriverRegistrar(). The instantiation of the driver is normally handled by a global C callable registration function, similar to the following placed in the same file as the driver class.

```
void RegisterOGRSPF()

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

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

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

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

The constructor generally does nothing. The **OGRSFDriver::GetName()** (p. 218) method returns a static string with the name of the driver. This name is specified on the commandline when creating datasources so it is generally good to keep it short and without any special characters or spaces.

```
OGRSPFDriver::~OGRSPFDriver()
```

```
{
}

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

The Open() method is called by **OGRSFDriverRegistrar::Open()** (p. 221), or from the C API **OGROpen()** (p. 407). The **OGRSFDriver::Open()** (p. 218) method should quietly return NULL if the passed filename is not of the format supported by the driver. If it is the target format, then a new **OGRDataSource** (p. 92) object for the datasource should be returned.

Is is common for the Open() method to be delegated to an Open() method on the actual format's **OGRDataSource** (p. 92) class. In the case of the SPF format update in place is not supported, so we always fail if bUpdate is FALSE.

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

    OGRSPFDataSource   *poDS = new OGRSPFDataSource();

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

In OGR the capabilities of drivers, datasources and layers are determined by calling TestCapability() on the various objects with names strings representing specific optional capabilities. For the driver the only two capabilities currently tested for are the ability to create datasources and to delete them. In our first pass as a read only SPF driver, these are both disabled. The default return value for unrecognised capabilities should always be FALSE, and the symbolic defines for capability names (defined in **ogr_core.h** (p. 409)) should be used instead of the literal strings to avoid typos.

```
int OGRSPFDriver::TestCapability( const char * pszCap )

{
    if( EQUAL(pszCap,ODrCCreateDataSource) )
        return FALSE;
    else if( EQUAL(pszCap,ODrCDeleteDataSource) )
        return FALSE;
    else
        return FALSE;
}
```

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

4.4 Basic Read Only Data Source

We will start implementing a minimal read-only datasource. No attempt is made to optimize operations, and default implementations of many methods inherited from **OGRDataSource** (p. 92) are used.

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

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

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

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

    OGRSPFLayer       **papoLayers;
    int                 nLayers;

  public:
                        OGRSPFDataSource();
                        ~OGRSPFDataSource();

    int                 Open( const char * pszFilename );

    const char          *GetName() { return pszName; }

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

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

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

```
OGRSPFDataSource::OGRSPFDataSource()

{
    papoLayers = NULL;
    nLayers = 0;

    pszName = NULL;
}

OGRSPFDataSource::~OGRSPFDataSource()

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

    CPLFree( pszName );
}
```

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

Note that Open() methods should try and determine that a file isn't of the identified format as efficiently as possible, since many drivers may be invoked with files of the wrong format before the correct driver

is reached. In this particular Open() we just test the file extension but this is generally a poor way of identifying a file format. If available, checking "magic header values" or something similar is preferrable.

```
int  OGRSPFDataSource::Open( const char *pszFilename )

{
// ------------------------------------------------------------------
//      Does this appear to be an .spf file?
// ------------------------------------------------------------------
    if( !EQUAL( CPLGetExtension(pszFilename), "spf" ) )
        return FALSE;

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

    papoLayers[0] = new OGRSPFLayer( pszFilename );

    pszName = CPLStrdup( pszFilename );

    return TRUE;
}
```

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

```
OGRLayer *OGRSPFDataSource::GetLayer( int iLayer )

{
    if( iLayer < 0 || iLayer >= nLayers )
        return NULL;
    else
        return papoLayers[iLayer];
}
```

4.5 Read Only Layer

The OGRSPFLayer is implements layer semantics for an .spf file. It provides access to a set of feature objects in a constistent 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. 116) 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. 116). As OGRFeature's for this layer will also take a reference to this definition it is important that we also establish a reference on behalf of the layer itself.

```
OGRSPFLayer::OGRSPFLayer( const char *pszFilename )

{
    nNextFID = 0;

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

    OGRFieldDefn oFieldTemplate( "Name", OFTString );

    poFeatureDefn->AddFieldDefn( &oFieldTemplate );

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

Note that the destructor uses Release() on the **OGRFeatureDefn** (p. 116). 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. 160) implementations. It is responsible for reading the next feature according to the current spatial and attribute filters installed.

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

```
OGRFeature *OGRSPFLayer::GetNextFeature()

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

        pszLine = CPLReadLineL( fp );

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

```
        double dfX;
        double dfY;

        dfX = atof(pszLine);

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

        dfY = atof(pszLine);

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

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

```
        OGRFeature *poFeature = new OGRFeature( poFeatureDefn );

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

Next we check if the feature matches our current attribute or spatial filter if we have them. Methods on the **OGRLayer** (p. 160) base class support maintain filters in the **OGRLayer** (p. 160) member fields m_poFilterGeom (spatial filter) and m_poAttrQuery (attribute filter) so we can just use these values here if they are non-NULL. The following test is essentially "stock" and done the same in all formats. Some formats also do some spatial filtering ahead of time using a spatial index.

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

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

        delete poFeature;
    }
}
```

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

```
void OGRSPFLayer::ResetReading()

{
    VSIFSeekL( fp, 0, SEEK_SET );
    nNextFID = 0;
}
```

In this implementation we do not provide a custom implementation for the GetFeature() method. This means an attempt to read a particular feature by it's feature id will result in many calls to GetNextFeature()

till the desired feature is found. However, in a sequential text format like spf there is little else we could do anyways.

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

Chapter 5

OGR SQL

The **OGRDataSource** (p. 92) supports executing commands against a datasource via the **OGRDataSource::ExecuteSQL()** (p. 94) 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.

5.1 Supported SQL syntax

OGR SQL supports the following pseudo-syntax:

```
SELECT <field-list> FROM <table_def>
     [LEFT JOIN <table_def>
      ON [<table_ref>.]<key_field> = [<table_ref>.].<key_field>]*
     [WHERE <where-expr>]
     [ORDER BY <sort specification list>]

<field-list> ::= <column-spec> [ { , <column-spec> }... ]

<column-spec> ::= <field-spec> [ <as clause> ]
                  | CAST ( <field-spec> AS <data type> ) [ <as clause> ]

<field-spec> ::= [DISTINCT] <field_ref>
                  | <field_func> ( [DISTINCT] <field-ref> )
                  | Count(*)

<as clause> ::= [ AS ] <column_name>

<data type> ::= character [ ( field_length ) ]
                | float [ ( field_length ) ]
                | numeric [ ( field_length [, field_precision ] ) ]
                | integer [ ( field_length ) ]
                | date [ ( field_length ) ]
                | time [ ( field_length ) ]
                | timestamp [ ( field_length ) ]

<field-func> ::= AVG | MAX | MIN | SUM | COUNT

<field_ref>  ::= [<table_ref>.]field_name

<sort specification list> ::=
              <sort specification> [ { <comma> <sort specification> }... ]

<sort specification> ::= <sort key> [ <ordering specification> ]

<sort key> ::=  <field_ref>

<ordering specification> ::= ASC | DESC

<table_def> ::= ['<datasource name>'.]table_name [table_alias]

<table_ref> ::= table_name | table_alias
```

5.2 SELECT

The SELECT statement is used to fetch layer features (analygous 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 analygous to tables in an RDBMS and feature attributes are analygous to column values. The simpliest 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 equivelent to accessing the layer directly. In this example the "*" is the list of fields to fetch from the layer, with "*" meaning that all fields should be fetched.

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

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

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

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

This select statement will produce a table with just one feature, with one attribute (named something like "count_eas_id") containing the number of distinct values of the eas_id attribute.

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

5.2.1 Field List Operators

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

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

```
SELECT DISTINCT areacode FROM polylayer
```

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

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

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 though it would get the same result as giving it any of the column names. 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
```

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.

5.2.1.1 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 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:

```
SELECT COUNT(areacode) AS 'count' FROM polylayer
```

We can optionally omit the AS keyword in the field name aliases, like:

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

5.2.1.2 Changing the type of the fields

OGR SQL supports changing the type of the columns by using the SQL92 compliant CAST operator according to the following example:

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

Currently casting to the following target types are supported:

1. character(field_length)
2. float(field_length)
3. numeric(field_length, field_precision)
4. integer(field_length)
5. date(field_length)
6. time(field_length)
7. timestamp(field_length)

Specifying the field_length and/or the field_precision is optional. Conversion to the 'integer list', 'double list' and 'string list' OGR data types are not supported, which doesn't conform to the SQL92 specification.

5.2.1.3 Field List Limitations

1. Field arithmetic, and other binary operators are not supported, so you can't do something like:

   ```
   SELECT prop_value / area FROM invoices
   ```

2. Lots of operators are missing.

5.2.2 WHERE

The argument to the WHERE clause is a fairly simplistic logical expression used select records to be selected 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.

A WHERE clause consists of a set of attribute tests. Each basic test is of the form **fieldname operator value**. The **fieldname** is any of the fields in the source layer. The operator is one of **=**, **!=**, <>, <, >, <=, >=, **LIKE** and **ILIKE** and **IN**.

Most of the operators are self explanitory, but is is worth nothing that **!=** is the same as <>, the string equality is case insensitive, but the <, >, <= and >= operators *are* case sensitive. Both the LIKE and ILIKE operators are case insensitive.

The value argument to the **LIKE** operator is a pattern against which the value string is matched. In this pattern percent (%) matches any number of characters, and underscore (_) matches any one character.

```
String             Pattern       Matches?
------             -------       --------
Alberta            ALB%          Yes
Alberta            _lberta       Yes
St. Alberta        _lberta       No
St. Alberta        %lberta       Yes
Robarts St.        %Robarts%     Yes
12345              123%45        Yes
123.45             12?45         No
N0N 1P0            %N0N%         Yes
L4C 5E2            %N0N%         No
```

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

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

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 precidence clear. Some more complicated predicates are:

```
SELECT * FROM poly WHERE (prop_value >= 100000) AND (prop_value < 200000)
SELECT * FROM poly WHERE NOT (area_code LIKE "N0N%")
SELECT * FROM poly WHERE (prop_value IS NOT NULL) AND (prop_value < 100000)
```

5.2.3 WHERE Limitations

1. The left of any comparison operator must be a field name, and the right must be a literal value. Fields cannot currently be compared to fields.
2. Fields must all come from the primary table (the one listed in the FROM clause, and must not have any table prefix ... they must just be the field name.
3. No arithmetric operations are supported. You can't test "WHERE (a+b) < 10" for instance.
4. All string comparisons are case insensitive except for <, >, <= and >=.

5.2.4 ORDER BY

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

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

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

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

5.2.5 JOINs

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

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

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

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

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

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

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

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

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

In the above examples, the *nation* table was found in the same datasource as the *city* table. However, the OGR join support includes the ability to join against a table in a different data source, potentially of a different format. This is indicated by qualifying the secondary table name with a datasource name. In this case the secondary datasource is opened using normal OGR semantics and utilized to access the secondary table untill 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 ables of the same name are being used from different data sources. For instance, if the actual tables names were messy we might want to do something like:

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

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

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

5.2.6 JOIN Limitations

1. Joins can be very expensive operations if the secondary table is not indexed on the key field being used.

2. Joined fields may not be used in WHERE clauses, or ORDER BY clauses at this time. The join is essentially evaluated after all primary table subsetting is complete, and after the ORDER BY pass.

3. Joined fields may not be used as keys in later joins. So you could not use the province id in a city to lookup the province record, and then use a nation id from the province id to lookup the nation record. This is a sensible thing to want and could be implemented, but is not currently supported.

4. Datasource names for joined tables are evaluated relative to the current processes working directory, not the path to the primary datasource.

5. These are not true LEFT or RIGHT joins in the RDBMS sense. Whether or not a secondary record exists for the join key or not, one and only one copy of the primary record is returned in the result set. If a secondary record cannot be found, the secondary derived fields will be NULL. If more than one matching secondary field is found only the first will be used.

5.3 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 pecedence over the other fields with the same names in the data source.

5.3.1 FID

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

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

5.3.2 OGR_GEOMETRY

Some of the data sources (like MapInfo tab) can handle geometries of different types within the same layer. The **OGR_GEOMETRY** special field represents the geometry type returned by **OGRGeometry::getGeometryName()** (p. 135) and can be used to distinguish the various types. By using this field one can select particular types of the geometries like:

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

5.3.3 OGR_GEOM_WKT

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

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

Using the **OGR_GEOM_WKT** and the **LIKE** operator in the WHERE clause we can get similar effect as using OGR_GEOMETRY:

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

5.3.4 OGR_STYLE

The **OGR_STYLE** special field represents the style string of the feature returned by **OGRFeature::GetStyleString()** (p. 109). By using this field and the **LIKE** operator the result of the query can be filtered by the style. For example we can select the annotation features as:

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

5.4 CREATE INDEX

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

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

5.4.1 Index Limitations

1. Indexes are not maintained dynamically when new features are added to or removed from a layer.
2. Very long strings (longer than 256 characters?) cannot currently be indexed.
3. To recreate an index it is necessary to drop all indexes on a layer and then recreate all the indexes.
4. Indexes are not used in any complex queries. Currently the only query the will accelerate is a simple "field = value" query.

5.5 DROP INDEX

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

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

5.6 ExecuteSQL()

SQL is executed against an **OGRDataSource** (p. 92), not against a specific layer. The call looks like this:

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

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

The poSpatialFilter argument is a geometry used to select a bounding rectangle for features to be returned in a manner similar to the **OGRLayer::SetSpatialFilter()** (p. 167) method. It may be NULL for no special spatial restriction.

The result of an ExecuteSQL() call is usually a temporary **OGRLayer** (p. 160) representing the results set from the statement. This is the case for a SELECT statement for instance. The returned temporary layer should be released with OGRDataSource::ReleaseResultsSet() method when no longer needed. Failure to release it before the datasource is destroyed may result in a crash.

5.7 Non-OGR SQL

All OGR drivers for database systems: `MySQL`, PostgreSQL and PostGIS (`PG`), Oracle (`OCI`), `SQLite`, `ODBC` and ESRI Personal Geodatabase (`PGeo`) override the **OGRDataSource::ExecuteSQL()** (p. 94) function with dedicated implementation and, by default, pass the SQL statements directly to the underlying RDBMS. In these cases the SQL syntax varies in some particulars from OGR SQL. Also, anything possible in SQL can then be accomplished for these particular databases. Only the result of SQL WHERE statements will be returned as layers.

Chapter 6

OGR Utility Programs

The following utilities are distributed as part of the OGR Simple Features toolkit:

- **ogrinfo** (p. 37)
- **ogr2ogr** (p. 41)
- **ogrtindex** (p. 45)

Chapter 7

ogrinfo

lists information about an OGR supported data source

```
ogrinfo [-ro] [-q] [-where restricted_where]
               [-spat xmin ymin xmax ymax] [-fid fid]
               [-sql statement] [-al] [-so] [--formats]
               datasource_name [layer [layer ...]]
```

The ogrinfo program lists various information about an OGR supported data source to stdout (the terminal).

-ro: Open the data source in read-only mode.

-al: List all features of all layers (used instead of having to give layer names as arguments).

-so: Summary Only: supress listing of features, show only the summary information like projection, schema, feature count and extents.

-q: Quiet verbose reporting of various information, including coordinate system, layer schema, extents, and feature count.

-where *restricted_where*: An attribute query in a restricted form of the queries used in the SQL WHERE statement. Only features matching the attribute query will be reported.

-sql *statement*: Execute the indicated statement and return the result.

-spat *xmin ymin xmax ymax*: The area of interest. Only features within the rectangle will be reported.

-fid *fid*: If provided, only the feature with this feature id will be reported. Operates exclusive of the spatial or attribute queries.

–formats: List the format drivers that are enabled.

***datasource_name*:** The data source to open. May be a filename, directory or other virtual name. See the `OGR Vector Formats` list for supported datasources.

***layer*:** One or more layer names may be reported.

If no layer names are passed then ogrinfo will report a list of available layers (and their layerwide geometry type). If layer name(s) are given then their extents, coordinate system, feature count, geometry type, schema and all features matching query parameters will be reported to the terminal. If no query parameters are provided, all features are reported.

Geometries are reported in OGC WKT format.

Example reporting all layers in an NTF file:

```
% ogrinfo wrk/SHETLAND_ISLANDS.NTF
INFO: Open of `wrk/SHETLAND_ISLANDS.NTF'
using driver `UK .NTF' successful.
1: BL2000_LINK (Line String)
2: BL2000_POLY (None)
3: BL2000_COLLECTIONS (None)
4: FEATURE_CLASSES (None)
```

Example using an attribute query is used to restrict the output of the features in a layer:

```
% ogrinfo -ro -where 'GLOBAL_LINK_ID=185878' wrk/SHETLAND_ISLANDS.NTF BL2000_LINK
INFO: Open of `wrk/SHETLAND_ISLANDS.NTF'
using driver `UK .NTF' successful.

Layer name: BL2000_LINK
```

```
Geometry: Line String
Feature Count: 1
Extent: (419794.100000, 1069031.000000) - (419927.900000, 1069153.500000)
Layer SRS WKT:
PROJCS["OSGB 1936 / British National Grid",
    GEOGCS["OSGB 1936",
        DATUM["OSGB_1936",
            SPHEROID["Airy 1830",6377563.396,299.3249646]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",49],
    PARAMETER["central_meridian",-2],
    PARAMETER["scale_factor",0.999601272],
    PARAMETER["false_easting",400000],
    PARAMETER["false_northing",-100000],
    UNIT["metre",1]]
LINE_ID: Integer (6.0)
GEOM_ID: Integer (6.0)
FEAT_CODE: String (4.0)
GLOBAL_LINK_ID: Integer (10.0)
TILE_REF: String (10.0)
OGRFeature(BL2000_LINK):2
  LINE_ID (Integer) = 2
  GEOM_ID (Integer) = 2
  FEAT_CODE (String) = (null)
  GLOBAL_LINK_ID (Integer) = 185878
  TILE_REF (String) = SHETLAND I
  LINESTRING (419832.100 1069046.300,419820.100 1069043.800,419808.300
  1069048.800,419805.100 1069046.000,419805.000 1069040.600,419809.400
  1069037.400,419827.400 1069035.600,419842 1069031,419859.000
  1069032.800,419879.500 1069049.500,419886.700 1069061.400,419890.100
  1069070.500,419890.900 1069081.800,419896.500 1069086.800,419898.400
  1069092.900,419896.700 1069094.800,419892.500 1069094.300,419878.100
  1069085.600,419875.400 1069087.300,419875.100 1069091.100,419872.200
  1069094.600,419890.400 1069106.400,419907.600 1069112.800,419924.600
  1069133.800,419927.900 1069146.300,419927.600 1069152.400,419922.600
  1069153.500,419917.100 1069153.500,419911.500 1069153.000,419908.700
  1069152.500,419903.400 1069150.800,419898.800 1069149.400,419894.800
  1069149.300,419890.700 1069149.400,419890.600 1069149.400,419880.800
  1069149.800,419876.900 1069148.900,419873.100 1069147.500,419870.200
  1069146.400,419862.100 1069143.000,419860 1069142,419854.900
  1069138.600,419850 1069135,419848.800 1069134.100,419843
  1069130,419836.200 1069127.600,419824.600 1069123.800,419820.200
  1069126.900,419815.500 1069126.900,419808.200 1069116.500,419798.700
  1069117.600,419794.100 1069115.100,419796.300 1069109.100,419801.800
  1069106.800,419805.000  1069107.300)
```

Chapter 8

ogr2ogr

converts simple features data between file formats

```
Usage: ogr2ogr [-skipfailures] [-append] [-update] [-f format_name] [-gt n]
               [-select field_list] [-where restricted_where]
               [-sql <sql statement>] [--help-general]
               [-spat xmin ymin xmax ymax] [-preserve_fid] [-fid FID]
               [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def]
               [[-dsco NAME=VALUE] ...] dst_datasource_name
               src_datasource_name
               [-lco NAME=VALUE] [-nln name] [-nlt type] [layer [layer ...]]
```

This program can be used to convert simple features data between file formats performing various operations during the process such as spatial or attribute selections, reducing the set of attributes, setting the output coordinate system or even reprojecting the features during translation.

-f*format_name*: output file format name, some possible values are:

```
-f "ESRI Shapefile"
-f "TIGER"
-f "MapInfo File"
-f "GML"
-f "PostgreSQL"
```

-append: Append to existing layer instead of creating new

-overwrite: Delete the output layer and recreate it empty

-update: Open existing output datasource in update mode rather than trying to create a new one

-select*field_list*: Comma-delimited list of fields from input layer to copy to the new layer (defaults to all)

-sql *sql_statement*: SQL statement to execute. The resulting table/layer will be saved to the output.

-where*restricted_where*: Attribute query (like SQL WHERE)

-skipfailures: Continue after a failure, skipping the failed feature.

-gt *n*: group *n* features per transaction (default 200)

-spat*xmin ymin xmax ymax*: spatial query extents

-dsco *NAME=VALUE*: Dataset creation option (format specific)

-lco*NAME=VALUE*: Layer creation option (format specific)

-nln*name*: Assign an alternate name to the new layer

-nlt*type*: Define the geometry type for the created layer. One of NONE, GEOMETRY, POINT, LINESTRING, POLYGON, GEOMETRYCOLLECTION, MULTIPOINT, MULTIPOLYGON or MULTILINESTRING. Add "25D" to the name to get 2.5D versions.

-a_srs*srs_def*: Assign an output SRS

-t_srs*srs_def*: Reproject/transform to this SRS on output

-s_srs*srs_def*: Override source SRS

-fid *fid*: If provided, only the feature with this feature id will be reported. Operates exclusive of the spatial or attribute queries.

Srs_def can be a full WKT definition (hard to escape properly), or a well known definition (ie. EPSG:4326) or a file with a WKT definition.

Example appending to an existing layer (both flags need to be used):

```
% ogr2ogr -update -append -f PostgreSQL PG:dbname=warmerda abc.tab
```

More examples are given in the individual format pages.

Chapter 9

ogrtindex

creates a tileindex

```
ogrtindex [-lnum n]... [-lname name]... [-f output_format]
          [-write_absolute_path] [-skip_different_projection]
                 output_dataset src_dataset...
```

The ogrtindex program can be used to create a tileindex - a file containing a list of the identities of a bunch of other files along with there spatial extents. This is primarily intended to be used with the UMN MapServer for tiled access to layers using the OGR connection type.

-lnum *n*: Add layer number 'n' from each source file in the tile index.

-lname *name*: Add the layer named 'name' from each source file in the tile index.

-f *output_format*: Select an output format name. The default is to create a shapefile.

-tileindex *field_name*: The name to use for the dataset name. Defaults to LOCATION.

-write_absolute_path: Filenames are written with absolute paths

-skip_different_projection: Only layers with same projection ref as layers already inserted in the tileindex will be inserted.

If no -lnum or -lname arguments are given it is assumed that all layers in source datasets should be added to the tile index as independent records.

If the tile index already exists it will be appended to, otherwise it will be created.

It is a flaw of the current ogrtindex program that no attempt is made to copy the coordinate system definition from the source datasets to the tile index (as is expected by MapServer when PROJECTION AUTO is in use).

This example would create a shapefile (tindex.shp) containing a tile index of the BL2000_LINK layers in all the NTF files in the wrk directory:

```
% ogrtindex tindex.shp wrk/*.NTF
```

Chapter 10

OGR Projections Tutorial

10.1 Introduction

The **OGRSpatialReference** (p. 224), and **OGRCoordinateTransformation** (p. 88) classes provide services to represent coordinate systems (projections and datums) and to transform between them. These services are loosely modelled on the OpenGIS Coordinate Transformations specification, and use the same Well Known Text format for describing coordinate systems.

Some background on OpenGIS coordinate systems and services can be found in the Simple Features for COM, and Spatial Reference Systems Abstract Model documents available from `www.opengis.org`. The GeoTIFF Projections Transform List (`http://www.remotesensing.org/geotiff/proj_list`) may also be of assistance in understanding formulations of projections in WKT. The `EPSG` Geodesy web page is also a useful resource.

10.2 Defining a Geographic Coordinate System

Coordinate systems are encapsulated in the **OGRSpatialReference** (p. 224) class. There are a number of ways of initializing an **OGRSpatialReference** (p. 224) 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. 224) has built in support for a few well known coordinate systems, which include "NAD27", "NAD83", "WGS72" and "WGS84" which can be defined in a single call to SetWellKnownGeogCS().

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

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

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

For serialization, and transmission of projection definitions to other packages, the OpenGIS Well Known Text format for coordinate systems is used. An **OGRSpatialReference** (p. 224) 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. 250) method can be used to set an **OGRSpatialReference** (p. 224) from a WKT coordinate system definition.

10.3 Defining a Projected Coordinate System

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

10.4 Querying Coordinate System

Once an **OGRSpatialReference** (p. 224) 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. 251) and **OGRSpatialReference::IsGeographic()** (p. 251) methods. The **OGRSpatialReference::GetSemiMajor()** (p. 238), **OGRSpatialReference::GetSemiMinor()** (p. 239) and **OGRSpatialReference::GetInvFlattening()** (p. 236) methods can be used to get information about the spheroid. The **OGRSpatialReference::GetAttrValue()** (p. 234) method can be used to get the PROJCS, GEOGCS, DATUM, SPHEROID, and PROJECTION names strings. The **OGRSpatialReference::GetProjParm()** (p. 238) method can be used to get the projection parameters. The **OGRSpatialReference::GetLinearUnits()** (p. 237) method can be used to fetch the linear units type, and translation to meters.

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

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

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

```
            poSRS->GetProjParm(SRS_PP_FALSE_NORTHING,0.0) );
}
...
```

10.5 Coordinate Transformation

The **OGRCoordinateTransformation** (p. 88) class is used for translating positions between different coordinate systems. New transformation objects are created using **OGRCreateCoordinateTransformation()** (p. 417), and then the **OGRCoordinateTransformation::Transform()** (p. 88) 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. 417) 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. 417) fails it will return a NULL.

The **OGRCoordinateTransformation::Transform()** (p. 88) method itself can also fail. This may be as a delayed result of one of the above problems, or as a result of an operation being numerically undefined for one or more of the passed in points. The Transform() function will return TRUE on success, or FALSE if any of the points fail to transform. The point array is left in an indeterminate state on error.

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

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

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

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

poLatLong = oUTM.CloneGeogCS();
```

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

    ...

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

10.6 Alternate Interfaces

A C interface to the coordinate system services is defined in **ogr_srs_api.h** (p. 418), 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):
```

10.7 Internal Implementation

The **OGRCoordinateTransformation** (p. 88) service is implemented on top of the `PROJ.4` library originally written by Gerald Evenden of the USGS.

Chapter 11

Directory Hierarchy

11.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 12

Class Index

12.1 Class Hierarchy

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

Chapter 13

Class Index

13.1 Class List

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

Chapter 14

File Index

14.1 File List

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

Chapter 15

Directory Documentation

15.1 ogrsf_frmts/generic/ Directory Reference

Files

- file **ogr_attrind.cpp**
- file **ogr_gensql.cpp**
- file **ogr_gensql.h**
- file **ogr_miattrind.cpp**
- file **ogrdatasource.cpp**
- file **ogrlayer.cpp**
- file **ogrregisterall.cpp**
- file **ogrsfdriver.cpp**
- file **ogrsfdriverregistrar.cpp**

15.2 ogrsf_frmts/ Directory Reference

Directories

- directory **generic**

Files

- file **ogr_attrind.h**
- file **ogrsf_frmts.h**

15.3 /builddir/build/BUILD/gdal-1.6.0-fedora/port/ Directory Reference

Files

- file **cpl_config.h**
- file **cpl_conv.cpp**
- file **cpl_conv.h**
- file **cpl_csv.cpp**
- file **cpl_csv.h**
- file **cpl_error.cpp**
- file **cpl_error.h**
- file **cpl_findfile.cpp**
- file **cpl_getexecpath.cpp**
- file **cpl_hash_set.cpp**
- file **cpl_hash_set.h**
- file **cpl_http.cpp**
- file **cpl_http.h**
- file **cpl_list.cpp**
- file **cpl_list.h**
- file **cpl_minixml.cpp**
- file **cpl_minixml.h**
- file **cpl_minizip_ioapi.cpp**
- file **cpl_minizip_ioapi.h**
- file **cpl_minizip_unzip.cpp**
- file **cpl_minizip_unzip.h**
- file **cpl_multiproc.cpp**
- file **cpl_multiproc.h**
- file **cpl_odbc.cpp**
- file **cpl_odbc.h**
- file **cpl_path.cpp**
- file **cpl_port.h**
- file **cpl_quad_tree.cpp**
- file **cpl_quad_tree.h**
- file **cpl_recode_stub.cpp**
- file **cpl_string.cpp**
- file **cpl_string.h**
- file **cpl_strtod.cpp**
- file **cpl_vsi.h**
- file **cpl_vsi_mem.cpp**
- file **cpl_vsi_virtual.h**
- file **cpl_vsil.cpp**
- file **cpl_vsil_gzip.cpp**
- file **cpl_vsil_simple.cpp**
- file **cpl_vsil_unix_stdio_64.cpp**
- file **cpl_vsil_win32.cpp**
- file **cpl_vsisimple.cpp**
- file **cpl_win32ce_api.cpp**
- file **cpl_win32ce_api.h**
- file **cpl_wince.h**

- file **cplgetsymbol.cpp**
- file **cplkeywordparser.cpp**
- file **cplkeywordparser.h**
- file **cplstring.cpp**
- file **xmlreformat.cpp**

Chapter 16

Class Documentation

16.1 _CPLList Struct Reference

`#include <cpl_list.h>`

Public Attributes

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

16.1.1 Detailed Description

List element structure.

16.1.2 Member Data Documentation

16.1.2.1 void* _CPLList::pData

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

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

16.1.2.2 struct _CPLList* _CPLList::psNext [read]

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

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

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

- **cpl_list.h**

16.2 CPLODBCDriverInstaller Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

- int **InstallDriver** (const char *pszDriver, const char *pszPathIn, WORD fRequest=ODBC_INSTALL_COMPLETE)
- int **RemoveDriver** (const char *pszDriverName, int fRemoveDSN=0)

16.2.1 Detailed Description

A class providing functions to install or remove ODBC driver.

16.2.2 Member Function Documentation

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

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

Parameters:

pszDriver - The driver definition as a list of keyword-value pairs describing the driver (See ODBC API Reference).

pszPathIn - Full path of the target directory of the installation, or a null pointer (for unixODBC, NULL is passed).

fRequest - The fRequest argument must contain one of the following values: ODBC_INSTALL_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.

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

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

Parameters:

pszDriverName - The name of the driver as registered in the Odbcinst.ini key of the system information.

fRemoveDSN - TRUE: Remove DSNs associated with the driver specified in lpszDriver. FALSE: Do not remove DSNs associated with the driver specified in lpszDriver.

Returns:

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

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

- **cpl_odbc.h**
- cpl_odbc.cpp

16.3 CPLODBCSession Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

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

16.3.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

16.3.2 Member Function Documentation

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

Connect to database and logon.

Parameters:

pszDSN The name of the DSN being used to connect. This is not optional.

pszUserid the userid to logon as, may be NULL if not not required, or provided by the DSN.

pszPassword the password to logon with. May be NULL if not required or provided by the DSN.

Returns:

TRUE on success or FALSE on failure. Call **GetLastError()** (p. 70) to get details on failure.

References GetLastError().

16.3.2.2 const char * CPLODBCSession::GetLastError ()

Returns the last ODBC error message.

Returns:

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

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

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

- **cpl_odbc.h**
- cpl_odbc.cpp

16.4 CPLODBCStatement Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

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

Static Public Member Functions

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

16.4.1 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.

16.4.2 Member Function Documentation

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

Append to internal command.

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

Parameters:

dfValue value to append to the command.

References Append().

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

Append to internal command.

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

Parameters:

nValue value to append to the command.

References Append().

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

Append text to internal command.

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

Parameters:

pszText text to append.

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

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

Append text to internal command.

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

Parameters:

pszText text to append.

References Append().

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

Append to internal command.

The passed format is used to format other arguments and the result is appended to the internal command text. Long results may not be formatted properly, and should be appended with the direct **Append()** (p. 72) methods.

Parameters:

pszFormat printf() style format string.

Returns:

FALSE if formatting fails dueto result being too large.

References Append().

16.4.2.6 void CPLODBCStatement::Clear ()

Clear internal command text and result set definitions.

Referenced by ExecuteSQL().

16.4.2.7 void CPLODBCStatement::DumpResult (FILE * *fp*, int *bShowSchema* = 0)

Dump resultset to file.

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

Parameters:

fp the file to write to. stdout or stderr are acceptable.

bShowSchema TRUE to force writing schema information for the rowset before the rowset data itself. Default is FALSE.

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

16.4.2.8 int CPLODBCStatement::ExecuteSQL (const char * *pszStatement* = 0)

Execute an SQL statement.

This method will execute the passed (or stored) SQL statement, and initialize information about the resultset if there is one. If a NULL statement is passed, the internal stored statement that has been previously set via **Append()** (p. 72) or **Appendf()** (p. 72) calls will be used.

Parameters:

pszStatement the SQL statement to execute, or NULL if the internally saved one should be used.

Returns:

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

References Append(), and Clear().

16.4.2.9 int CPLODBCStatement::Fetch (int *nOrientation* = SQL_FETCH_NEXT, int *nOffset* = 0)

Fetch a new record.

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

Currently it isn't clear how to determine whether an error or a normal out of data condition has occured if **Fetch()** (p. 73) fails.

Parameters:

nOrientation One of SQL_FETCH_NEXT, SQL_FETCH_LAST, SQL_FETCH_PRIOR, SQL_-FETCH_ABSOLUTE, or SQL_FETCH_RELATIVE (default is SQL_FETCH_NEXT).

nOffset the offset (number of records), ignored for some orientations.

Returns:

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

References CPLODBCSession::GetLastError(), and GetTypeMapping().

Referenced by DumpResult().

16.4.2.10 int CPLODBCStatement::GetColCount ()

Fetch the resultset column count.

Returns:

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

Referenced by DumpResult().

16.4.2.11 const char * CPLODBCStatement::GetColData (const char * *pszColName*, const char * *pszDefault* = 0)

Fetch column data.

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

Parameters:

pszColName the name of the column requested.

pszDefault the value to return if the column does not exist, or is NULL. Defaults to NULL.

Returns:

pointer to internal column data or NULL on failure.

References GetColData(), and GetColId().

16.4.2.12 const char * CPLODBCStatement::GetColData (int *iCol*, const char * *pszDefault* = 0)

Fetch column data.

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

Parameters:

iCol the zero based column to fetch.

pszDefault the value to return if the column does not exist, or is NULL. Defaults to NULL.

Returns:

pointer to internal column data or NULL on failure.

Referenced by DumpResult(), and GetColData().

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

Fetch column index.

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

Parameters:

pszColName the name to search for.

Returns:

the column index, or -1 if not found.

Referenced by GetColData().

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

Fetch a column name.

Parameters:

iCol the zero based column index.

Returns:

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

Referenced by DumpResult().

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

Fetch the column nullability.

Parameters:

iCol the zero based column index.

Returns:

TRUE if the column may contains or FALSE otherwise.

Referenced by DumpResult().

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

Fetch the column precision.

Parameters:

iCol the zero based column index.

Returns:

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

Referenced by DumpResult().

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

Fetch the column width.

Parameters:

iCol the zero based column index.

Returns:

column width, zero for unknown width columns.

Referenced by DumpResult().

16.4.2.18 short CPLODBCStatement::GetColType (int *iCol*)

Fetch a column data type.

The return type code is a an ODBC SQL_ code, one of SQL_UNKNOWN_TYPE, SQL_CHAR, SQL_-NUMERIC, SQL_DECIMAL, SQL_INTEGER, SQL_SMALLINT, SQL_FLOAT, SQL_REAL, SQL_-DOUBLE, SQL_DATETIME, SQL_VARCHAR, SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_-TIMESTAMPT.

Parameters:

iCol the zero based column index.

Returns:

type code or -1 if the column is illegal.

Referenced by DumpResult().

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

Fetch a column data type name.

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

Parameters:

iCol the zero based column index.

Returns:

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

16.4.2.20 int CPLODBCStatement::GetColumns (const char * *pszTable*, const char * *pszCatalog* = 0, const char * *pszSchema* = 0)

Fetch column definitions for a table.

The SQLColumn() method is used to fetch the definitions for the columns of a table (or other queriable object such as a view). The column definitions are digested and used to populate the **CPLODBCStatement** (p. 71) column definitions essentially as if a "SELECT * FROM tablename" had been done; however, no resultset will be available.

Parameters:

pszTable the name of the table to query information on. This should not be empty.

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

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

Returns:

TRUE on success or FALSE on failure.

16.4.2.21 int CPLODBCStatement::GetPrimaryKeys (const char * *pszTable*, const char * *pszCatalog* = 0, const char * *pszSchema* = 0)

Fetch primary keys for a table.

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

Parameters:

pszTable the name of the table to query information on. This should not be empty.

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

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

Returns:

TRUE on success or FALSE on failure.

16.4.2.22 int CPLODBCStatement::GetTables (const char * *pszCatalog* = 0, const char * *pszSchema* = 0)

Fetch tables in database.

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

Parameters:

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

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

Returns:

TRUE on success or FALSE on failure.

16.4.2.23 SQLSMALLINT CPLODBCStatement::GetTypeMapping (SQLSMALLINT *nTypeCode*) [static]

Get appropriate C data type for SQL column type.

Returns a C data type code, corresponding to the indicated SQL data type code (as returned from **CPLODBCStatement::GetColType()** (p. 76)).

Parameters:

nTypeCode the SQL_ code, such as SQL_CHAR.

Returns:

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

Referenced by Fetch().

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

Get name for SQL column type.

Returns a string name for the indicated type code (as returned from **CPLODBCStatement::GetColType()** (p. 76)).

Parameters:

nTypeCode the SQL_ code, such as SQL_CHAR.

Returns:

internal string, "UNKNOWN" if code not recognised.

Referenced by DumpResult().

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

- **cpl_odbc.h**
- cpl_odbc.cpp

16.5 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.

16.5.1 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. 314). The structure contents should not normally be altered by application code, but may be freely examined by application code.

Using the psChild and psNext pointers, a heirarchical tree structure for a document can be represented as a tree of **CPLXMLNode** (p. 79) structures.

16.5.2 Member Data Documentation

16.5.2.1 CPLXMLNodeType CPLXMLNode::eType

Node type.

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

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

16.5.2.2 struct CPLXMLNode* CPLXMLNode::psChild [read]

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().

16.5.2.3 struct CPLXMLNode* CPLXMLNode::psNext [read]

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(), and CPLStripXMLNamespace().

16.5.2.4 char* CPLXMLNode::pszValue

Node value.

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

For CXT_Attribute the pszValue is the attribute name. The value of the attribute will be a CXT_Text child.

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

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

For CXT_Comment the value is all the literal text within the comment, but not including the comment start/end indicators ("<–" and "- → ").

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

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

- **cpl_minixml.h**

16.6 OGR_SRSNode Class Reference

`#include <ogr_spatialref.h>`

Public Member Functions

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

16.6.1 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. 224) object, which uses the **OGR_SRSNode** (p. 81) to implement it's data structure; however, this class is user accessable for detailed access to components of an SRS definition.

16.6.2 Constructor & Destructor Documentation

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

Constructor.

Parameters:

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

Referenced by Clone(), and importFromWkt().

16.6.3 Member Function Documentation

16.6.3.1 void OGR_SRSNode::AddChild (OGR_SRSNode * *poNew*)

Add passed node as a child of target node.

Note that ownership of the passed node is assumed by the node on which the method is invoked ... use the **Clone()** (p. 82) method if the original is to be preserved. New children are always added at the end of the list.

Parameters:

poNew the node to add as a child.

References InsertChild().

Referenced by Clone(), importFromWkt(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetLinearUnits(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), and OGRSpatialReference::SetTOWGS84().

16.6.3.2 OGRErr OGR_SRSNode::applyRemapper (const char * *pszNode*, char ** *papszSrcValues*, char ** *papszDstValues*, int *nStepSize* = 1, int *bChildOfHit* = FALSE)

Remap node values matching list.

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

Parameters:

pszNode Restrict remapping to children of this type of node (eg. "PROJECTION")

papszSrcValues a NULL terminated array of source string. If the node value matches one of these (case insensitive) then replacement occurs.

papszDstValues an array of destination strings. On a match, the one corresponding to a source value will be used to replace a node.

nStepSize increment when stepping through source and destination arrays, allowing source and destination arrays to be one interleaved array for instances. Defaults to 1.

bChildOfHit Only TRUE if we the current node is the child of a match, and so needs to be set. Application code would normally pass FALSE for this argument.

Returns:

returns OGRERR_NONE unless something bad happens. There is no indication returned about whether any replacement occured.

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

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

16.6.3.3 OGR_SRSNode * OGR_SRSNode::Clone () const

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

Returns:

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

References AddChild(), and OGR_SRSNode().

Referenced by OGRSpatialReference::Clone(), and OGRSpatialReference::CopyGeogCSFrom().

16.6.3.4 void OGR_SRSNode::DestroyChild (int *iChild*)

Remove a child node, and it's subtree.

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

Parameters:

iChild the index of the child.

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

16.6.3.5 OGRErr OGR_SRSNode::exportToWkt (char ** *ppszResult*) const

Convert this tree of nodes into WKT format.

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

Parameters:

ppszResult the resulting string is returned in this pointer.

Returns:

currently OGRERR_NONE is always returned, but the future it is possible error conditions will develop.

References exportToWkt().

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

16.6.3.6 int OGR_SRSNode::FindChild (const char * *pszValue*) const

Find the index of the child matching the given string.

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

Parameters:

pszValue the node value being searched for.

Returns:

the child index, or -1 on failure.

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

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

Fetch requested child.

Parameters:

iChild the index of the child to fetch, from 0 to **GetChildCount()** (p. 84) - 1.

Returns:

a pointer to the child **OGR_SRSNode** (p. 81), or NULL if there is no such child.

Referenced by applyRemapper(), OGRSpatialReference::EPSGTreatsAsLatLong(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAttrValue(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::GetAxis(), OGRSpatialReference::GetExtension(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetLinearUnits(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::IsSame(), MakeValueSafe(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetLinearUnits(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), StripNodes(), and OGRSpatialReference::Validate().

16.6.3.8 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::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAttrValue(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::GetAxis(), OGRSpatialReference::GetExtension(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetLinearUnits(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::IsSame(), MakeValueSafe(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTOWGS84(), StripNodes(), and OGRSpatialReference::Validate().

16.6.3.9 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. 85) on a subtree to be more specific.

Parameters:

pszName the name of the node to search for.

Returns:

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

References GetNode().

Referenced by OGRSpatialReference::GetAttrNode(), GetNode(), and OGRSpatialReference::Validate().

16.6.3.10 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::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAttrValue(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::GetAxis(), OGRSpatialReference::GetExtension(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetLinearUnits(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::IsProjected(), OGRSpatialReference::IsSame(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::StripCTParms(), and OGRSpatialReference::Validate().

16.6.3.11 OGRErr OGR_SRSNode::importFromWkt (char ** *ppszInput*)

Import from WKT string.

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

Parameters:

ppszInput Pointer to pointer to input. The pointer is updated to point to remaining unused input text.

Returns:

OGRERR_NONE if import succeeds, or OGRERR_CORRUPT_DATA if it fails for any reason.

References AddChild(), importFromWkt(), OGR_SRSNode(), and SetValue().

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

16.6.3.12 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. 82) method if the original is to be preserved. All existing children at location iChild and beyond are push down one space to make space for the new child.

Parameters:

poNew the node to add as a child.

iChild position to insert, use 0 to insert at the beginning.

References poParent.

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

16.6.3.13 void OGR_SRSNode::MakeValueSafe ()

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

The operation is also applies to all subnodes of the current node.

References GetChild(), GetChildCount(), and MakeValueSafe().

Referenced by MakeValueSafe().

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

Set the node value.

Parameters:

pszNewValue the new value to assign to this node. The passed string is duplicated and remains the responsibility of the caller.

Referenced by applyRemapper(), importFromWkt(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetLinearUnits(), OGRSpatialReference::SetNode(), and OGRSpatialReference::SetProjParm().

16.6.3.15 void OGR_SRSNode::StripNodes (const char * *pszName*)

Strip child nodes matching name.

Removes any decendent nodes of this node that match the given name. Of course children of removed nodes are also discarded.

Parameters:

pszName the name for nodes that should be removed.

References DestroyChild(), FindChild(), GetChild(), GetChildCount(), and StripNodes().

Referenced by OGRSpatialReference::importFromEPSG(), OGRSpatialReference::StripCTParms(), and StripNodes().

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

- **ogr_spatialref.h**
- ogr_srsnode.cpp

16.7 OGRCoordinateTransformation Class Reference

`#include <ogr_spatialref.h>`

Inherited by OGRProj4CT.

Public Member Functions

- virtual **OGRSpatialReference** * **GetSourceCS** ()=0
- virtual **OGRSpatialReference** * **GetTargetCS** ()=0
- virtual int **Transform** (int nCount, double *x, double *y, double *z=NULL)=0
- virtual int **TransformEx** (int nCount, double *x, double *y, double *z=NULL, int *pabSuccess=NULL)=0

16.7.1 Detailed Description

Object for transforming between coordinate systems.

Also, see OGRCreateSpatialReference() for creating transformations.

16.7.2 Member Function Documentation

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

Fetch internal source coordinate system.

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

Fetch internal target coordinate system.

Referenced by OGRPolygon::transform(), OGRPoint::transform(), OGRLineString::transform(), and OGRGeometryCollection::transform().

16.7.2.3 virtual int OGRCoordinateTransformation::Transform (int *nCount*, double * *x*, double * *y*, double * *z* = NULL) [pure virtual]

Transform points from source to destination space.

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

The method **TransformEx()** (p. 89) allows extended success information to be captured indicating which points failed to transform.

Parameters:

nCount number of points to transform.

x array of nCount X vertices, modified in place.

y array of nCount Y vertices, modified in place.

z array of nCount Z vertices, modified in place.

Returns:

TRUE on success, or FALSE if some or all points fail to transform.

Referenced by OGRPoint::transform(), and OGRLineString::transform().

16.7.2.4 virtual int OGRCoordinateTransformation::TransformEx (int *nCount*, double * *x*, double * *y*, double * *z* = NULL, int * *pabSuccess* = NULL) [pure virtual]

Transform points from source to destination space.

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

Parameters:

nCount number of points to transform.

x array of nCount X vertices, modified in place.

y array of nCount Y vertices, modified in place.

z array of nCount Z vertices, modified in place.

pabSuccess array of per-point flags set to TRUE if that point transforms, or FALSE if it does not.

Returns:

TRUE if some or all points transform successfully, or FALSE if if none transform.

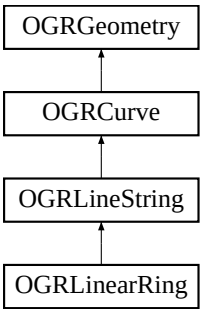
The documentation for this class was generated from the following file:

- **ogr_spatialref.h**

16.8 OGRCurve Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRCurve::



Public Member Functions

- virtual double **get_Length** () const =0
- virtual void **StartPoint** (**OGRPoint** ∗) const =0
- virtual void **EndPoint** (**OGRPoint** ∗) const =0
- virtual int **get_IsClosed** () const
- virtual void **Value** (double, **OGRPoint** ∗) const =0

16.8.1 Detailed Description

Abstract curve base class.

16.8.2 Member Function Documentation

16.8.2.1 void OGRCurve::EndPoint (OGRPoint ∗ *poPoint*) const [pure virtual]

Return the curve end point.

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

Parameters:

poPoint the point to be assigned the end location.

Implemented in **OGRLineString** (p. 177).

Referenced by get_IsClosed().

16.8.2.2 int OGRCurve::get_IsClosed () const [virtual]

Return TRUE if curve is closed.

Tests if a curve is closed. A curve is closed if its start point is equal to its end point.

This method relates to the SFCOM ICurve::get_IsClosed() method.

Returns:

TRUE if closed, else FALSE.

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

16.8.2.3 double OGRCurve::get_Length () const [pure virtual]

Returns the length of the curve.

This method relates to the SFCOM ICurve::get_Length() method.

Returns:

the length of the curve, zero if the curve hasn't been initialized.

Implemented in **OGRLineString** (p. 179).

16.8.2.4 void OGRCurve::StartPoint (OGRPoint * *poPoint*) const [pure virtual]

Return the curve start point.

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

Parameters:

poPoint the point to be assigned the start location.

Implemented in **OGRLineString** (p. 186).

Referenced by get_IsClosed().

16.8.2.5 void OGRCurve::Value (double *dfDistance*, OGRPoint * *poPoint*) const [pure virtual]

Fetch point at given distance along curve.

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

Parameters:

dfDistance distance along the curve at which to sample position. This distance should be between zero and **get_Length()** (p. 91) for this curve.

poPoint the point to be assigned the curve position.

Implemented in **OGRLineString** (p. 186).

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

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

16.9 OGRDataSource Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- virtual const char * **GetName** ()=0
- virtual int **GetLayerCount** ()=0
- virtual **OGRLayer** * **GetLayer** (int)=0
- virtual **OGRLayer** * **GetLayerByName** (const char *)
- virtual OGRErr **DeleteLayer** (int)
- virtual int **TestCapability** (const char *)=0
- virtual **OGRLayer** * **CreateLayer** (const char *pszName, **OGRSpatialReference** *poSpatialRef=NULL, **OGRwkbGeometryType** eGType=wkbUnknown, char **papszOptions=NULL)
- OGRStyleTable * **GetStyleTable** ()
- void **SetStyleTableDirectly** (OGRStyleTable *poStyleTable)
- void **SetStyleTable** (OGRStyleTable *poStyleTable)
- virtual **OGRLayer** * **ExecuteSQL** (const char *pszStatement, **OGRGeometry** *poSpatialFilter, const char *pszDialect)
- virtual void **ReleaseResultSet** (**OGRLayer** *poResultsSet)
- virtual OGRErr **SyncToDisk** ()
- int **Reference** ()
- int **Dereference** ()
- int **GetRefCount** () const
- int **GetSummaryRefCount** () const
- OGRErr **Release** ()
- **OGRSFDriver** * **GetDriver** () const
- void **SetDriver** (**OGRSFDriver** *poDriver)

Friends

- class **OGRSFDriverRegistrar**

16.9.1 Detailed Description

This class represents a data source. A data source potentially consists of many layers (**OGRLayer** (p. 160)). 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. 92) is destroyed, all it's associated OGRLayers objects are also destroyed.

16.9.2 Member Function Documentation

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

This method attempts to create a new layer on the data source with the indicated name, coordinate system, geometry type. The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

Parameters:

pszName the name for the new layer. This should ideally not match any existing layer on the datasource.

poSpatialRef the coordinate system to use for the new layer, or NULL if no coordinate system is available.

eGType the geometry type for the layer. Use wkbUnknown if there are no constraints on the types geometry to be written.

papszOptions a StringList of name=value options. Options are driver specific.

Returns:

NULL is returned on failure, or a new **OGRLayer** (p. 160) handle on success.

Example:

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

...

        OGRLayer *poLayer;
        char     *papszOptions;

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

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

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

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

Delete the indicated layer from the datasource. If this method is supported the ODsCDeleteLayer capability will test TRUE on the **OGRDataSource** (p. 92).

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

Parameters:

iLayer the index of the layer to delete.

Returns:

OGRERR_NONE on success, or OGRERR_UNSUPPORTED_OPERATION if deleting layers is not supported for this datasource.

16.9.2.3 int OGRDataSource::Dereference ()

Decrement datasource reference count.

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

Returns:

the reference count after decrementing.

Referenced by ExecuteSQL().

16.9.2.4 OGRLayer * OGRDataSource::ExecuteSQL (const char * *pszStatement*, OGRGeometry * *poSpatialFilter*, const char * *pszDialect*) [virtual]

Execute an SQL statement against the data store.

The result of an SQL query is either NULL for statements that are in error, or that have no results set, or an **OGRLayer** (p. 160) pointer representing a results set from the query. Note that this **OGRLayer** (p. 160) is in addition to the layers in the data store and must be destroyed with OGRDataSource::ReleaseResultsSet() before the data source is closed (destroyed).

This method is the same as the C function **OGR_DS_ExecuteSQL()** (p. 350).

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

Parameters:

pszStatement the SQL statement to execute.

poSpatialFilter geometry which represents a spatial filter.

pszDialect allows control of the statement dialect. By default it is assumed to be "generic" SQL, whatever that is.

Returns:

an **OGRLayer** (p. 160) containing the results of the query. Deallocate with ReleaseResultsSet().

References Dereference(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetLayerByName(), OGRLayer::GetLayerDefn(), OGRFieldDefn::GetNameRef(), OGRFieldDefn::GetType(), OFTInteger, OFTReal, and OFTString.

16.9.2.5 OGRSFDriver * OGRDataSource::GetDriver () const

Returns the driver that the dataset was opened with.

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

Returns:

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

Referenced by OGR_Dr_CreateDataSource(), OGR_Dr_Open(), and OGRSFDriverRegistrar::Open().

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

Fetch a layer by index. The returned layer remains owned by the **OGRDataSource** (p. 92) and should not be deleted by the application.

This method is the same as the C function **OGR_DS_GetLayer()** (p. 351).

Parameters:

iLayer a layer number between 0 and **GetLayerCount()** (p. 95)-1.

Returns:

the layer, or NULL if iLayer is out of range or an error occurs.

Referenced by GetLayerByName(), GetSummaryRefCount(), and SyncToDisk().

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

Fetch a layer by name. The returned layer remains owned by the **OGRDataSource** (p. 92) and should not be deleted by the application.

This method is the same as the C function **OGR_DS_GetLayerByName()** (p. 351).

Parameters:

pszLayerName the layer name of the layer to fetch.

Returns:

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

References GetLayer(), GetLayerCount(), OGRLayer::GetLayerDefn(), and OGRFeatureDefn::GetName().

Referenced by ExecuteSQL().

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

Get the number of layers in this data source.

This method is the same as the C function **OGR_DS_GetLayerCount()** (p. 352).

Returns:

layer count.

Referenced by GetLayerByName(), GetSummaryRefCount(), and SyncToDisk().

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

Returns the name of the data source. This string should be sufficient to open the data source if passed to the same **OGRSFDriver** (p. 217) that this data source was opened with, but it need not be exactly the same string that was used to open the data source. Normally this is a filename.

This method is the same as the C function **OGR_DS_GetName()** (p. 352).

Returns:

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

16.9.2.10 int OGRDataSource::GetRefCount () const

Fetch reference count.

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

Returns:

the current reference count for the datasource object itself.

16.9.2.11 void OGRDataSource::GetStyleTable () [inline]

Returns data source style table.

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

Returns:

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

16.9.2.12 int OGRDataSource::GetSummaryRefCount () const

Fetch reference count of datasource and all owned layers.

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

Returns:

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

References GetLayer(), GetLayerCount(), and OGRLayer::GetRefCount().

16.9.2.13 int OGRDataSource::Reference ()

Increment datasource reference count.

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

Returns:

the reference count after incrementing.

Referenced by OGRSFDriverRegistrar::Open().

16.9.2.14 OGRErr OGRDataSource::Release ()

Drop a reference to this datasource, and if the reference count drops to zero close (destroy) the datasource. Internally this actually calls the OGRSFDriverRegistrary::ReleaseDataSource() method. This method is essentially a convenient alias.

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

Returns:

OGRERR_NONE on success or an error code.

References OGRSFDriverRegistrar::GetRegistrar(), and OGRSFDriverRegistrar::ReleaseDataSource().

16.9.2.15 void OGRDataSource::ReleaseResultSet (OGRLayer * *poResultsSet*) [virtual]

Release results of **ExecuteSQL()** (p. 94).

This method should only be used to deallocate OGRLayers resulting from an **ExecuteSQL()** (p. 94) call on the same **OGRDataSource** (p. 92). Failure to deallocate a results set before destroying the **OGRDataSource** (p. 92) may cause errors.

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

Parameters:

poResultsSet the result of a previous **ExecuteSQL()** (p. 94) call.

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

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

Note:

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

Parameters:

poDriver pointer to driver instance associated with the data source.

Referenced by OGR_Dr_CreateDataSource(), and OGR_Dr_Open().

16.9.2.17 void OGRDataSource::SetStyleTable (OGRStyleTable * *poStyleTable*) [inline]

Set data source style table.

This method operate exactly as **OGRDataSource::SetStyleTableDirectly()** (p. 98) except that it does not assume ownership of the passed table.

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

Parameters:

poStyleTable pointer to style table to set

16.9.2.18 void OGRDataSource::SetStyleTableDirectly (OGRStyleTable * *poStyleTable*) [inline]

Set data source style table.

This method operate exactly as **OGRDataSource::SetStyleTable()** (p. 97) except that it assumes ownership of the passed table.

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

Parameters:

poStyleTable pointer to style table to set

16.9.2.19 OGRErr OGRDataSource::SyncToDisk () [virtual]

Flush pending changes to disk.

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

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

The default implementation of this method just calls the **SyncToDisk()** (p. 98) method on each of the layers. Conceptionally, calling **SyncToDisk()** (p. 98) on a datasource should include any work that might be accomplished by calling **SyncToDisk()** (p. 98) on layers in that data source.

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

Returns:

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

References GetLayer(), GetLayerCount(), and OGRLayer::SyncToDisk().

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

Test if capability is available.

One of the following data source capability names can be passed into this method, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

- **ODsCCreateLayer**: True if this datasource can create new layers.

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

This method is the same as the C function **OGR_DS_TestCapability()** (p. 353).

Parameters:

pszCapability the capability to test.

Returns:

TRUE if capability available otherwise FALSE.

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

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

16.10 OGREnvelope Class Reference

`#include <ogr_core.h>`

16.10.1 Detailed Description

Simple container for a bounding region.

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

- **ogr_core.h**

16.11 OGRFeature Class Reference

`#include <ogr_feature.h>`

Public Member Functions

- **OGRFeature** (**OGRFeatureDefn** *)
- **OGRFeatureDefn** * **GetDefnRef** ()
- OGRErr **SetGeometryDirectly** (**OGRGeometry** *)
- OGRErr **SetGeometry** (**OGRGeometry** *)
- **OGRGeometry** * **GetGeometryRef** ()
- **OGRGeometry** * **StealGeometry** ()
- **OGRFeature** * **Clone** ()
- virtual OGRBoolean **Equal** (**OGRFeature** *poFeature)
- int **GetFieldCount** ()
- **OGRFieldDefn** * **GetFieldDefnRef** (int iField)
- int **GetFieldIndex** (const char *pszName)
- int **IsFieldSet** (int iField) const
- void **UnsetField** (int iField)
- **OGRField** * **GetRawFieldRef** (int i)
- int **GetFieldAsInteger** (int i)
- double **GetFieldAsDouble** (int i)
- const char * **GetFieldAsString** (int i)
- const int * **GetFieldAsIntegerList** (int i, int *pnCount)
- const double * **GetFieldAsDoubleList** (int i, int *pnCount)
- char ** **GetFieldAsStringList** (int i) const
- GByte * **GetFieldAsBinary** (int i, int *pnCount)
- int **GetFieldAsDateTime** (int i, int *pnYear, int *pnMonth, int *pnDay, int *pnHour, int *pnMinute, int *pnSecond, int *pnTZFlag)
- void **SetField** (int i, int nValue)
- void **SetField** (int i, double dfValue)
- void **SetField** (int i, const char *pszValue)
- void **SetField** (int i, int nCount, int *panValues)
- void **SetField** (int i, int nCount, double *padfValues)
- void **SetField** (int i, char **papszValues)
- void **SetField** (int i, **OGRField** *puValue)
- void **SetField** (int i, int nCount, GByte *pabyBinary)
- void **SetField** (int i, int nYear, int nMonth, int nDay, int nHour=0, int nMinute=0, int nSecond=0, int nTZFlag=0)
- long **GetFID** ()
- virtual OGRErr **SetFID** (long nFID)
- void **DumpReadable** (FILE *, char **papszOptions=NULL)
- OGRErr **SetFrom** (**OGRFeature** *, int=TRUE)
- virtual const char * **GetStyleString** ()
- virtual void **SetStyleString** (const char *)
- virtual void **SetStyleStringDirectly** (char *)

Static Public Member Functions

- static **OGRFeature** ∗ **CreateFeature** (**OGRFeatureDefn** ∗)
- static void **DestroyFeature** (**OGRFeature** ∗)

16.11.1 Detailed Description

A simple feature, including geometry and attributes.

16.11.2 Constructor & Destructor Documentation

16.11.2.1 OGRFeature::OGRFeature (OGRFeatureDefn ∗ *poDefnIn*)

Constructor

Note that the **OGRFeature** (p. 101) will increment the reference count of it's defining **OGRFeatureDefn** (p. 116). Destruction of the **OGRFeatureDefn** (p. 116) 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. 353).

Parameters:

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

References OGRFeatureDefn::GetFieldCount(), OGRField::nMarker1, OGRField::nMarker2, OGRFeatureDefn::Reference(), and OGRField::Set.

Referenced by Clone(), and CreateFeature().

16.11.3 Member Function Documentation

16.11.3.1 OGRFeature ∗ OGRFeature::Clone ()

Duplicate feature.

The newly created feature is owned by the caller, and will have it's own reference to the **OGRFeatureDefn** (p. 116).

This method is the same as the C function **OGR_F_Clone()** (p. 353).

Returns:

new feature, exactly matching this feature.

References GetFID(), OGRFeatureDefn::GetFieldCount(), GetStyleString(), OGRFeature(), SetFID(), SetField(), SetGeometry(), and SetStyleString().

16.11.3.2 OGRFeature ∗ OGRFeature::CreateFeature (OGRFeatureDefn ∗ *poDefn*) [static]

Feature factory.

This is essentially a feature factory, useful for applications creating features but wanting to ensure they are created out of the OGR/GDAL heap.

This method is the same as the C function **OGR_F_Create()** (p. 353).

Parameters:

poDefn Feature definition defining schema.

Returns:

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

References OGRFeature().

16.11.3.3 void OGRFeature::DestroyFeature (OGRFeature * *poFeature*) [static]

Destroy feature

The feature is deleted, but within the context of the GDAL/OGR heap. This is necessary when higher level applications use GDAL/OGR from a DLL and they want to delete a feature created within the DLL. If the delete is done in the calling application the memory will be freed onto the application heap which is inappropriate.

This method is the same as the C function **OGR_F_Destroy()** (p. 354).

Parameters:

poFeature the feature to delete.

16.11.3.4 void OGRFeature::DumpReadable (FILE * *fpOut*, char ** *papszOptions* = NULL)

Dump this feature in a human readable form.

This dumps the attributes, and geometry; however, it doesn't definition information (other than field types and names), nor does it report the geometry spatial reference system.

This method is the same as the C function **OGR_F_DumpReadable()** (p. 354).

Parameters:

fpOut the stream to write to, such as stdout. If NULL stdout will be used.

References OGRGeometry::dumpReadable(), GetFID(), GetFieldAsString(), GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetFieldTypeName(), OGRFeatureDefn::GetName(), OGRFieldDefn::GetNameRef(), GetStyleString(), OGRFieldDefn::GetType(), and IsFieldSet().

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

Test if two features are the same.

Two features are considered equal if the share them (pointer equality) same **OGRFeatureDefn** (p. 116), 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. 354).

Parameters:

poFeature the other feature to test this one against.

Returns:

TRUE if they are equal, otherwise FALSE.

References OGRGeometry::Equals(), GetDefnRef(), GetFID(), and GetGeometryRef().

16.11.3.6 OGRFeatureDefn * OGRFeature::GetDefnRef () [inline]

Fetch feature definition.

This method is the same as the C function **OGR_F_GetDefnRef()** (p. 355).

Returns:

a reference to the feature definition object.

Referenced by Equal().

16.11.3.7 long OGRFeature::GetFID () [inline]

Get feature identifier.

This method is the same as the C function **OGR_F_GetFID()** (p. 355).

Returns:

feature id or OGRNullFID if none has been assigned.

Referenced by Clone(), DumpReadable(), Equal(), OGRLayer::GetFeature(), GetFieldAsDouble(), GetFieldAsInteger(), and GetFieldAsString().

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

Fetch field value as binary data.

Currently this method only works for OFTBinary fields.

This method is the same as the C function **OGR_F_GetFieldAsBinary()** (p. 355).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

pnBytes location to put the number of bytes returned.

Returns:

the field value. This data is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGRField::Binary, OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OGRField::nCount, OFTBinary, and OGRField::paData.

16.11.3.9 int OGRFeature::GetFieldAsDateTime (int *iField*, int * *pnYear*, int * *pnMonth*, int * *pnDay*, int * *pnHour*, int * *pnMinute*, int * *pnSecond*, int * *pnTZFlag*)

Fetch field value as date and time.

Currently this method only works for OFTDate, OFTTime and OFTDateTime fields.

This method is the same as the C function **OGR_F_GetFieldAsDateTime()** (p. 356).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

int pnYear (including century)

int pnMonth (1-12)

int pnDay (1-31)

int pnHour (0-23)

int pnMinute (0-59)

int pnSecond (0-59)

int pnTZFlag (0=unknown, 1=localtime, 100=GMT, see data model for details)

Returns:

TRUE on success or FALSE on failure.

References OGRField::Date, OGRField::Day, OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::Hour, IsFieldSet(), OGRField::Minute, OGRField::Month, OFTDate, OFTDateTime, OFTTime, OGRField::Second, OGRField::TZFlag, and OGRField::Year.

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

Fetch field value as a double.

OFTString features will be translated using atof(). OFTInteger fields will be cast to double. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR_F_GetFieldAsDouble()** (p. 356).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

Returns:

the field value.

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

Referenced by SetFrom().

16.11.3.11 const double * OGRFeature::GetFieldAsDoubleList (int *iField*, int * *pnCount*)

Fetch field value as a list of doubles.

Currently this method only works for OFTRealList fields.

This method is the same as the C function **OGR_F_GetFieldAsDoubleList()** (p. 357).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

pnCount an integer to put the list count (number of doubles) into.

Returns:

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief. If ∗pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OGRField::nCount, OFTRealList, OGRField::paList, and OGRField::RealList.

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

Fetch field value as integer.

OFTString features will be translated using atoi(). OFTReal fields will be cast to integer. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR_F_GetFieldAsInteger()** (p. 357).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

Returns:

the field value.

References GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTReal, OFTString, and OGRField::Real.

Referenced by SetFrom().

16.11.3.13 const int ∗ OGRFeature::GetFieldAsIntegerList (int *iField*, int ∗ *pnCount*)

Fetch field value as a list of integers.

Currently this method only works for OFTIntegerList fields.

This method is the same as the C function **OGR_F_GetFieldAsIntegerList()** (p. 358).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

pnCount an integer to put the list count (number of integers) into.

Returns:

the field value. This list is internal, and should not be modified, or freed. It's 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(), OGRField::IntegerList, IsFieldSet(), OGRField::nCount, OFTIntegerList, and OGRField::paList.

16.11.3.14 const char * OGRFeature::GetFieldAsString (int *iField*)

Fetch field value as a string.

OFTReal and OFTInteger fields will be translated to string using sprintf(), but not necessarily using the established formatting rules. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR_F_GetFieldAsString()** (p. 358).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

Returns:

the field value. This string is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGRField::Binary, OGRField::Date, OGRField::Day, OGRGeometry::exportToWkt(), GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRGeometry::getGeometryName(), OGRFieldDefn::GetPrecision(), GetStyleString(), OGRFieldDefn::GetType(), OGRFieldDefn::GetWidth(), OGRField::Hour, OGRField::IntegerList, IsFieldSet(), OGRField::Minute, OGRField::Month, OGRField::nCount, OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, OFTTime, OGRField::paData, OGRField::paList, OGRField::RealList, OGRField::Second, OGRField::String, OGRField::StringList, OGRField::TZFlag, and OGRField::Year.

Referenced by DumpReadable(), GetStyleString(), and SetFrom().

16.11.3.15 char ** OGRFeature::GetFieldAsStringList (int *iField*) const

Fetch field value as a list of strings.

Currently this method only works for OFTStringList fields.

The returned list is terminated by a NULL pointer. The number of elements can also be calculated using **CSLCount()** (p. 328).

This method is the same as the C function **OGR_F_GetFieldAsStringList()** (p. 358).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

Returns:

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTStringList, OGRField::paList, and OGRField::StringList.

16.11.3.16 int OGRFeature::GetFieldCount () [inline]

Fetch number of fields on this feature. This will always be the same as the field count for the **OGRFeatureDefn** (p. 116).

This method is the same as the C function **OGR_F_GetFieldCount()** (p. 359).

Returns:

count of fields.

Referenced by DumpReadable(), and SetFrom().

16.11.3.17 OGRFieldDefn * OGRFeature::GetFieldDefnRef (int *iField*) [inline]

Fetch definition for this field.

This method is the same as the C function **OGR_F_GetFieldDefnRef()** (p. 359).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

Returns:

the field definition (from the **OGRFeatureDefn** (p. 116)). This is an internal reference, and should not be deleted or modified.

Referenced by SetFrom().

16.11.3.18 int OGRFeature::GetFieldIndex (const char * *pszName*) [inline]

Fetch the field index given field name.

This is a cover for the **OGRFeatureDefn::GetFieldIndex()** (p. 118) method.

This method is the same as the C function **OGR_F_GetFieldIndex()** (p. 360).

Parameters:

pszName the name of the field to search for.

Returns:

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

Referenced by GetStyleString(), and SetFrom().

16.11.3.19 OGRGeometry * OGRFeature::GetGeometryRef () [inline]

Fetch pointer to feature geometry.

This method is the same as the C function **OGR_F_GetGeometryRef()** (p. 360).

Returns:

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

Referenced by Equal(), OGRLayer::GetExtent(), and SetFrom().

16.11.3.20 OGRField * OGRFeature::GetRawFieldRef (int *iField*) [inline]

Fetch a pointer to the internal field value given the index.

This method is the same as the C function **OGR_F_GetRawFieldRef()** (p. 360).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

Returns:

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

Referenced by SetFrom().

16.11.3.21 const char * OGRFeature::GetStyleString () [virtual]

Fetch style string for this feature.

Set the OGR Feature Style Specification for details on the format of this string, and **ogr_featurestyle.h** (p. **??**) for services available to parse it.

This method is the same as the C function **OGR_F_GetStyleString()** (p. 361).

Returns:

a reference to a representation in string format, or NULL if there isn't one.

References GetFieldAsString(), and GetFieldIndex().

Referenced by Clone(), DumpReadable(), GetFieldAsString(), and SetFrom().

16.11.3.22 int OGRFeature::IsFieldSet (int *iField*) const [inline]

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. 361).

Parameters:

iField the field to test.

Returns:

TRUE if the field has been set, otherwise false.

Referenced by DumpReadable(), GetFieldAsBinary(), GetFieldAsDateTime(), GetFieldAsDouble(), GetFieldAsDoubleList(), GetFieldAsInteger(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldAsStringList(), SetField(), SetFrom(), and UnsetField().

16.11.3.23 OGRErr OGRFeature::SetFID (long *nFID*) [virtual]

Set the feature identifier.

For specific types of features this operation may fail on illegal features ids. Generally it always succeeds. Feature ids should be greater than or equal to zero, with the exception of OGRNullFID (-1) indicating that the feature id is unknown.

This method is the same as the C function **OGR_F_SetFID()** (p. 361).

Parameters:

nFID the new feature identifier value to assign.

Returns:

On success OGRERR_NONE, or on failure some other value.

Referenced by Clone(), and SetFrom().

16.11.3.24 void OGRFeature::SetField (int *iField*, int *nYear*, int *nMonth*, int *nDay*, int *nHour* = 0, int *nMinute* = 0, int *nSecond* = 0, int *nTZFlag* = 0)

Set field to date.

This method currently only has an effect for OFTDate, OFTTime and OFTDateTime fields.

This method is the same as the C function **OGR_F_SetFieldDateTime()** (p. 362).

Parameters:

iField the field to set, from 0 to **GetFieldCount()** (p. 107)-1.

nYear (including century)

nMonth (1-12)

nDay (1-31)

nHour (0-23)

nMinute (0-59)

nSecond (0-59)

nTZFlag (0=unknown, 1=localtime, 100=GMT, see data model for details)

References OGRField::Date, OGRField::Day, OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::Hour, OGRField::Minute, OGRField::Month, OFTDate, OFTDateTime, OFTTime, OGRField::Second, OGRField::TZFlag, and OGRField::Year.

16.11.3.25 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. 362).

Parameters:

iField the field to set, from 0 to **GetFieldCount()** (p. 107)-1.

nBytes bytes of data being set.

pabyData the raw data being applied.

References OGRField::Binary, OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::nCount, OFTBinary, OGRField::paData, and SetField().

16.11.3.26 void OGRFeature::SetField (int *iField*, OGRField * *puValue*)

Set field.

The passed value **OGRField** (p. 121) 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. 364).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

puValue the value to assign.

References OGRField::Binary, OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::IntegerList, IsFieldSet(), OGRField::nCount, OGRField::nMarker1, OGRField::nMarker2, OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, OFTTime, OGRField::paData, OGRField::paList, OGRField::RealList, OGRField::Set, OGRField::String, and OGRField::StringList.

16.11.3.27 void OGRFeature::SetField (int *iField*, char ** *papszValues*)

Set field to list of strings value.

This method currently on has an effect of OFTStringList fields.

This method is the same as the C function **OGR_F_SetFieldStringList()** (p. 365).

Parameters:

iField the field to set, from 0 to **GetFieldCount()** (p. 107)-1.

papszValues the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::nCount, OFTStringList, OGRField::paList, SetField(), and OGRField::StringList.

16.11.3.28 void OGRFeature::SetField (int *iField*, int *nCount*, double * *padfValues*)

Set field to list of doubles value.

This method currently on has an effect of OFTRealList fields.

This method is the same as the C function **OGR_F_SetFieldDoubleList()** (p. 363).

Parameters:

iField the field to set, from 0 to **GetFieldCount()** (p. 107)-1.

nCount the number of values in the list being assigned.

padfValues the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::nCount, OFTRealList, OGRField::paList, OGRField::RealList, and SetField().

16.11.3.29 void OGRFeature::SetField (int *iField*, int *nCount*, int * *panValues*)

Set field to list of integers value.

This method currently on has an effect of OFTIntegerList fields.

This method is the same as the C function **OGR_F_SetFieldIntegerList()** (p. 364).

Parameters:

iField the field to set, from 0 to **GetFieldCount()** (p. 107)-1.

nCount the number of values in the list being assigned.

panValues the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::IntegerList, OGRField::nCount, OFTIntegerList, OGRField::paList, and SetField().

16.11.3.30 void OGRFeature::SetField (int *iField*, const char * *pszValue*)

Set field to string value.

OFTInteger fields will be set based on an atoi() conversion of the string. OFTReal fields will be set based on an atof() conversion of the string. Other field types may be unaffected.

This method is the same as the C function **OGR_F_SetFieldString()** (p. 364).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

pszValue the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::Integer, IsFieldSet(), OGRField::nMarker2, OFTDate, OFTDateTime, OFTInteger, OFTReal, OFTString, OFTTime, OGRParseDate(), OGRField::Real, OGRField::Set, and OGRField::String.

16.11.3.31 void OGRFeature::SetField (int *iField*, double *dfValue*)

Set field to double value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This method is the same as the C function **OGR_F_SetFieldDouble()** (p. 362).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

dfValue the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::Integer, IsFieldSet(), OGRField::nMarker2, OFTInteger, OFTReal, OFTString, OGRField::Real, OGRField::Set, and OGRField::String.

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

Set field to integer value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This method is the same as the C function **OGR_F_SetFieldInteger()** (p. 363).

Parameters:

iField the field to fetch, from 0 to **GetFieldCount()** (p. 107)-1.

nValue the value to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OGRField::Integer, IsFieldSet(), OGRField::nMarker2, OFTInteger, OFTReal, OFTString, OGRField::Real, OGRField::Set, and OGRField::String.

Referenced by Clone(), SetField(), and SetFrom().

16.11.3.33 OGRErr OGRFeature::SetFrom (OGRFeature * *poSrcFeature*, int *bForgiving* = TRUE)

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The poSrcFeature does not need to have the same **OGRFeatureDefn** (p. 116). Field values are copied by corresponding field names. Field types do not have to exactly match. **SetField()** (p. 113) method conversion rules will be applied as needed.

This method is the same as the C function **OGR_F_SetFrom()** (p. 365).

Parameters:

poSrcFeature the feature from which geometry, and field values will be copied.

bForgiving TRUE if the operation should continue despite lacking output fields matching some of the source fields.

Returns:

OGRERR_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References GetFieldAsDouble(), GetFieldAsInteger(), GetFieldAsString(), GetFieldCount(), GetFieldDefnRef(), GetFieldIndex(), GetGeometryRef(), OGRFieldDefn::GetNameRef(), GetRawFieldRef(), GetStyleString(), OGRFieldDefn::GetType(), IsFieldSet(), OFTDate, OFTDateTime, OFTInteger, OFTReal, OFTString, OFTTime, SetFID(), SetField(), SetGeometry(), SetStyleString(), and UnsetField().

16.11.3.34 OGRErr OGRFeature::SetGeometry (OGRGeometry * *poGeomIn*)

Set feature geometry.

This method updates the features geometry, and operate exactly as **SetGeometryDirectly()** (p. 114), 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. 365).

Parameters:

poGeomIn new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.

Returns:

OGRERR_NONE if successful, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. 116) (checking not yet implemented).

References OGRGeometry::clone().

Referenced by Clone(), and SetFrom().

16.11.3.35 OGRErr OGRFeature::SetGeometryDirectly (OGRGeometry * *poGeomIn*)

Set feature geometry.

This method updates the features geometry, and operate exactly as **SetGeometry()** (p. 113), except that this method assumes ownership of the passed geometry.

This method is the same as the C function **OGR_F_SetGeometryDirectly()** (p. 366).

Parameters:

poGeomIn new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.

Returns:

OGRERR_NONE if successful, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. 116) (checking not yet implemented).

16.11.3.36 void OGRFeature::SetStyleString (const char * *pszString*) [virtual]

Set feature style string. This method operate exactly as **OGRFeature::SetStyleStringDirectly()** (p. 114) 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. 366).

Parameters:

pszString the style string to apply to this feature, cannot be NULL.

Referenced by Clone(), and SetFrom().

16.11.3.37 void OGRFeature::SetStyleStringDirectly (char * *pszString*) [virtual]

Set feature style string. This method operate exactly as **OGRFeature::SetStyleString()** (p. 114) except that it assumes ownership of the passed string.

This method is the same as the C function **OGR_F_SetStyleStringDirectly()** (p. 366).

Parameters:

pszString the style string to apply to this feature, cannot be NULL.

16.11.3.38 OGRGeometry * OGRFeature::StealGeometry ()

Take away ownership of geometry.

Fetch the geometry from this feature, and clear the reference to the geometry on the feature. This is a mechanism for the application to take over ownship of the geometry from the feature without copying. Sort of an inverse to **SetGeometryDirectly()** (p. 114).

After this call the **OGRFeature** (p. 101) will have a NULL geometry.

Returns:

the pointer to the geometry.

16.11.3.39 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. 367).

Parameters:

iField the field to unset.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OGRField::nMarker1, OGRField::nMarker2, OFTBinary, OFTIntegerList, OFTRealList, OFTString, OFTStringList, OGRField::paData, OGRField::paList, and OGRField::Set.

Referenced by SetFrom().

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

- **ogr_feature.h**
- ogrfeature.cpp

16.12 OGRFeatureDefn Class Reference

`#include <ogr_feature.h>`

Public Member Functions

- **OGRFeatureDefn** (const char *pszName=NULL)
- const char * **GetName** ()
- int **GetFieldCount** ()
- **OGRFieldDefn** * **GetFieldDefn** (int i)
- int **GetFieldIndex** (const char *)
- void **AddFieldDefn** (**OGRFieldDefn** *)
- **OGRwkbGeometryType GetGeomType** ()
- void **SetGeomType** (**OGRwkbGeometryType**)
- **OGRFeatureDefn** * **Clone** ()
- int **Reference** ()
- int **Dereference** ()
- int **GetReferenceCount** ()
- void **Release** ()

16.12.1 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. 116) is essentially a layer. In more object oriented approaches (such as SF CORBA) this can represent a class of features but doesn't necessarily relate to all of a layer, or just one layer.

This object also can contain some other information such as a name, the base geometry type and potentially other metadata.

It is reasonable for different translators to derive classes from **OGRFeatureDefn** (p. 116) with additional translator specific information.

16.12.2 Constructor & Destructor Documentation

16.12.2.1 OGRFeatureDefn::OGRFeatureDefn (const char * *pszName* = NULL)

Constructor

The **OGRFeatureDefn** (p. 116) maintains a reference count, but this starts at zero. It is mainly intended to represent a count of OGRFeature's based on this definition.

This method is the same as the C function **OGR_FD_Create()** (p. 367).

Parameters:

pszName the name to be assigned to this layer/class. It does not need to be unique.

References wkbUnknown.

Referenced by Clone().

16.12.3 Member Function Documentation

16.12.3.1 void OGRFeatureDefn::AddFieldDefn (OGRFieldDefn * *poNewDefn*)

Add a new field definition.

This method should only be called while there are no **OGRFeature** (p. 101) objects in existance based on this **OGRFeatureDefn** (p. 116). The **OGRFieldDefn** (p. 122) passed in is copied, and remains the responsibility of the caller.

This method is the same as the C function **OGR_FD_AddFieldDefn()** (p. 367).

Parameters:

poNewDefn the definition of the new field.

Referenced by Clone().

16.12.3.2 OGRFeatureDefn * OGRFeatureDefn::Clone ()

Create a copy of this feature definition.

Creates a deep copy of the feature definition.

Returns:

the copy.

References AddFieldDefn(), GetFieldCount(), GetFieldDefn(), GetGeomType(), GetName(), OGRFeatureDefn(), and SetGeomType().

16.12.3.3 int OGRFeatureDefn::Dereference () [inline]

Decrements the reference count by one.

This method is the same as the C function **OGR_FD_Dereference()** (p. 368).

Returns:

the updated reference count.

Referenced by Release().

16.12.3.4 int OGRFeatureDefn::GetFieldCount () [inline]

Fetch number of fields on this feature.

This method is the same as the C function **OGR_FD_GetFieldCount()** (p. 368).

Returns:

count of fields.

Referenced by Clone(), OGRFeature::Clone(), OGRDataSource::ExecuteSQL(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsString(), and OGRFeature::OGRFeature().

16.12.3.5 OGRFieldDefn * OGRFeatureDefn::GetFieldDefn (int *iField*)

Fetch field definition.

This method is the same as the C function **OGR_FD_GetFieldDefn()** (p. 369).

Parameters:

iField the field to fetch, between 0 and **GetFieldCount()** (p. 117)-1.

Returns:

a pointer to an internal field definition object. This object should not be modified or freed by the application.

Referenced by Clone(), OGRFeature::DumpReadable(), OGRDataSource::ExecuteSQL(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRFeature::SetField(), and OGRFeature::UnsetField().

16.12.3.6 int OGRFeatureDefn::GetFieldIndex (const char * *pszFieldName*)

Find field by name.

The field index of the first field matching the passed field name (case insensitively) is returned.

This method is the same as the C function **OGR_FD_GetFieldIndex()** (p. 369).

Parameters:

pszFieldName the field name to search for.

Returns:

the field index, or -1 if no match found.

16.12.3.7 OGRwkbGeometryType OGRFeatureDefn::GetGeomType () [inline]

Fetch the geometry base type.

Note that some drivers are unable to determine a specific geometry type for a layer, in which case wkbUnknown is returned. A value of wkbNone indicates no geometry is available for the layer at all. Many drivers do not properly mark the geometry type as 25D even if some or all geometries are in fact 25D. A few (broken) drivers return wkbPolygon for layers that also include wkbMultiPolygon.

This method is the same as the C function **OGR_FD_GetGeomType()** (p. 369).

Returns:

the base type for all geometry related to this definition.

Referenced by Clone().

16.12.3.8 const char * OGRFeatureDefn::GetName () [inline]

Get name of this **OGRFeatureDefn** (p. 116).

This method is the same as the C function **OGR_FD_GetName()** (p. 370).

Returns:

the name. This name is internal and should not be modified, or freed.

Referenced by Clone(), OGRFeature::DumpReadable(), and OGRDataSource::GetLayerByName().

16.12.3.9 int OGRFeatureDefn::GetReferenceCount () [inline]

Fetch current reference count.

This method is the same as the C function **OGR_FD_GetReferenceCount()** (p. 370).

Returns:

the current reference count.

16.12.3.10 int OGRFeatureDefn::Reference () [inline]

Increments the reference count by one.

The reference count is used keep track of the number of **OGRFeature** (p. 101) objects referencing this definition.

This method is the same as the C function **OGR_FD_Reference()** (p. 370).

Returns:

the updated reference count.

Referenced by OGRFeature::OGRFeature().

16.12.3.11 void OGRFeatureDefn::Release ()

Drop a reference to this object, and destroy if no longer referenced.

References Dereference().

16.12.3.12 void OGRFeatureDefn::SetGeomType (OGRwkbGeometryType *eNewType*)

Assign the base geometry type for this layer.

All geometry objects using this type must be of the defined type or a derived type. The default upon creation is wkbUnknown which allows for any geometry type. The geometry type should generally not be changed after any OGRFeatures have been created against this definition.

This method is the same as the C function **OGR_FD_SetGeomType()** (p. 371).

Parameters:

eNewType the new type to assign.

Referenced by Clone().

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

- **ogr_feature.h**
- ogrfeaturedefn.cpp

16.13 OGRField Union Reference

`#include <ogr_core.h>`

16.13.1 Detailed Description

OGRFeature (p. 101) field attribute value union.

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

- **ogr_core.h**

16.14 OGRFieldDefn Class Reference

`#include <ogr_feature.h>`

Public Member Functions

- **OGRFieldDefn** (const char *, **OGRFieldType**)
- **OGRFieldDefn** (**OGRFieldDefn** *)
- void **SetName** (const char *)
- const char * **GetNameRef** ()
- **OGRFieldType GetType** ()
- void **SetType** (**OGRFieldType** eTypeIn)
- **OGRJustification GetJustify** ()
- void **SetJustify** (**OGRJustification** eJustifyIn)
- int **GetWidth** ()
- void **SetWidth** (int nWidthIn)
- int **GetPrecision** ()
- void **SetPrecision** (int nPrecisionIn)
- void **Set** (const char *, **OGRFieldType**, int=0, int=0, **OGRJustification**=OJUndefined)
- void **SetDefault** (const **OGRField** *)

Static Public Member Functions

- static const char * **GetFieldTypeName** (**OGRFieldType**)

16.14.1 Detailed Description

Definition of an attribute of an **OGRFeatureDefn** (p. 116).

16.14.2 Constructor & Destructor Documentation

16.14.2.1 OGRFieldDefn::OGRFieldDefn (const char * *pszNameIn*, OGRFieldType *eTypeIn*)

Constructor.

Parameters:

pszNameIn the name of the new field.

eTypeIn the type of the new field.

16.14.2.2 OGRFieldDefn::OGRFieldDefn (OGRFieldDefn * *poPrototype*)

Constructor.

Create by cloning an existing field definition.

Parameters:

poPrototype the field definition to clone.

References GetJustify(), GetNameRef(), GetPrecision(), GetType(), GetWidth(), SetJustify(), SetPrecision(), and SetWidth().

16.14.3 Member Function Documentation

16.14.3.1 const char * OGRFieldDefn::GetFieldTypeName (OGRFieldType *eType*) [static]

Fetch human readable name for a field type.

This static method is the same as the C function **OGR_GetFieldTypeName()** (p. 390).

Parameters:

eType the field type to get name for.

Returns:

pointer to an internal static name string. It should not be modified or freed.

References OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

Referenced by OGRFeature::DumpReadable(), and OGR_GetFieldTypeName().

16.14.3.2 OGRJustification OGRFieldDefn::GetJustify () [inline]

Get the justification for this field.

This method is the same as the C function **OGR_Fld_GetJustify()** (p. 372).

Returns:

the justification.

Referenced by OGRFieldDefn().

16.14.3.3 const char * OGRFieldDefn::GetNameRef () [inline]

Fetch name of this field.

This method is the same as the C function **OGR_Fld_GetNameRef()** (p. 372).

Returns:

pointer to an internal name string that should not be freed or modified.

Referenced by OGRFeature::DumpReadable(), OGRDataSource::ExecuteSQL(), OGRFieldDefn(), and OGRFeature::SetFrom().

16.14.3.4 int OGRFieldDefn::GetPrecision () [inline]

Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.

This method is the same as the C function **OGR_Fld_GetPrecision()** (p. 372).

Returns:

the precision.

Referenced by OGRFeature::GetFieldAsString(), and OGRFieldDefn().

16.14.3.5 OGRFieldType OGRFieldDefn::GetType () [inline]

Fetch type of this field.

This method is the same as the C function **OGR_Fld_GetType()** (p. 373).

Returns:

field type.

Referenced by OGRFeature::DumpReadable(), OGRDataSource::ExecuteSQL(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRFieldDefn(), OGRFeature::SetField(), OGRFeature::SetFrom(), and OGRFeature::UnsetField().

16.14.3.6 int OGRFieldDefn::GetWidth () [inline]

Get the formatting width for this field.

This method is the same as the C function **OGR_Fld_GetWidth()** (p. 373).

Returns:

the width, zero means no specified width.

Referenced by OGRFeature::GetFieldAsString(), and OGRFieldDefn().

16.14.3.7 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. 373).

Parameters:

pszNameIn the new name to assign.

eTypeIn the new type (one of the OFT values like OFTInteger).

nWidthIn the preferred formatting width. Defaults to zero indicating undefined.

nPrecisionIn number of decimals places for formatting, defaults to zero indicating undefined.

eJustifyIn the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References SetJustify(), SetName(), SetPrecision(), SetType(), and SetWidth().

16.14.3.8 void OGRFieldDefn::SetDefault (const OGRField * *puDefaultIn*)

Set default field value.

Currently use of **OGRFieldDefn** (p. 122) "defaults" is discouraged. This feature may be fleshed out in the future.

References OFTInteger, OFTReal, and OFTString.

16.14.3.9 void OGRFieldDefn::SetJustify (OGRJustification *eJustify*) [inline]

Set the justification for this field.

This method is the same as the C function **OGR_Fld_SetJustify()** (p. 374).

Parameters:

eJustify the new justification.

Referenced by OGRFieldDefn(), and Set().

16.14.3.10 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. 374).

Parameters:

pszNameIn the new name to apply.

Referenced by Set().

16.14.3.11 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. 374).

Parameters:

nPrecision the new precision.

Referenced by OGRFieldDefn(), and Set().

16.14.3.12 void OGRFieldDefn::SetType (OGRFieldType *eType*) [inline]

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. 122) that is already part of an **OGRFeatureDefn** (p. 116).

This method is the same as the C function **OGR_Fld_SetType()** (p. 375).

Parameters:

eType the new field type.

Referenced by Set().

16.14.3.13 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. 375).

Parameters:

nWidth the new width.

Referenced by OGRFieldDefn(), and Set().

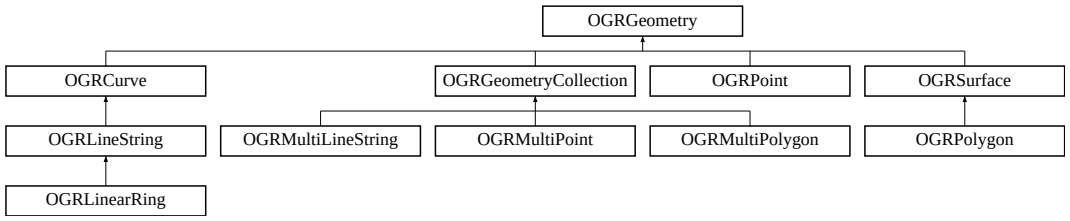
The documentation for this class was generated from the following files:

- **ogr_feature.h**
- ogrfielddefn.cpp

16.15 OGRGeometry Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRGeometry::



Public Member Functions

- virtual int **getDimension** () const =0
- virtual int **getCoordinateDimension** () const
- virtual OGRBoolean **IsEmpty** () const =0
- virtual OGRBoolean **IsValid** () const
- virtual OGRBoolean **IsSimple** () const
- virtual OGRBoolean **IsRing** () const
- virtual void **empty** ()=0
- virtual **OGRGeometry** * **clone** () const =0
- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const =0
- virtual int **WkbSize** () const =0
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)=0
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const =0
- virtual OGRErr **importFromWkt** (char **ppszInput)=0
- virtual OGRErr **exportToWkt** (char **ppszDstText) const =0
- virtual **OGRwkbGeometryType** **getGeometryType** () const =0
- virtual const char * **getGeometryName** () const =0
- virtual void **dumpReadable** (FILE *, const char *=NULL, char **papszOptions=NULL) const
- virtual void **flattenTo2D** ()=0
- virtual char * **exportToGML** () const
- virtual char * **exportToKML** () const
- virtual char * **exportToJson** () const
- virtual void **closeRings** ()
- virtual void **setCoordinateDimension** (int nDimension)
- void **assignSpatialReference** (**OGRSpatialReference** *poSR)
- **OGRSpatialReference** * **getSpatialReference** (void) const
- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)=0
- OGRErr **transformTo** (**OGRSpatialReference** *poSR)
- virtual void **segmentize** (double dfMaxLength)
- virtual OGRBoolean **Intersects** (**OGRGeometry** *) const
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const =0
- virtual OGRBoolean **Disjoint** (const **OGRGeometry** *) const
- virtual OGRBoolean **Touches** (const **OGRGeometry** *) const
- virtual OGRBoolean **Crosses** (const **OGRGeometry** *) const
- virtual OGRBoolean **Within** (const **OGRGeometry** *) const

- virtual OGRBoolean **Contains** (const **OGRGeometry** ∗) const
- virtual OGRBoolean **Overlaps** (const **OGRGeometry** ∗) const
- virtual **OGRGeometry** ∗ **getBoundary** () const
- virtual double **Distance** (const **OGRGeometry** ∗) const
- virtual **OGRGeometry** ∗ **ConvexHull** () const
- virtual **OGRGeometry** ∗ **Buffer** (double dfDist, int nQuadSegs=30) const
- virtual **OGRGeometry** ∗ **Intersection** (const **OGRGeometry** ∗) const
- virtual **OGRGeometry** ∗ **Union** (const **OGRGeometry** ∗) const
- virtual **OGRGeometry** ∗ **Difference** (const **OGRGeometry** ∗) const
- virtual **OGRGeometry** ∗ **SymmetricDifference** (const **OGRGeometry** ∗) const

16.15.1 Detailed Description

Abstract base class for all geometry classes.

Note that the family of spatial analysis methods (Equal(), **Disjoint()** (p. 131), ..., **ConvexHull()** (p. 130), **Buffer()** (p. 128), ...) are not implemented at ths time. Some other required and optional geometry methods have also been omitted at this time.

16.15.2 Member Function Documentation

16.15.2.1 void OGRGeometry::assignSpatialReference (OGRSpatialReference ∗ *poSR*)

Assign spatial reference to this object. Any existing spatial reference is replaced, but under no circumstances does this result in the object being reprojected. It is just changing the interpretation of the existing geometry. Note that assigning a spatial reference increments the reference count on the **OGRSpatialReference** (p. 224), but does not copy it.

This is similar to the SFCOM IGeometry::put_SpatialReference() method.

This method is the same as the C function **OGR_G_AssignSpatialReference()** (p. 377).

Parameters:

poSR new spatial reference system to apply.

References OGRSpatialReference::Reference(), and OGRSpatialReference::Release().

Referenced by OGRPolygon::clone(), OGRPoint::clone(), OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRLineString::clone(), OGRLinearRing::clone(), OGRGeometryCollection::clone(), OGRGeometryFactory::createFromFgf(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), OGRPolygon::transform(), OGRPoint::transform(), OGRLineString::transform(), and OGRGeometryCollection::transform().

16.15.2.2 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 segements should be used to define a 90 degree

curve - a quadrant of a circle. A value of 30 is a reasonable default. Large values result in large numbers of vertices in the resulting buffer geometry while small numbers reduce the accuracy of the result.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

dfDist the buffer distance to be applied.

nQuadSegs the number of segments used to approximate a 90 degree (quadrant) of curvature.

Returns:

the newly created geometry, or NULL if an error occurs.

16.15.2.3 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. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Implemented in **OGRPoint** (p. 199), **OGRLineString** (p. 177), **OGRLinearRing** (p. 172), **OGRPolygon** (p. 208), **OGRGeometryCollection** (p. 146), **OGRMultiPolygon** (p. 195), **OGRMultiPoint** (p. 192), and **OGRMultiLineString** (p. 189).

Referenced by OGRGeometryCollection::addGeometry(), and OGRFeature::SetGeometry().

16.15.2.4 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 **OGRLinearRing** (p. 172), **OGRPolygon** (p. 208), and **OGRGeometryCollection** (p. 146).

16.15.2.5 OGRBoolean OGRGeometry::Contains (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for containment.

Tests if actual geometry object contains the passed geometry.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the geometry to compare to this geometry.

Returns:

TRUE if poOtherGeom contains this geometry, otherwise FALSE.

References exportToGEOS().

16.15.2.6 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().

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.

16.15.2.7 OGRBoolean OGRGeometry::Crosses (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for crossing.

Tests if this geometry and the other passed into the method are crossing.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the geometry to compare to this geometry.

Returns:

TRUE if they are crossing, otherwise FALSE.

References exportToGEOS().

16.15.2.8 OGRGeometry * OGRGeometry::Difference (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute difference.

Generates a new geometry which is the region of this geometry with the region of the second geometry removed.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the other geometry removed from "this" geometry.

Returns:

a new geometry representing the difference or NULL if the difference is empty or an error occurs.

References exportToGEOS().

16.15.2.9 OGRBoolean OGRGeometry::Disjoint (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for disjointness.

Tests if this geometry and the other passed into the method are disjoint.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the geometry to compare to this geometry.

Returns:

TRUE if they are disjoint, otherwise FALSE.

References exportToGEOS().

16.15.2.10 double OGRGeometry::Distance (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute distance between two geometries.

Returns the shortest distance between the two geometries.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the other geometry to compare against.

Returns:

the distance between the geometries or -1 if an error occurs.

References exportToGEOS().

16.15.2.11 void OGRGeometry::dumpReadable (FILE * *fp*, const char * *pszPrefix* = NULL, char ** *papszOptions* = NULL) const [virtual]

Dump geometry in well known text format to indicated output file.

This method is the same as the C function **OGR_G_DumpReadable()** (p. 379).

Parameters:

fp the text file to write the geometry to.

pszPrefix the prefix to put on each line of output.

References dumpReadable(), exportToWkt(), OGRPolygon::getExteriorRing(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), OGRLineString::getNumPoints(), wkbGeometryCollection, wkbGeometryCollection25D, wkbLinearRing, wkbLineString, wkbLineString25D, wkbMultiLineString, wkbMultiLineString25D, wkbMultiPoint, wkbMultiPoint25D, wkbMultiPolygon, wkbMultiPolygon25D, wkbNone, wkbPoint, wkbPoint25D, wkbPolygon, wkbPolygon25D, and wkbUnknown.

Referenced by dumpReadable(), and OGRFeature::DumpReadable().

16.15.2.12 void OGRGeometry::empty () [pure virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 379).

Implemented in **OGRPoint** (p. 199), **OGRLineString** (p. 177), **OGRPolygon** (p. 208), and **OGRGeometryCollection** (p. 146).

16.15.2.13 int OGRGeometry::Equals (OGRGeometry * *poOtherGeom*) const [pure virtual]

Returns two if two geometries are equivalent.

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

Returns:

TRUE if equivalent or FALSE otherwise.

Implemented in **OGRPoint** (p. 199), **OGRLineString** (p. 178), **OGRPolygon** (p. 208), and **OGRGeometryCollection** (p. 146).

Referenced by OGRFeature::Equal(), and OGRGeometryCollection::Equals().

16.15.2.14 char * OGRGeometry::exportToGML () const [virtual]

Convert a geometry into GML format.

The GML geometry is expressed directly in terms of GML basic data types assuming the this is available in the gml namespace. The returned string should be freed with CPLFree() when no longer required.

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

Returns:

A GML fragment or NULL in case of error.

16.15.2.15 char * OGRGeometry::exportToJson () const [virtual]

Convert a geometry into GeoJSON format.

The returned string should be freed with CPLFree() when no longer required.

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

Returns:

A GeoJSON fragment or NULL in case of error.

16.15.2.16 char * OGRGeometry::exportToKML () const [virtual]

Convert a geometry into KML format.

The returned string should be freed with CPLFree() when no longer required.

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

Returns:

A KML fragment or NULL in case of error.

16.15.2.17 OGRErr OGRGeometry::exportToWkb (OGRwkbByteOrder *eByteOrder*, unsigned char * *pabyData*) const [pure virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 380).

Parameters:

eByteOrder One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.

pabyData a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 143) byte in size.

Returns:

Currently OGRERR_NONE is always returned.

Implemented in **OGRPoint** (p. 199), **OGRLineString** (p. 178), **OGRLinearRing** (p. 172), **OGRPolygon** (p. 209), and **OGRGeometryCollection** (p. 147).

Referenced by OGRGeometryCollection::exportToWkb().

16.15.2.18 OGRErr OGRGeometry::exportToWkt (char ** *ppszDstText*) const [pure virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 380).

Parameters:

ppszDstText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

Implemented in **OGRPoint** (p. 200), **OGRLineString** (p. 178), **OGRPolygon** (p. 209), **OGRGeometryCollection** (p. 147), **OGRMultiPolygon** (p. 195), **OGRMultiPoint** (p. 192), and **OGRMultiLineString** (p. 189).

Referenced by dumpReadable(), OGRMultiPolygon::exportToWkt(), OGRMultiLineString::exportToWkt(), OGRGeometryCollection::exportToWkt(), and OGRFeature::GetFieldAsString().

16.15.2.19 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. 381).

Implemented in **OGRPoint** (p. 200), **OGRLineString** (p. 179), **OGRPolygon** (p. 209), and **OGRGeometryCollection** (p. 148).

16.15.2.20 OGRGeometry * OGRGeometry::getBoundary () 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_GetBoundary().

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.

16.15.2.21 int OGRGeometry::getCoordinateDimension () const [virtual]

Get the dimension of the coordinates in this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method.

This method is the same as the C function **OGR_G_GetCoordinateDimension()** (p. 381).

Returns:

in practice this always returns 2 indicating that coordinates are specified within a two dimensional space.

Referenced by OGRGeometryCollection::addGeometryDirectly(), OGRPolygon::addRing(), OGRPolygon::addRingDirectly(), OGRLineString::clone(), OGRLinearRing::closeRings(), OGRPolygon::exportToWkb(), OGRLineString::exportToWkb(), OGRPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRLineString::exportToWkt(), OGRPolygon::getGeometryType(), OGRMultiPolygon::getGeometryType(), OGRMultiPoint::getGeometryType(), OGRMultiLineString::getGeometryType(), OGRLineString::getGeometryType(), OGRGeometryCollection::getGeometryType(), OGRLineString::getPoint(), OGRGeometryCollection::importFromWkb(), OGRLineString::segmentize(), OGRLineString::setNumPoints(), OGRLineString::setPoint(), OGRLineString::setPoints(), OGRLineString::Value(), OGRPolygon::WkbSize(), and OGRLineString::WkbSize().

16.15.2.22 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. 134)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 382).

Returns:

0 for points, 1 for lines and 2 for surfaces.

Implemented in **OGRPoint** (p. 200), **OGRLineString** (p. 179), **OGRPolygon** (p. 210), and **OGRGeometryCollection** (p. 148).

16.15.2.23 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. 382).

Parameters:

psEnvelope the structure in which to place the results.

Implemented in **OGRPoint** (p. 201), **OGRLineString** (p. 179), **OGRPolygon** (p. 210), and **OGRGeometryCollection** (p. 148).

Referenced by OGRGeometryCollection::getEnvelope(), OGRLayer::GetExtent(), Intersects(), and OGRGeometryFactory::organizePolygons().

16.15.2.24 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. 382).

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 **OGRPoint** (p. 201), **OGRLineString** (p. 180), **OGRLinearRing** (p. 173), **OGRPolygon** (p. 211), **OGRGeometryCollection** (p. 149), **OGRMultiPolygon** (p. 196), **OGRMultiPoint** (p. 192), and **OGRMultiLineString** (p. 189).

Referenced by dumpReadable(), and OGRFeature::GetFieldAsString().

16.15.2.25 OGRwkbGeometryType OGRGeometry::getGeometryType () const [pure virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 383).

Returns:

the geometry type code.

Implemented in **OGRPoint** (p. 201), **OGRLineString** (p. 180), **OGRPolygon** (p. 211), **OGRGeometryCollection** (p. 149), **OGRMultiPolygon** (p. 196), **OGRMultiPoint** (p. 193), and **OGRMultiLineString** (p. 190).

Referenced by OGRMultiPolygon::addGeometryDirectly(), OGRMultiPoint::addGeometryDirectly(), OGRMultiLineString::addGeometryDirectly(), dumpReadable(), OGRPolygon::Equals(), OGRPoint::Equals(), OGRLineString::Equals(), OGRGeometryCollection::Equals(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), and OGRGeometryCollection::get_Area().

16.15.2.26 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. 384).

Returns:

a reference to the spatial reference object. The object may be shared with many geometry objects, and should not be modified.

Referenced by OGRPolygon::clone(), OGRPoint::clone(), OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRLineString::clone(), OGRLinearRing::clone(), OGRGeometryCollection::clone(), and transformTo().

16.15.2.27 OGRErr OGRGeometry::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1) [pure virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 385).

Parameters:

pabyData the binary input data.

nSize the size of pabyData in bytes, or zero if not known.

Returns:

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

Implemented in **OGRPoint** (p. 202), **OGRLineString** (p. 182), **OGRLinearRing** (p. 173), **OGRPolygon** (p. 212), and **OGRGeometryCollection** (p. 150).

Referenced by OGRGeometryFactory::createFromWkb().

16.15.2.28 OGRErr OGRGeometry::importFromWkt (char ** *ppszInput*) [pure virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 386).

Parameters:

ppszInput pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

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

Implemented in **OGRPoint** (p. 203), **OGRLineString** (p. 183), **OGRPolygon** (p. 212), **OGRGeometryCollection** (p. 151), **OGRMultiPolygon** (p. 196), **OGRMultiPoint** (p. 193), and **OGRMultiLineString** (p. 190).

Referenced by OGRGeometryFactory::createFromWkt().

16.15.2.29 OGRGeometry * OGRGeometry::Intersection (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute intersection.

Generates a new geometry which is the region of intersection of the two geometries operated on. The Intersect() method can be used to test if two geometries intersect.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the other geometry intersected with "this" geometry.

Returns:

a new geometry representing the intersection or NULL if there is no intersection or an error occurs.

References exportToGEOS().

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

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigerous fashion otherwise TRUE is returned if the envelopes (bounding boxes) of the two features overlap.

The poOtherGeom argument may be safely NULL, but in this case the method will always return TRUE. That is, a NULL geometry is treated as being everywhere.

This method is the same as the C function **OGR_G_Intersects()** (p. 386).

Parameters:

poOtherGeom the other geometry to test against.

Returns:

TRUE if the geometries intersect, otherwise FALSE.

References exportToGEOS(), getEnvelope(), OGREnvelope::MaxX, OGREnvelope::MaxY, OGREnvelope::MinX, and OGREnvelope::MinY.

16.15.2.31 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 **OGRPoint** (p. 203), **OGRLineString** (p. 183), **OGRPolygon** (p. 213), and **OGRGeometryCollection** (p. 151).

16.15.2.32 OGRBoolean OGRGeometry::IsRing () const [virtual]

Test if the geometry is a ring

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

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.

16.15.2.33 OGRBoolean OGRGeometry::IsSimple () const [virtual]

Test if the geometry is simple

This method is the same as the C function **OGR_G_IsSimple()** (p. 387).

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.

16.15.2.34 OGRBoolean OGRGeometry::IsValid () const [virtual]

Test if the geometry is valid

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

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.

16.15.2.35 OGRBoolean OGRGeometry::Overlaps (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for overlap.

Tests if this geometry and the other passed into the method overlap, that is their intersection has a non-zero area.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the geometry to compare to this geometry.

Returns:

TRUE if they are overlapping, otherwise FALSE.

References exportToGEOS().

16.15.2.36 void OGRGeometry::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance. Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR_G_Segmentize()** (p. 388)

Parameters:

hGeom handle on the geometry to segmentize

dfMaxLength the maximum distance between 2 points after segmentization

Reimplemented in **OGRLineString** (p. 183), **OGRPolygon** (p. 213), and **OGRGeometryCollection** (p. 152).

16.15.2.37 void OGRGeometry::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters:

nNewDimension New coordinate dimension value, either 2 or 3.

Reimplemented in **OGRPoint** (p. 203), **OGRLineString** (p. 184), **OGRPolygon** (p. 214), and **OGRGeometryCollection** (p. 152).

Referenced by OGRGeometryCollection::setCoordinateDimension().

16.15.2.38 OGRGeometry * OGRGeometry::SymmetricDifference (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute symmetric difference.

Generates a new geometry which is the symmetric difference of this geometry and the second geometry passed into the method.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the other geometry.

Returns:

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

References exportToGEOS().

16.15.2.39 OGRBoolean OGRGeometry::Touches (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for touching.

Tests if this geometry and the other passed into the method are touching.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the geometry to compare to this geometry.

Returns:

TRUE if they are touching, otherwise FALSE.

References exportToGEOS().

16.15.2.40 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. 88) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 224) of the **OGRCoordinateTransformation** (p. 88) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 389).

Parameters:

poCT the transformation to apply.

Returns:

OGRERR_NONE on success or an error code.

Implemented in **OGRPoint** (p. 204), **OGRLineString** (p. 186), **OGRPolygon** (p. 214), and **OGRGeometryCollection** (p. 152).

Referenced by OGRGeometryCollection::transform(), and transformTo().

16.15.2.41 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. 88) object it is significantly more expensive to use this method to transform many geometries than it is to create the **OGRCoordinateTransformation** (p. 88) in advance, and call **transform()** (p. 141) 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. 389).

Parameters:

poSR spatial reference system to transform to.

Returns:

OGRERR_NONE on success, or an error code.

References getSpatialReference(), OGRCreateCoordinateTransformation(), and transform().

16.15.2.42 OGRGeometry * OGRGeometry::Union (const OGRGeometry * *poOtherGeom*) const [virtual]

Compute union.

Generates a new geometry which is the region of union of the two geometries operated on.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the other geometry unioned with "this" geometry.

Returns:

a new geometry representing the union or NULL if an error occurs.

References exportToGEOS().

16.15.2.43 OGRBoolean OGRGeometry::Within (const OGRGeometry * *poOtherGeom*) const [virtual]

Test for containment.

Tests if actual geometry object is within the passed geometry.

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

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE_NotSupported error.

Parameters:

poOtherGeom the geometry to compare to this geometry.

Returns:

TRUE if poOtherGeom is within this geometry, otherwise FALSE.

References exportToGEOS().

16.15.2.44 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. 390).

Returns:

size of binary representation in bytes.

Implemented in **OGRPoint** (p. 205), **OGRLineString** (p. 187), **OGRLinearRing** (p. 174), **OGRPolygon** (p. 214), and **OGRGeometryCollection** (p. 153).

Referenced by OGRGeometryCollection::exportToWkb(), OGRGeometryCollection::importFromWkb(), and OGRGeometryCollection::WkbSize().

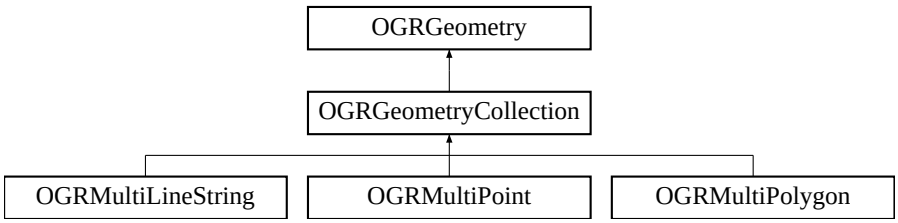
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrgeometry.cpp

16.16 OGRGeometryCollection Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRGeometryCollection::



Public Member Functions

- **OGRGeometryCollection** ()
- virtual const char * **getGeometryName** () const
- virtual **OGRwkbGeometryType getGeometryType** () const
- virtual **OGRGeometry** * **clone** () const
- virtual void **empty** ()
- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)
- virtual void **flattenTo2D** ()
- virtual OGRBoolean **IsEmpty** () const
- virtual void **segmentize** (double dfMaxLength)
- virtual int **WkbSize** () const
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const
- virtual OGRErr **importFromWkt** (char **)
- virtual OGRErr **exportToWkt** (char **ppszDstText) const
- virtual double **get_Area** () const
- virtual int **getDimension** () const
- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const
- int **getNumGeometries** () const
- **OGRGeometry** * **getGeometryRef** (int)
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const
- virtual void **setCoordinateDimension** (int nDimension)
- virtual OGRErr **addGeometry** (const **OGRGeometry** *)
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** *)
- virtual OGRErr **removeGeometry** (int iIndex, int bDelete=TRUE)
- void **closeRings** ()

16.16.1 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.

16.16.2 Constructor & Destructor Documentation

16.16.2.1 OGRGeometryCollection::OGRGeometryCollection ()

Create an empty geometry collection.

16.16.3 Member Function Documentation

16.16.3.1 OGRErr OGRGeometryCollection::addGeometry (const OGRGeometry * *poNewGeom*) [virtual]

Add a geometry to the container.

Some subclasses of **OGRGeometryCollection** (p. 144) 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. 375).

Parameters:

poNewGeom geometry to add to the container.

Returns:

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

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

Referenced by OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), and clone().

16.16.3.2 OGRErr OGRGeometryCollection::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 144) 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. 145) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 376).

There is no SFCOM analog to this method.

Parameters:

poNewGeom geometry to add to the container.

Returns:

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented in **OGRMultiPolygon** (p. 194), **OGRMultiPoint** (p. 191), and **OGRMultiLineString** (p. 188).

References OGRGeometry::getCoordinateDimension().

Referenced by addGeometry(), OGRGeometryFactory::createFromFgf(), and importFromWkt().

16.16.3.3 OGRGeometry * OGRGeometryCollection::clone () const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 129).

Reimplemented in **OGRMultiPolygon** (p. 195), **OGRMultiPoint** (p. 192), and **OGRMultiLineString** (p. 189).

References addGeometry(), OGRGeometry::assignSpatialReference(), and OGRGeometry::getSpatialReference().

16.16.3.4 void OGRGeometryCollection::closeRings () [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. 129).

References getGeometryType(), and wkbPolygon.

16.16.3.5 void OGRGeometryCollection::empty () [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 379).

Implements **OGRGeometry** (p. 132).

Referenced by importFromWkb(), OGRMultiPolygon::importFromWkt(), OGRMultiPoint::importFromWkt(), OGRMultiLineString::importFromWkt(), and importFromWkt().

16.16.3.6 OGRBoolean OGRGeometryCollection::Equals (OGRGeometry * *poOtherGeom*) const [virtual]

Returns two if two geometries are equivalent.

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

Returns:

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 132).

References OGRGeometry::Equals(), getGeometryRef(), getGeometryType(), OGRGeometry::getGeometryType(), and getNumGeometries().

16.16.3.7 OGRErr OGRGeometryCollection::exportToWkb (OGRwkbByteOrder *eByteOrder*, unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 380).

Parameters:

eByteOrder One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.

pabyData a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 143) byte in size.

Returns:

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 133).

References OGRGeometry::exportToWkb(), getGeometryType(), and OGRGeometry::WkbSize().

16.16.3.8 OGRErr OGRGeometryCollection::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 380).

Parameters:

ppszDstText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 134).

Reimplemented in **OGRMultiPolygon** (p. 195), **OGRMultiPoint** (p. 192), and **OGRMultiLineString** (p. 189).

References OGRGeometry::exportToWkt(), getGeometryName(), and getNumGeometries().

16.16.3.9 void OGRGeometryCollection::flattenTo2D () [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 381).

Implements **OGRGeometry** (p. 134).

16.16.3.10 double OGRGeometryCollection::get_Area () const [virtual]

Compute area of geometry collection.

The area is computed as the sum of the areas of all members in this collection.

Note:

No warning will be issued if a member of the collection does not support the get_Area method.

Returns:

computed area.

Reimplemented in **OGRMultiPolygon** (p. 195).

References getGeometryName(), OGRGeometry::getGeometryType(), wkbGeometryCollection, wkbLinearRing, wkbLineString, wkbMultiPolygon, and wkbPolygon.

16.16.3.11 int OGRGeometryCollection::getDimension () const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 134)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 382).

Returns:

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 135).

16.16.3.12 void OGRGeometryCollection::getEnvelope (OGREnvelope * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 382).

Parameters:

psEnvelope the structure in which to place the results.

Implements **OGRGeometry** (p. 135).

References OGRGeometry::getEnvelope(), OGREnvelope::MaxX, OGREnvelope::MaxY, OGREnvelope::MinX, and OGREnvelope::MinY.

16.16.3.13 const char * OGRGeometryCollection::getGeometryName () const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 382).

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. 135).

Reimplemented in **OGRMultiPolygon** (p. 196), **OGRMultiPoint** (p. 192), and **OGRMultiLineString** (p. 189).

Referenced by exportToWkt(), get_Area(), and importFromWkt().

16.16.3.14 OGRGeometry * OGRGeometryCollection::getGeometryRef (int *i*)

Fetch geometry from container.

This method returns a pointer to an geometry within the container. The returned geometry remains owned by the container, and should not be modified. The pointer is only valid untill the next change to the geometry container. Use IGeometry::clone() to make a copy.

This method relates to the SFCOM IGeometryCollection::get_Geometry() method.

Parameters:

i the index of the geometry to fetch, between 0 and **getNumGeometries()** (p. 150) - 1.

Returns:

pointer to requested geometry.

Referenced by OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRGeometry::dumpReadable(), Equals(), OGRMultiPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRMultiLineString::exportToWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRMultiPolygon::get_Area(), and OGRBuildPolygonFromEdges().

16.16.3.15 OGRwkbGeometryType OGRGeometryCollection::getGeometryType () const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 383).

Returns:

the geometry type code.

Implements **OGRGeometry** (p. 136).

Reimplemented in **OGRMultiPolygon** (p. 196), **OGRMultiPoint** (p. 193), and **OGRMultiLineString** (p. 190).

References OGRGeometry::getCoordinateDimension(), wkbGeometryCollection, and wkbGeometryCollection25D.

Referenced by closeRings(), Equals(), and exportToWkb().

16.16.3.16 int OGRGeometryCollection::getNumGeometries () const

Fetch number of geometries in container.

This method relates to the SFCOM IGeometryCollect::get_NumGeometries() method.

Returns:

count of children geometries. May be zero.

Referenced by OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRGeometry::dumpReadable(), Equals(), OGRMultiPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRMultiLineString::exportToWkt(), exportToWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRMultiPolygon::get_Area(), and OGRBuildPolygonFromEdges().

16.16.3.17 OGRErr OGRGeometryCollection::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 385).

Parameters:

pabyData the binary input data.

nSize the size of pabyData in bytes, or zero if not known.

Returns:

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

Implements **OGRGeometry** (p. 136).

References OGRGeometryFactory::createFromWkb(), empty(), OGRGeometry::getCoordinateDimension(), wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and OGRGeometry::WkbSize().

16.16.3.18 OGRErr OGRGeometryCollection::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 386).

Parameters:

ppszInput pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

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

Implements **OGRGeometry** (p. 137).

Reimplemented in **OGRMultiPolygon** (p. 196), **OGRMultiPoint** (p. 193), and **OGRMultiLineString** (p. 190).

References addGeometryDirectly(), OGRGeometryFactory::createFromWkt(), empty(), and getGeometryName().

16.16.3.19 OGRBoolean OGRGeometryCollection::IsEmpty () const [virtual]

Returns TRUE (non-zero) if the object has no points. Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns:

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. 138).

Referenced by OGRMultiPoint::exportToWkt().

16.16.3.20 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. 387).

Parameters:

iGeom the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.

bDelete if TRUE the geometry will be deallocated, otherwise it will not. The default is TRUE as the container is considered to own the geometries in it.

Returns:

OGRERR_NONE if successful, or OGRERR_FAILURE if the index is out of range.

Referenced by OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), and OGRGeometryFactory::forceToMultiPolygon().

16.16.3.21 void OGRGeometryCollection::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance. Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR_G_Segmentize()** (p. 388)

Parameters:

hGeom handle on the geometry to segmentize

dfMaxLength the maximum distance between 2 points after segmentization

Reimplemented from **OGRGeometry** (p. 140).

16.16.3.22 void OGRGeometryCollection::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters:

nNewDimension New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. 140).

References OGRGeometry::setCoordinateDimension().

16.16.3.23 OGRErr OGRGeometryCollection::transform (OGRCoordinateTransformation * *poCT*) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 88) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 224) of the **OGRCoordinateTransformation** (p. 88) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 389).

Parameters:

poCT the transformation to apply.

Returns:

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 141).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), and OGRGeometry::transform().

16.16.3.24 int OGRGeometryCollection::WkbSize () const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 390).

Returns:

size of binary representation in bytes.

Implements **OGRGeometry** (p. 143).

References OGRGeometry::WkbSize().

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

- **ogr_geometry.h**
- ogrgeometrycollection.cpp

16.17 OGRGeometryFactory Class Reference

`#include <ogr_geometry.h>`

Static Public Member Functions

- static OGRErr **createFromWkb** (unsigned char *, **OGRSpatialReference** *, **OGRGeometry** **, int=-1)
- static OGRErr **createFromWkt** (char **, **OGRSpatialReference** *, **OGRGeometry** **)
- static OGRErr **createFromFgf** (unsigned char *, **OGRSpatialReference** *, **OGRGeometry** **, int=-1, int *=NULL)
- static **OGRGeometry** * **createFromGML** (const char *)
- static void **destroyGeometry** (**OGRGeometry** *)
- static **OGRGeometry** * **createGeometry** (**OGRwkbGeometryType**)
- static **OGRGeometry** * **forceToPolygon** (**OGRGeometry** *)
- static **OGRGeometry** * **forceToMultiPolygon** (**OGRGeometry** *)
- static **OGRGeometry** * **forceToMultiPoint** (**OGRGeometry** *)
- static **OGRGeometry** * **forceToMultiLineString** (**OGRGeometry** *)
- static **OGRGeometry** * **organizePolygons** (**OGRGeometry** **papoPolygons, int nPolygonCount, int *pbResultValidGeometry, const char **papszOptions=NULL)
- static int **haveGEOS** ()

16.17.1 Detailed Description

Create geometry objects from well known text/binary.

16.17.2 Member Function Documentation

16.17.2.1 OGRErr OGRGeometryFactory::createFromFgf (unsigned char * *pabyData*, OGRSpatialReference * *poSR*, OGRGeometry ** *ppoReturn*, int *nBytes* = -1, int * *pnBytesConsumed* = NULL) [static]

Create a geometry object of the appropriate type from it's FGF (FDO Geometry Format) binary representation.

Also note that this is a static method, and that there is no need to instantiate an **OGRGeometryFactory** (p. 154) object.

The C function OGR_G_CreateFromFgf() is the same as this method.

Parameters:

pabyData pointer to the input BLOB data.

poSR pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.

ppoReturn the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure.

nBytes the number of bytes available in pabyData.

pnBytesConsumed if not NULL, it will be set to the number of bytes consumed (at most nBytes).

Returns:

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

References OGRGeometryCollection::addGeometryDirectly(), OGRPolygon::addRingDirectly(), OGRGeometry::assignSpatialReference(), OGRLineString::setNumPoints(), and OGRLineString::setPoint().

16.17.2.2 OGRGeometry * OGRGeometryFactory::createFromGML (const char * *pszData*) [static]

Create geometry from GML.

This method translates a fragment of GML containing only the geometry portion into a corresponding **OGRGeometry** (p. 127). There are many limitations on the forms of GML geometries supported by this parser, but they are too numerous to list here.

The C function OGR_G_CreateFromGML() is the same as this method.

Parameters:

pszData The GML fragment for the geometry.

Returns:

a geometry on succes, or NULL on error.

16.17.2.3 OGRErr OGRGeometryFactory::createFromWkb (unsigned char * *pabyData*, OGRSpatialReference * *poSR*, OGRGeometry ** *ppoReturn*, int *nBytes* = -1) [static]

Create a geometry object of the appropriate type from it's well known binary representation.

Note that if nBytes is passed as zero, no checking can be done on whether the pabyData is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **OGRGeometry::WkbSize()** (p. 143) 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. 154) object.

The C function **OGR_G_CreateFromWkb()** (p. 377) is the same as this method.

Parameters:

pabyData pointer to the input BLOB data.

poSR pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.

ppoReturn the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure.

nBytes the number of bytes available in pabyData, or -1 if it isn't known.

Returns:

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

References OGRGeometry::assignSpatialReference(), createGeometry(), and OGRGeometry::importFromWkb().

Referenced by OGRGeometryCollection::importFromWkb(), and OGR_G_CreateFromWkb().

16.17.2.4 OGRErr OGRGeometryFactory::createFromWkt (char ** *ppszData*, OGRSpatialReference * *poSR*, OGRGeometry ** *ppoReturn*) [static]

Create a geometry object of the appropriate type from it's well known text representation.

The C function **OGR_G_CreateFromWkt()** (p. 378) is the same as this method.

Parameters:

ppszData input zero terminated string containing well known text representation of the geometry to be created. The pointer is updated to point just beyond that last character consumed.

poSR pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.

ppoReturn the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL if the method fails.

Example:

```
const char* wkt= "POINT(0 0)";

// cast because OGR_G_CreateFromWkt will move the pointer
char* pszWkt = (char*) wkt.c_str();
OGRSpatialReferenceH ref = OSRNewSpatialReference(NULL);
OGRGeometryH new_geom;
OGRErr err = OGR_G_CreateFromWkt(&pszWkt, ref, &new_geom);
```

Returns:

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

References OGRGeometry::assignSpatialReference(), and OGRGeometry::importFromWkt().

Referenced by OGRGeometryCollection::importFromWkt(), and OGR_G_CreateFromWkt().

16.17.2.5 OGRGeometry * OGRGeometryFactory::createGeometry (OGRwkbGeometryType *eGeometryType*) [static]

Create an empty geometry of desired type.

This is equivelent 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. 378).

Parameters:

eGeometryType the type code of the geometry class to be instantiated.

Returns:

the newly create geometry or NULL on failure.

References wkbGeometryCollection, wkbLinearRing, wkbLineString, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbPoint, and wkbPolygon.

Referenced by createFromWkb(), and OGR_G_CreateGeometry().

16.17.2.6 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. 379).

Parameters:

poGeom the geometry to deallocate.

Referenced by OGR_G_DestroyGeometry().

16.17.2.7 OGRGeometry * OGRGeometryFactory::forceToMultiLineString (OGRGeometry * *poGeom*) [static]

Convert to multilinestring.

Tries to force the provided geometry to be a multilinestring.

- linestrings are placed in a multilinestring.
- geometry collections will be converted to multilinestring if they only contain linestrings.
- polygons will be changed to a collection of linestrings (one per ring).

The passed in geometry is consumed and a new one returned (or potentially the same one).

Returns:

new geometry.

References OGRMultiLineString::addGeometryDirectly(), OGRLineString::addSubLineString(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), OGRGeometryCollection::removeGeometry(), wkbGeometryCollection, wkbLineString, and wkbPolygon.

16.17.2.8 OGRGeometry * OGRGeometryFactory::forceToMultiPoint (OGRGeometry * *poGeom*) [static]

Convert to multipoint.

Tries to force the provided geometry to be a multipoint. Currently this just effects a change on points. The passed in geometry is consumed and a new one returned (or potentially the same one).

Returns:

new geometry.

References OGRMultiPoint::addGeometryDirectly(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::removeGeometry(), wkbGeometryCollection, and wkbPoint.

16.17.2.9 OGRGeometry * OGRGeometryFactory::forceToMultiPolygon (OGRGeometry * *poGeom*) [static]

Convert to multipolygon.

Tries to force the provided geometry to be a multipolygon. Currently this just effects a change on polygons. The passed in geometry is consumed and a new one returned (or potentially the same one).

Returns:

new geometry.

References OGRMultiPolygon::addGeometryDirectly(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRGeometryCollection::removeGeometry(), wkbGeometryCollection, and wkbPolygon.

16.17.2.10 OGRGeometry * OGRGeometryFactory::forceToPolygon (OGRGeometry * *poGeom*) [static]

Convert to polygon.

Tries to force the provided geometry to be a polygon. Currently this just effects a change on multipolygons. The passed in geometry is consumed and a new one returned (or potentially the same one).

Returns:

new geometry.

References OGRPolygon::addRing(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), wkbGeometryCollection, wkbMultiPolygon, and wkbPolygon.

16.17.2.11 int OGRGeometryFactory::haveGEOS () [static]

Test if GEOS enabled.

This static method returns TRUE if GEOS support is built into OGR, otherwise it returns FALSE.

Returns:

TRUE if available, otherwise FALSE.

Referenced by organizePolygons().

16.17.2.12 OGRGeometry * OGRGeometryFactory::organizePolygons (OGRGeometry ** *papoPolygons*, int *nPolygonCount*, int * *pbIsValidGeometry*, const char ** *papszOptions* = NULL) [static]

Organize polygons based on geometries.

Analyse a set of rings (passed as simple polygons), and based on a geometric analysis convert them into a polygon with inner rings, or a MultiPolygon if dealing with more than one polygon.

All the input geometries must be OGRPolygons with only a valid exterior ring (at least 4 points) and no interior rings.

The passed in geometries become the responsibility of the method, but the papoPolygons "pointer array" remains owned by the caller.

For faster computation, a polygon is considered to be inside another one if a single point of its external ring is included into the other one. (unless 'OGR_DEBUG_ORGANIZE_POLYGONS' configuration option is set to TRUE. In that case, a slower algorithm that tests exact topological relationships is used if GEOS is available.)

In cases where a big number of polygons is passed to this function, the default processing may be really slow. You can skip the processing by adding METHOD=SKIP to the option list (the result of the function will be a multi-polygon with all polygons as toplevel polygons) or only make it analyze counterclockwise polygons by adding METHOD=ONLY_CCW to the option list if you can assume that the outline of holes is counterclockwise defined (this is the convention for shapefiles e.g.)

If the OGR_ORGANIZE_POLYGONS configuration option is defined, its value will override the value of the METHOD option of papszOptions (usefull to modify the behaviour of the shapefile driver)

Parameters:

papoPolygons array of geometry pointers - should all be OGRPolygons. Ownership of the geometries is passed, but not of the array itself.

nPolygonCount number of items in papoPolygons

pbIsValidGeometry value will be set TRUE if result is valid or FALSE otherwise.

papszOptions a list of strings for passing options

Returns:

a single resulting geometry (either **OGRPolygon** (p. 206) or **OGRMultiPolygon** (p. 194)).

References OGRGeometry::getEnvelope(), haveGEOS(), and wkbPolygon.

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

- **ogr_geometry.h**
- ogrgeometryfactory.cpp

16.18 OGRLayer Class Reference

`#include <ogrsf_frmts.h>`

Inherited by OGRGenSQLResultsLayer.

Public Member Functions

- virtual **OGRGeometry** * **GetSpatialFilter** ()
- virtual void **SetSpatialFilter** (**OGRGeometry** *)
- virtual void **SetSpatialFilterRect** (double dfMinX, double dfMinY, double dfMaxX, double dfMaxY)
- virtual OGRErr **SetAttributeFilter** (const char *)
- virtual void **ResetReading** ()=0
- virtual **OGRFeature** * **GetNextFeature** ()=0
- virtual OGRErr **SetNextByIndex** (long nIndex)
- virtual **OGRFeature** * **GetFeature** (long nFID)
- virtual OGRErr **SetFeature** (**OGRFeature** *poFeature)
- virtual OGRErr **CreateFeature** (**OGRFeature** *poFeature)
- virtual OGRErr **DeleteFeature** (long nFID)
- virtual **OGRFeatureDefn** * **GetLayerDefn** ()=0
- virtual **OGRSpatialReference** * **GetSpatialRef** ()
- virtual int **GetFeatureCount** (int bForce=TRUE)
- virtual OGRErr **GetExtent** (**OGREnvelope** *psExtent, int bForce=TRUE)
- virtual int **TestCapability** (const char *)=0
- virtual const char * **GetInfo** (const char *)
- virtual OGRErr **CreateField** (**OGRFieldDefn** *poField, int bApproxOK=TRUE)
- virtual OGRErr **SyncToDisk** ()
- OGRStyleTable * **GetStyleTable** ()
- void **SetStyleTableDirectly** (OGRStyleTable *poStyleTable)
- void **SetStyleTable** (OGRStyleTable *poStyleTable)
- virtual const char * **GetFIDColumn** ()
- virtual const char * **GetGeometryColumn** ()
- int **Reference** ()
- int **Dereference** ()
- int **GetRefCount** () const

16.18.1 Detailed Description

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

16.18.2 Member Function Documentation

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

Create and write a new feature within a layer.

The passed feature is written to the layer as a new feature, rather than overwriting an existing one. If the feature has a feature id other than OGRNullFID, then the native implementation may use that as the feature

id of the new feature, but not necessarily. Upon successful return the passed feature will have been updated with the new feature id.

This method is the same as the C function **OGR_L_CreateFeature()** (p. 391).

Parameters:

poFeature the feature to write to disk.

Returns:

OGRERR_NONE on success.

16.18.2.2 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. 116) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. 116) used by a layer directly.

This function is the same as the C function **OGR_L_CreateField()** (p. 391).

Parameters:

poField field definition to write to disk.

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

Returns:

OGRERR_NONE on success.

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

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The **TestCapability()** (p. 169) 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. 392).

Parameters:

nFID the feature id to be deleted from the layer

Returns:

OGRERR_NONE on success.

16.18.2.4 int OGRLayer::Dereference ()

Decrement layer reference count.

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

Returns:

the reference count after decrementing.

16.18.2.5 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. 162) without setting a spatial filter.

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

This method is the same as the C function **OGR_L_GetExtent()** (p. 392).

Parameters:

psExtent the structure in which the extent value will be returned.

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

Returns:

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

References OGRGeometry::getEnvelope(), OGRFeature::GetGeometryRef(), GetLayerDefn(), GetNextFeature(), OGREnvelope::MaxX, OGREnvelope::MaxY, OGREnvelope::MinX, OGREnvelope::MinY, ResetReading(), and wkbNone.

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

Fetch a feature by it's identifier.

This function will attempt to read the identified feature. The nFID value cannot be OGRNullFID. Success or failure of this operation is unaffected by the spatial or attribute filters.

If this method returns a non-NULL feature, it is guaranteed that it's feature id (**OGRFeature::GetFID()** (p. 104)) will be the same as nFID.

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. 162); however, the call should always work if the feature exists as a fallback implementation just scans all the features in the layer looking for the desired feature.

Sequential reads are generally considered interrupted by a **GetFeature()** (p. 162) call.

This method is the same as the C function **OGR_L_GetFeature()** (p. 393).

Parameters:

nFID the feature id of the feature to read.

Returns:

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

References OGRFeature::GetFID(), GetNextFeature(), and ResetReading().

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

Fetch the feature count in this layer.

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

The returned count takes the spatial filter into account.

This method is the same as the C function **OGR_L_GetFeatureCount()** (p. 393).

Parameters:

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

Returns:

feature count, -1 if count not known.

References GetNextFeature(), and ResetReading().

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

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

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

Returns:

fid column name.

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

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

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

Returns:

fid column name.

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

Fetch metadata from layer.

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

This method is deprecated and will be replaced with a more general metadata model in the future. At this time no drivers return information via the **GetInfo()** (p. 164) call.

Parameters:

pszTag the tag for which information is being requested.

Returns:

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

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

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p. 116) is owned by the **OGRLayer** (p. 160), 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. 394).

Returns:

feature definition.

Referenced by OGRDataSource::ExecuteSQL(), GetExtent(), OGRDataSource::GetLayerByName(), and SetAttributeFilter().

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

Fetch the next available feature from this layer. The returned feature becomes the responsiblity of the caller to delete.

Only features matching the current spatial filter (set with **SetSpatialFilter()** (p. 167)) will be returned.

This method implements sequential access to the features of a layer. The **ResetReading()** (p. 166) method can be used to start at the beginning again.

This method is the same as the C function **OGR_L_GetNextFeature()** (p. 394).

Returns:

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

Referenced by GetExtent(), GetFeature(), GetFeatureCount(), and SetNextByIndex().

16.18.2.13 int OGRLayer::GetRefCount () const

Fetch reference count.

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

Returns:

the current reference count for the layer object itself.

Referenced by OGRDataSource::GetSummaryRefCount().

16.18.2.14 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. 394).

Returns:

spatial filter geometry.

16.18.2.15 OGRSpatialReference * OGRLayer::GetSpatialRef () [inline, virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. 160) and should not be modified or freed by the application.

This method is the same as the C function **OGR_L_GetSpatialRef()** (p. 395).

Returns:

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

16.18.2.16 void OGRLayer::GetStyleTable () [inline]

Returns layer style table.

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

Returns:

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

16.18.2.17 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.

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

Reset feature reading to start on the first feature. This affects **GetNextFeature()** (p. 164).

This method is the same as the C function **OGR_L_ResetReading()** (p. 395).

Referenced by GetExtent(), GetFeature(), GetFeatureCount(), SetAttributeFilter(), SetNextByIndex(), and SetSpatialFilter().

16.18.2.19 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. 164) 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 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 **ResetReading()** (p. 166)).

This method is the same as the C function **OGR_L_SetAttributeFilter()** (p. 396).

Parameters:

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

Returns:

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

References GetLayerDefn(), and ResetReading().

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

Rewrite an existing feature.

This method will write a feature to the layer, based on the feature id within the **OGRFeature** (p. 101).

Use OGRLayer::TestCapability(OLCRandomWrite) to establish if this layer supports random access writing via **SetFeature()** (p. 166).

This method is the same as the C function **OGR_L_SetFeature()** (p. 396).

Parameters:

poFeature the feature to write.

Returns:

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

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

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

This method allows positioning of a layer such that the **GetNextFeature()** (p. 164) 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. 164) 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. 167) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. 166) and then calls **GetNextFeature()** (p. 164) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. 169) method with a value of OLCFastSetNextByIndex.

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

Parameters:

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

Returns:

OGRERR_NONE on success or an error code.

References GetNextFeature(), and ResetReading().

16.18.2.22 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. 164) 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. 135)) overlaps the envelope of the spatial filter will be returned. This can result in more shapes being returned that should strictly be the case.

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

For the time being the passed filter geometry should be in the same SRS as the layer (as returned by **OGRLayer::GetSpatialRef()** (p. 165)). In the future this may be generalized.

This method is the same as the C function **OGR_L_SetSpatialFilter()** (p. 397).

Parameters:

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

References ResetReading().

Referenced by SetSpatialFilterRect().

16.18.2.23 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. 164) 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. 165)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. 167). 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. 397).

Parameters:

dfMinX the minimum X coordinate for the rectangular region.

dfMinY the minimum Y coordinate for the rectangular region.

dfMaxX the maximum X coordinate for the rectangular region.

dfMaxY the maximum Y coordinate for the rectangular region.

References OGRLineString::addPoint(), OGRPolygon::addRing(), and SetSpatialFilter().

16.18.2.24 void OGRLayer::SetStyleTable (OGRStyleTable * *poStyleTable*) [inline]

Set layer style table.

This method operate exactly as **OGRLayer::SetStyleTableDirectly()** (p. 168) except that it does not assume ownership of the passed table.

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

Parameters:

poStyleTable pointer to style table to set

16.18.2.25 void OGRLayer::SetStyleTableDirectly (OGRStyleTable * *poStyleTable*) [inline]

Set layer style table.

This method operate exactly as **OGRLayer::SetStyleTable()** (p. 168) except that it assumes ownership of the passed table.

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

Parameters:

poStyleTable pointer to style table to set

16.18.2.26 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.

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

Returns:

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

Referenced by OGRDataSource::SyncToDisk().

16.18.2.27 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. 162) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. 166) and **GetNextFeature()** (p. 164) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. 160) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 160) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. 166) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 160) 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. 101) intersection methods should return FALSE. This can be used as a clue by the application whether it should build and maintain it's own spatial index for features in this layer.
- **OLCFastFeatureCount** / "FastFeatureCount": TRUE if this layer can return a feature count (via **OGRLayer::GetFeatureCount()** (p. 163)) efficiently ... ie. without counting the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastGetExtent** / "FastGetExtent": TRUE if this layer can return its data extent (via **OGRLayer::GetExtent()** (p. 162)) efficiently ... ie. without scanning all the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastSetNextByIndex** / "FastSetNextByIndex": TRUE if this layer can perform the **SetNextByIndex()** (p. 167) call efficiently, otherwise FALSE.

This method is the same as the C function **OGR_L_TestCapability()** (p. 398).

Parameters:

pszCap the name of the capability to test.

Returns:

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

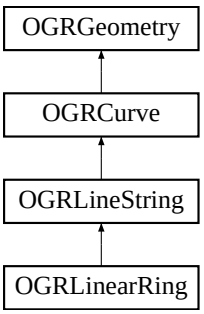
The documentation for this class was generated from the following files:

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

16.19 OGRLinearRing Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRLinearRing::



Public Member Functions

- virtual const char * **getGeometryName** () const
- virtual **OGRGeometry** * **clone** () const
- virtual int **isClockwise** () const
- virtual void **closeRings** ()
- virtual double **get_Area** () const
- virtual int **WkbSize** () const
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const

Friends

- class **OGRPolygon**

16.19.1 Detailed Description

Concrete representation of a closed ring.

This class is functionally equivelent to an **OGRLineString** (p. 175), 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. 206).

The **OGRLinearRing** (p. 171) has no corresponding free standing well known binary representation, so **importFromWkb()** (p. 173) and **exportToWkb()** (p. 172) will not actually work. There is a non-standard GDAL WKT representation though.

Because **OGRLinearRing** (p. 171) is not a "proper" free standing simple features object, it cannot be directly used on a feature via SetGeometry(), and cannot genearally be used with GEOS for operations like **Intersects()** (p. 138). Instead the polygon should be used, or the **OGRLinearRing** (p. 171) should be converted to an **OGRLineString** (p. 175) for such operations.

16.19.2 Member Function Documentation

16.19.2.1 OGRGeometry * OGRLinearRing::clone () const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRLineString** (p. 177).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), and OGRLineString::setPoints().

16.19.2.2 void OGRLinearRing::closeRings () [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. 129).

References OGRLineString::addPoint(), OGRGeometry::getCoordinateDimension(), OGRLineString::getX(), OGRLineString::getY(), and OGRLineString::getZ().

16.19.2.3 OGRErr OGRLinearRing::exportToWkb (OGRwkbByteOrder *eByteOrder*, unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 380).

Parameters:

eByteOrder One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.

pabyData a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 143) byte in size.

Returns:

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRLineString** (p. 178).

16.19.2.4 double OGRLinearRing::get_Area () const [virtual]

Compute area of ring.

The area is computed according to Green's Theorem:

Area is "Sum(x(i)∗y(i+1) - x(i+1)∗y(i))/2" for i = 0 to pointCount-1, assuming the last point is a duplicate of the first.

Returns:

computed area.

References OGRRawPoint::x, and OGRRawPoint::y.

Referenced by OGRPolygon::get_Area().

16.19.2.5 const char ∗ OGRLinearRing::getGeometryName () const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 382).

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. 180).

16.19.2.6 OGRErr OGRLinearRing::importFromWkb (unsigned char ∗ *pabyData*, int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 385).

Parameters:

pabyData the binary input data.

nSize the size of pabyData in bytes, or zero if not known.

Returns:

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_-UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Reimplemented from **OGRLineString** (p. 182).

16.19.2.7 int OGRLinearRing::isClockwise () const [virtual]

Returns TRUE if the ring has clockwise winding.

Returns:

TRUE if clockwise otherwise FALSE.

References OGRRawPoint::x, and OGRRawPoint::y.

16.19.2.8 int OGRLinearRing::WkbSize () const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 390).

Returns:

size of binary representation in bytes.

Reimplemented from **OGRLineString** (p. 187).

Referenced by OGR_G_AddGeometry(), and OGR_G_AddGeometryDirectly().

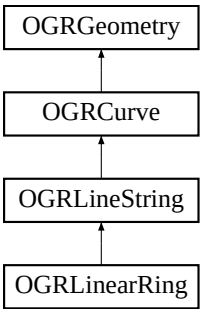
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrlinearring.cpp

16.20 OGRLineString Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRLineString::



Public Member Functions

- **OGRLineString** ()
- virtual int **WkbSize** () const
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const
- virtual OGRErr **importFromWkt** (char **)
- virtual OGRErr **exportToWkt** (char **ppszDstText) const
- virtual int **getDimension** () const
- virtual **OGRGeometry** * **clone** () const
- virtual void **empty** ()
- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const
- virtual OGRBoolean **IsEmpty** () const
- virtual double **get_Length** () const
- virtual void **StartPoint** (**OGRPoint** *) const
- virtual void **EndPoint** (**OGRPoint** *) const
- virtual void **Value** (double, **OGRPoint** *) const
- int **getNumPoints** () const
- void **getPoint** (int, **OGRPoint** *) const
- double **getX** (int i) const
- double **getY** (int i) const
- double **getZ** (int i) const
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const
- virtual void **setCoordinateDimension** (int nDimension)
- void **setNumPoints** (int)
- void **setPoint** (int, **OGRPoint** *)
- void **setPoint** (int, double, double, double)
- void **setPoints** (int, **OGRRawPoint** *, double *=NULL)
- void **setPoints** (int, double *padfX, double *padfY, double *padfZ=NULL)
- void **addPoint** (**OGRPoint** *)
- void **addPoint** (double, double, double)
- void **getPoints** (**OGRRawPoint** *, double *=NULL) const
- void **addSubLineString** (const **OGRLineString** *, int nStartVertex=0, int nEndVertex=-1)

- virtual **OGRwkbGeometryType getGeometryType** () const
- virtual const char * **getGeometryName** () const
- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)
- virtual void **flattenTo2D** ()
- virtual void **segmentize** (double dfMaxLength)

16.20.1 Detailed Description

Concrete representation of a multi-vertex line.

16.20.2 Constructor & Destructor Documentation

16.20.2.1 OGRLineString::OGRLineString ()

Create an empty line string.

Referenced by clone().

16.20.3 Member Function Documentation

16.20.3.1 void OGRLineString::addPoint (double *x*, double *y*, double *z*)

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

Parameters:

x the X coordinate to assign to the new point.

y the Y coordinate to assign to the new point.

z the Z coordinate to assign to the new point (defaults to zero).

References setPoint().

16.20.3.2 void OGRLineString::addPoint (OGRPoint * *poPoint*)

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

Parameters:

poPoint the point to assign to the new vertex.

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

Referenced by OGRLinearRing::closeRings(), OGRBuildPolygonFromEdges(), and OGRLayer::SetSpatialFilterRect().

16.20.3.3 void OGRLineString::addSubLineString (const OGRLineString * *poOtherLine*, int *nStartVertex* = 0, int *nEndVertex* = -1)

Add a segment of another linestring to this one.

Adds the request range of vertices to the end of this line string in an efficient manner. If the nStartVertex is larger than the nEndVertex then the vertices will be reversed as they are copied.

Parameters:

poOtherLine the other **OGRLineString** (p. 175).

nStartVertex the first vertex to copy, defaults to 0 to start with the first vertex in the other linestring.

nEndVertex the last vertex to copy, defaults to -1 indicating the last vertex of the other line string.

References getNumPoints(), padfZ, paoPoints, setNumPoints(), OGRRawPoint::x, and OGRRawPoint::y.

Referenced by OGRGeometryFactory::forceToMultiLineString().

16.20.3.4 OGRGeometry * OGRLineString::clone () const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 129).

Reimplemented in **OGRLinearRing** (p. 172).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getCoordinateDimension(), OGRGeometry::getSpatialReference(), OGRLineString(), setCoordinateDimension(), and setPoints().

16.20.3.5 void OGRLineString::empty () [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 379).

Implements **OGRGeometry** (p. 132).

References setNumPoints().

16.20.3.6 void OGRLineString::EndPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve end point.

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

Parameters:

poPoint the point to be assigned the end location.

Implements **OGRCurve** (p. 90).

References getPoint().

Referenced by Value().

16.20.3.7 OGRBoolean OGRLineString::Equals (OGRGeometry * *poOtherGeom*) const [virtual]

Returns two if two geometries are equivalent.

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

Returns:

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 132).

References getGeometryType(), OGRGeometry::getGeometryType(), getNumPoints(), getX(), getY(), and getZ().

Referenced by OGRPolygon::Equals().

16.20.3.8 OGRErr OGRLineString::exportToWkb (OGRwkbByteOrder *eByteOrder*, unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 380).

Parameters:

eByteOrder One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.

pabyData a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 143) byte in size.

Returns:

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 133).

Reimplemented in **OGRLinearRing** (p. 172).

References OGRGeometry::getCoordinateDimension(), and getGeometryType().

16.20.3.9 OGRErr OGRLineString::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 380).

Parameters:

ppszDstText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 134).

References OGRGeometry::getCoordinateDimension(), and getGeometryName().

Referenced by OGRPolygon::exportToWkt().

16.20.3.10 void OGRLineString::flattenTo2D () [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 381).

Implements **OGRGeometry** (p. 134).

16.20.3.11 double OGRLineString::get_Length () const [virtual]

Returns the length of the curve.

This method relates to the SFCOM ICurve::get_Length() method.

Returns:

the length of the curve, zero if the curve hasn't been initialized.

Implements **OGRCurve** (p. 91).

References OGRRawPoint::x, and OGRRawPoint::y.

16.20.3.12 int OGRLineString::getDimension () const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 134)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 382).

Returns:

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 135).

16.20.3.13 void OGRLineString::getEnvelope (OGREnvelope * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 382).

Parameters:

psEnvelope the structure in which to place the results.

Implements **OGRGeometry** (p. 135).

References OGREnvelope::MaxX, OGREnvelope::MaxY, OGREnvelope::MinX, OGREnvelope::MinY, OGRRawPoint::x, and OGRRawPoint::y.

Referenced by OGRPolygon::getEnvelope().

16.20.3.14 const char * OGRLineString::getGeometryName () const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 382).

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. 135).

Reimplemented in **OGRLinearRing** (p. 173).

Referenced by exportToWkt(), and importFromWkt().

16.20.3.15 OGRwkbGeometryType OGRLineString::getGeometryType () const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 383).

Returns:

the geometry type code.

Implements **OGRGeometry** (p. 136).

References OGRGeometry::getCoordinateDimension(), wkbLineString, and wkbLineString25D.

Referenced by Equals(), exportToWkb(), OGR_G_AddGeometry(), and OGR_G_AddGeometryDirectly().

16.20.3.16 int OGRLineString::getNumPoints () const [inline]

Fetch vertex count.

Returns the number of vertices in the line string.

Returns:

vertex count.

Referenced by addSubLineString(), OGRGeometry::dumpReadable(), Equals(), OGR_G_-GetPointCount(), and OGRBuildPolygonFromEdges().

16.20.3.17 void OGRLineString::getPoint (int *i*, OGRPoint * *poPoint*) const

Fetch a point in line string.

This method relates to the SFCOM ILineString::get_Point() method.

Parameters:

i the vertex to fetch, from 0 to **getNumPoints()** (p. 180)-1.

poPoint a point to initialize with the fetched point.

References OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGRPoint::setY(), and OGR-Point::setZ().

Referenced by EndPoint(), and StartPoint().

16.20.3.18 void OGRLineString::getPoints (OGRRawPoint * *paoPointsOut*, double * *padfZ* = NULL) const

Returns all points of line string.

This method copies all points into user list. This list must be at least sizeof(OGRRawPoint) * OGRGeom-etry::getNumPoints() byte in size. It also copies all Z coordinates.

There is no SFCOM analog to this method.

Parameters:

paoPointsOut a buffer into which the points is written.

padfZ the Z values that go with the points (optional, may be NULL).

16.20.3.19 double OGRLineString::getX (int *iVertex*) const [inline]

Get X at vertex.

Returns the X value at the indicated vertex. If iVertex is out of range a crash may occur, no internal range checking is performed.

Parameters:

iVertex the vertex to return, between 0 and **getNumPoints()** (p. 180)-1.

Returns:

X value.

Referenced by OGRLinearRing::closeRings(), Equals(), and OGRBuildPolygonFromEdges().

16.20.3.20 double OGRLineString::getY (int *iVertex*) const [inline]

Get Y at vertex.

Returns the Y value at the indicated vertex. If iVertex is out of range a crash may occur, no internal range checking is performed.

Parameters:

iVertex the vertex to return, between 0 and **getNumPoints()** (p. 180)-1.

Returns:

X value.

Referenced by OGRLinearRing::closeRings(), Equals(), and OGRBuildPolygonFromEdges().

16.20.3.21 double OGRLineString::getZ (int *iVertex*) const

Get Z at vertex.

Returns the Z (elevation) value at the indicated vertex. If no Z value is available, 0.0 is returned. If iVertex is out of range a crash may occur, no internal range checking is performed.

Parameters:

iVertex the vertex to return, between 0 and **getNumPoints()** (p. 180)-1.

Returns:

Z value.

Referenced by OGRLinearRing::closeRings(), Equals(), and OGRBuildPolygonFromEdges().

16.20.3.22 OGRErr OGRLineString::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 385).

Parameters:

pabyData the binary input data.

nSize the size of pabyData in bytes, or zero if not known.

Returns:

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

Implements **OGRGeometry** (p. 136).

Reimplemented in **OGRLinearRing** (p. 173).

References setNumPoints(), and wkbLineString.

16.20.3.23 OGRErr OGRLineString::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 386).

Parameters:

ppszInput pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

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

Implements **OGRGeometry** (p. 137).

References getGeometryName().

16.20.3.24 OGRBoolean OGRLineString::IsEmpty () const [virtual]

Returns TRUE (non-zero) if the object has no points. Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns:

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. 138).

16.20.3.25 void OGRLineString::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance. Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR_G_Segmentize()** (p. 388)

Parameters:

hGeom handle on the geometry to segmentize

dfMaxLength the maximum distance between 2 points after segmentization

Reimplemented from **OGRGeometry** (p. 140).

References OGRGeometry::getCoordinateDimension(), OGRRawPoint::x, and OGRRawPoint::y.

16.20.3.26 void OGRLineString::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters:

nNewDimension New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. 140).

Referenced by clone(), and OGRPolygon::exportToWkt().

16.20.3.27 void OGRLineString::setNumPoints (int *nNewPointCount*)

Set number of points in geometry.

This method primary exists to preset the number of points in a linestring geometry before **setPoint()** (p. 185) is used to assign them to avoid reallocating the array larger with each call to **addPoint()** (p. 176).

This method has no SFCOM analog.

Parameters:

nNewPointCount the new number of points for geometry.

References OGRGeometry::getCoordinateDimension().

Referenced by addSubLineString(), OGRGeometryFactory::createFromFgf(), empty(), importFromWkb(), setPoint(), and setPoints().

16.20.3.28 void OGRLineString::setPoint (int *iPoint*, double *xIn*, double *yIn*, double *zIn*)

Set the location of a vertex in line string.

If iPoint is larger than the number of necessary the number of existing points in the line string, the point count will be increased to accomodate the request.

There is no SFCOM analog to this method.

Parameters:

iPoint the index of the vertex to assign (zero based).

xIn input X coordinate to assign.

yIn input Y coordinate to assign.

zIn input Z coordinate to assign (defaults to zero).

References OGRGeometry::getCoordinateDimension(), setNumPoints(), OGRRawPoint::x, and OGRRawPoint::y.

16.20.3.29 void OGRLineString::setPoint (int *iPoint*, OGRPoint * *poPoint*)

Set the location of a vertex in line string.

If iPoint is larger than the number of necessary the number of existing points in the line string, the point count will be increased to accomodate the request.

There is no SFCOM analog to this method.

Parameters:

iPoint the index of the vertex to assign (zero based).

poPoint the value to assign to the vertex.

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

Referenced by addPoint(), and OGRGeometryFactory::createFromFgf().

16.20.3.30 void OGRLineString::setPoints (int *nPointsIn*, double * *padfX*, double * *padfY*, double * *padfZ* = NULL)

Assign all points in a line string.

This method clear any existing points assigned to this line string, and assigns a whole new set.

There is no SFCOM analog to this method.

Parameters:

nPointsIn number of points being passed in padfX and padfY.

padfX list of X coordinates of points being assigned.

padfY list of Y coordinates of points being assigned.

padfZ list of Z coordinates of points being assigned (defaults to NULL for 2D objects).

References setNumPoints(), OGRRawPoint::x, and OGRRawPoint::y.

16.20.3.31 void OGRLineString::setPoints (int *nPointsIn*, OGRRawPoint * *paoPointsIn*, double * *padfZ* = NULL)

Assign all points in a line string.

This method clears any existing points assigned to this line string, and assigns a whole new set. It is the most efficient way of assigning the value of a line string.

There is no SFCOM analog to this method.

Parameters:

nPointsIn number of points being passed in paoPointsIn

paoPointsIn list of points being assigned.

padfZ the Z values that go with the points (optional, may be NULL).

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

Referenced by clone(), OGRLinearRing::clone(), OGRPolygon::importFromWkt(), OGRMultiPolygon::importFromWkt(), OGRMultiLineString::importFromWkt(), and transform().

16.20.3.32 void OGRLineString::StartPoint (OGRPoint * *poPoint*) const [virtual]

Return the curve start point.

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

Parameters:

poPoint the point to be assigned the start location.

Implements **OGRCurve** (p. 91).

References getPoint().

Referenced by Value().

16.20.3.33 OGRErr OGRLineString::transform (OGRCoordinateTransformation * *poCT*) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 88) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 224) of the **OGRCoordinateTransformation** (p. 88) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 389).

Parameters:

poCT the transformation to apply.

Returns:

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 141).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), setPoints(), OGRCoordinateTransformation::Transform(), OGRRawPoint::x, and OGRRawPoint::y.

Referenced by OGRPolygon::transform().

16.20.3.34 void OGRLineString::Value (double *dfDistance*, OGRPoint * *poPoint*) const [virtual]

Fetch point at given distance along curve.

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

Parameters:

dfDistance distance along the curve at which to sample position. This distance should be between zero and **get_Length()** (p. 179) for this curve.

poPoint the point to be assigned the curve position.

Implements **OGRCurve** (p. 91).

References EndPoint(), OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGRPoint::setY(), OGRPoint::setZ(), StartPoint(), OGRRawPoint::x, and OGRRawPoint::y.

16.20.3.35 int OGRLineString::WkbSize () const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 390).

Returns:

size of binary representation in bytes.

Implements **OGRGeometry** (p. 143).

Reimplemented in **OGRLinearRing** (p. 174).

References OGRGeometry::getCoordinateDimension().

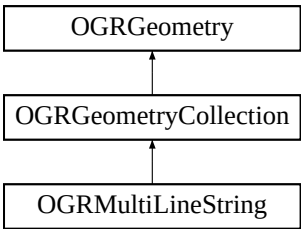
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrlinestring.cpp

16.21 OGRMultiLineString Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiLineString::



Public Member Functions

- virtual const char * **getGeometryName** () const
- virtual **OGRwkbGeometryType getGeometryType** () const
- virtual **OGRGeometry** * **clone** () const
- virtual OGRErr **importFromWkt** (char **)
- virtual OGRErr **exportToWkt** (char **) const
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** *)

16.21.1 Detailed Description

A collection of OGRLineStrings.

16.21.2 Member Function Documentation

16.21.2.1 OGRErr OGRMultiLineString::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 144) 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. 145) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 376).

There is no SFCOM analog to this method.

Parameters:

poNewGeom geometry to add to the container.

Returns:

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. 145).

References OGRGeometry::getGeometryType(), wkbLineString, and wkbLineString25D.

Referenced by OGRGeometryFactory::forceToMultiLineString(), and importFromWkt().

16.21.2.2 OGRGeometry * OGRMultiLineString::clone () const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 146).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

16.21.2.3 OGRErr OGRMultiLineString::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 380).

Parameters:

ppszDstText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. 147).

References OGRGeometry::exportToWkt(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

16.21.2.4 const char * OGRMultiLineString::getGeometryName () const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 382).

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. 149).

Referenced by importFromWkt().

16.21.2.5 OGRwkbGeometryType OGRMultiLineString::getGeometryType () const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 383).

Returns:

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 149).

References OGRGeometry::getCoordinateDimension(), wkbMultiLineString, and wkbMultiLineString25D.

16.21.2.6 OGRErr OGRMultiLineString::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 386).

Parameters:

ppszInput pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

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

Reimplemented from **OGRGeometryCollection** (p. 151).

References addGeometryDirectly(), OGRGeometryCollection::empty(), getGeometryName(), and OGRLineString::setPoints().

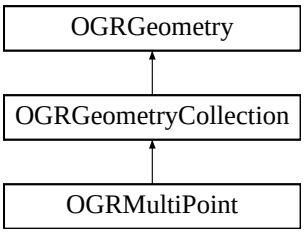
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultilinestring.cpp

16.22 OGRMultiPoint Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiPoint::



Public Member Functions

- virtual const char * **getGeometryName** () const
- virtual **OGRwkbGeometryType getGeometryType** () const
- virtual **OGRGeometry** * **clone** () const
- virtual OGRErr **importFromWkt** (char **)
- virtual OGRErr **exportToWkt** (char **) const
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** *)

16.22.1 Detailed Description

A collection of OGRPoints.

16.22.2 Member Function Documentation

16.22.2.1 OGRErr OGRMultiPoint::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 144) 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. 145) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 376).

There is no SFCOM analog to this method.

Parameters:

poNewGeom geometry to add to the container.

Returns:

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. 145).

References OGRGeometry::getGeometryType(), wkbPoint, and wkbPoint25D.

Referenced by OGRGeometryFactory::forceToMultiPoint(), and importFromWkt().

16.22.2.2 OGRGeometry * OGRMultiPoint::clone () const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 146).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

16.22.2.3 OGRErr OGRMultiPoint::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 380).

Parameters:

ppszDstText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. 147).

References OGRGeometry::getCoordinateDimension(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), OGRPoint::IsEmpty(), and OGRGeometryCollection::IsEmpty().

16.22.2.4 const char * OGRMultiPoint::getGeometryName () const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 382).

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. 149).

Referenced by exportToWkt(), and importFromWkt().

16.22.2.5 OGRwkbGeometryType OGRMultiPoint::getGeometryType () const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 383).

Returns:

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 149).

References OGRGeometry::getCoordinateDimension(), wkbMultiPoint, and wkbMultiPoint25D.

16.22.2.6 OGRErr OGRMultiPoint::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 386).

Parameters:

ppszInput pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

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

Reimplemented from **OGRGeometryCollection** (p. 151).

References addGeometryDirectly(), OGRGeometryCollection::empty(), and getGeometryName().

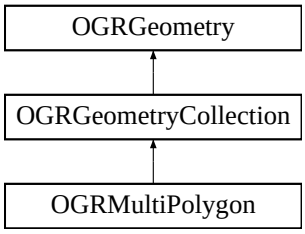
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultipoint.cpp

16.23 OGRMultiPolygon Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRMultiPolygon::



Public Member Functions

- virtual const char * **getGeometryName** () const
- virtual **OGRwkbGeometryType getGeometryType** () const
- virtual **OGRGeometry** * **clone** () const
- virtual OGRErr **importFromWkt** (char **)
- virtual OGRErr **exportToWkt** (char **) const
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** *)
- virtual double **get_Area** () const

16.23.1 Detailed Description

A collection of non-overlapping OGRPolygons.

Note that the IMultiSurface class hasn't been modelled, nor have any of it's methods.

16.23.2 Member Function Documentation

16.23.2.1 OGRErr OGRMultiPolygon::addGeometryDirectly (OGRGeometry * *poNewGeom*) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. 144) 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. 145) does.

This method is the same as the C function **OGR_G_AddGeometryDirectly()** (p. 376).

There is no SFCOM analog to this method.

Parameters:

poNewGeom geometry to add to the container.

Returns:

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. 145).

References OGRGeometry::getGeometryType(), wkbPolygon, and wkbPolygon25D.

Referenced by OGRGeometryFactory::forceToMultiPolygon(), and importFromWkt().

16.23.2.2 OGRGeometry * OGRMultiPolygon::clone () const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. 146).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

16.23.2.3 OGRErr OGRMultiPolygon::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 380).

Parameters:

ppszDstText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. 147).

References OGRGeometry::exportToWkt(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

16.23.2.4 double OGRMultiPolygon::get_Area () const [virtual]

Compute area of multipolygon.

The area is computed as the sum of the areas of all polygon members in this collection.

Returns:

computed area.

Reimplemented from **OGRGeometryCollection** (p. 148).

References OGRPolygon::get_Area(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

16.23.2.5 const char * OGRMultiPolygon::getGeometryName () const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 382).

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. 149).

Referenced by importFromWkt().

16.23.2.6 OGRwkbGeometryType OGRMultiPolygon::getGeometryType () const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 383).

Returns:

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. 149).

References OGRGeometry::getCoordinateDimension(), wkbMultiPolygon, and wkbMultiPolygon25D.

16.23.2.7 OGRErr OGRMultiPolygon::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 386).

Parameters:

ppszInput pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

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

Reimplemented from **OGRGeometryCollection** (p. 151).

References addGeometryDirectly(), OGRPolygon::addRingDirectly(), OGRGeometryCollection::empty(), getGeometryName(), and OGRLineString::setPoints().

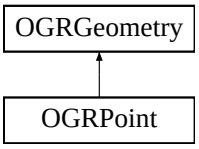
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrmultipolygon.cpp

16.24 OGRPoint Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRPoint::



Public Member Functions

- **OGRPoint** ()
- virtual int **WkbSize** () const
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const
- virtual OGRErr **importFromWkt** (char **)
- virtual OGRErr **exportToWkt** (char **ppszDstText) const
- virtual int **getDimension** () const
- virtual **OGRGeometry** * **clone** () const
- virtual void **empty** ()
- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const
- virtual OGRBoolean **IsEmpty** () const
- double **getX** () const
- double **getY** () const
- double **getZ** () const
- virtual void **setCoordinateDimension** (int nDimension)
- void **setX** (double xIn)
- void **setY** (double yIn)
- void **setZ** (double zIn)
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const
- virtual const char * **getGeometryName** () const
- virtual **OGRwkbGeometryType getGeometryType** () const
- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)
- virtual void **flattenTo2D** ()

16.24.1 Detailed Description

Point class.

Implements SFCOM IPoint methods.

16.24.2 Constructor & Destructor Documentation

16.24.2.1 OGRPoint::OGRPoint ()

Create a (0,0) point.

References empty().

Referenced by clone().

16.24.3 Member Function Documentation

16.24.3.1 OGRGeometry * OGRPoint::clone () const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 129).

References OGRGeometry::assignSpatialReference(), OGRGeometry::getSpatialReference(), OGRPoint(), and setCoordinateDimension().

16.24.3.2 void OGRPoint::empty () [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 379).

Implements **OGRGeometry** (p. 132).

Referenced by importFromWkt(), and OGRPoint().

16.24.3.3 OGRBoolean OGRPoint::Equals (OGRGeometry * *poOtherGeom*) const [virtual]

Returns two if two geometries are equivalent.

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

Returns:

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 132).

References getGeometryType(), OGRGeometry::getGeometryType(), getX(), getY(), and getZ().

16.24.3.4 OGRErr OGRPoint::exportToWkb (OGRwkbByteOrder *eByteOrder*, unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 380).

Parameters:

eByteOrder One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.

pabyData a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 143) byte in size.

Returns:

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 133).

References getGeometryType().

16.24.3.5 OGRErr OGRPoint::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 380).

Parameters:

ppszDstText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 134).

16.24.3.6 void OGRPoint::flattenTo2D () [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 381).

Implements **OGRGeometry** (p. 134).

16.24.3.7 int OGRPoint::getDimension () const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 134)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 382).

Returns:

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 135).

16.24.3.8 void OGRPoint::getEnvelope (OGREnvelope * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 382).

Parameters:

psEnvelope the structure in which to place the results.

Implements **OGRGeometry** (p. 135).

References getX(), getY(), OGREnvelope::MaxX, OGREnvelope::MaxY, OGREnvelope::MinX, and OGREnvelope::MinY.

16.24.3.9 const char * OGRPoint::getGeometryName () const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 382).

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. 135).

16.24.3.10 OGRwkbGeometryType OGRPoint::getGeometryType () const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 383).

Returns:

the geometry type code.

Implements **OGRGeometry** (p. 136).

References wkbPoint, and wkbPoint25D.

Referenced by OGRPolygon::Centroid(), Equals(), exportToWkb(), and OGRPolygon::PointOnSurface().

16.24.3.11 double OGRPoint::getX () const [inline]

Fetch X coordinate.

Relates to the SFCOM IPoint::get_X() method.

Returns:

the X coordinate of this point.

Referenced by OGRLineString::addPoint(), OGRPolygon::Centroid(), Equals(), OGRMultiPoint::exportToWkt(), OGRCurve::get_IsClosed(), getEnvelope(), OGRPolygon::PointOnSurface(), and OGRLineString::setPoint().

16.24.3.12 double OGRPoint::getY () const [inline]

Fetch Y coordinate.

Relates to the SFCOM IPoint::get_Y() method.

Returns:

the Y coordinate of this point.

Referenced by OGRLineString::addPoint(), OGRPolygon::Centroid(), Equals(), OGRMultiPoint::exportToWkt(), OGRCurve::get_IsClosed(), getEnvelope(), OGRPolygon::PointOnSurface(), and OGRLineString::setPoint().

16.24.3.13 double OGRPoint::getZ () const [inline]

Fetch Z coordinate.

Relates to the SFCOM IPoint::get_Z() method.

Returns:

the Z coordinate of this point, or zero if it is a 2D point.

Referenced by OGRLineString::addPoint(), Equals(), OGRMultiPoint::exportToWkt(), and OGRLineString::setPoint().

16.24.3.14 OGRErr OGRPoint::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 385).

Parameters:

pabyData the binary input data.

nSize the size of pabyData in bytes, or zero if not known.

Returns:

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

Implements **OGRGeometry** (p. 136).

References wkbPoint.

16.24.3.15 OGRErr OGRPoint::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 386).

Parameters:

ppszInput pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

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

Implements **OGRGeometry** (p. 137).

References empty(), OGRRawPoint::x, and OGRRawPoint::y.

16.24.3.16 OGRBoolean OGRPoint::IsEmpty () const [virtual]

Returns TRUE (non-zero) if the object has no points. Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns:

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. 138).

Referenced by OGRMultiPoint::exportToWkt().

16.24.3.17 void OGRPoint::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters:

nNewDimension New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. 140).

Referenced by clone().

16.24.3.18 void OGRPoint::setX (double *xIn*) [inline]

Assign point X coordinate.

There is no corresponding SFCOM method.

Referenced by OGRPolygon::Centroid(), OGRLineString::getPoint(), OGRPolygon::PointOnSurface(), and OGRLineString::Value().

16.24.3.19 void OGRPoint::setY (double *yIn*) [inline]

Assign point Y coordinate.

There is no corresponding SFCOM method.

Referenced by OGRPolygon::Centroid(), OGRLineString::getPoint(), OGRPolygon::PointOnSurface(), and OGRLineString::Value().

16.24.3.20 void OGRPoint::setZ (double *zIn*) [inline]

Assign point Z coordinate. Calling this method will force the geometry coordinate dimension to 3D (wkbPoint|wkbZ).

There is no corresponding SFCOM method.

Referenced by OGRLineString::getPoint(), and OGRLineString::Value().

16.24.3.21 OGRErr OGRPoint::transform (OGRCoordinateTransformation * *poCT*) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 88) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 224) of the **OGRCoordinateTransformation** (p. 88) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 389).

Parameters:

poCT the transformation to apply.

Returns:

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 141).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), and OGRCoordinateTransformation::Transform().

16.24.3.22 int OGRPoint::WkbSize () const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 390).

Returns:

size of binary representation in bytes.

Implements **OGRGeometry** (p. 143).

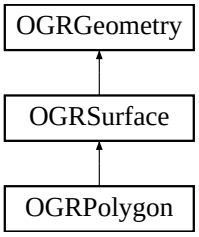
The documentation for this class was generated from the following files:

- **ogr_geometry.h**
- ogrpoint.cpp

16.25 OGRPolygon Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRPolygon::



Public Member Functions

- **OGRPolygon** ()
- virtual const char * **getGeometryName** () const
- virtual **OGRwkbGeometryType getGeometryType** () const
- virtual **OGRGeometry** * **clone** () const
- virtual void **empty** ()
- virtual OGRErr **transform** (**OGRCoordinateTransformation** *poCT)
- virtual void **flattenTo2D** ()
- virtual OGRBoolean **IsEmpty** () const
- virtual void **segmentize** (double dfMaxLength)
- virtual double **get_Area** () const
- virtual int **Centroid** (**OGRPoint** *poPoint) const
- virtual int **PointOnSurface** (**OGRPoint** *poPoint) const
- virtual int **WkbSize** () const
- virtual OGRErr **importFromWkb** (unsigned char *, int=-1)
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char *) const
- virtual OGRErr **importFromWkt** (char **)
- virtual OGRErr **exportToWkt** (char **ppszDstText) const
- virtual int **getDimension** () const
- virtual void **getEnvelope** (**OGREnvelope** *psEnvelope) const
- virtual OGRBoolean **Equals** (**OGRGeometry** *) const
- virtual void **setCoordinateDimension** (int nDimension)
- void **addRing** (**OGRLinearRing** *)
- void **addRingDirectly** (**OGRLinearRing** *)
- **OGRLinearRing** * **getExteriorRing** ()
- int **getNumInteriorRings** () const
- **OGRLinearRing** * **getInteriorRing** (int)
- virtual void **closeRings** ()

16.25.1 Detailed Description

Concrete class representing polygons.

Note that the OpenGIS simple features polygons consist of one outer ring, and zero or more inner rings. A polygon cannot represent disconnected regions (such as multiple islands in a political body). The **OGRMultiPolygon** (p. 194) must be used for this.

16.25.2 Constructor & Destructor Documentation

16.25.2.1 OGRPolygon::OGRPolygon ()

Create an empty polygon.

16.25.3 Member Function Documentation

16.25.3.1 void OGRPolygon::addRing (OGRLinearRing * *poNewRing*)

Add a ring to a polygon.

If the polygon has no external ring (it is empty) this will be used as the external ring, otherwise it is used as an internal ring. The passed **OGRLinearRing** (p. 171) remains the responsibility of the caller (an internal copy is made).

This method has no SFCOM analog.

Parameters:

poNewRing ring to be added to the polygon.

References OGRGeometry::getCoordinateDimension().

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

16.25.3.2 void OGRPolygon::addRingDirectly (OGRLinearRing * *poNewRing*)

Add a ring to a polygon.

If the polygon has no external ring (it is empty) this will be used as the external ring, otherwise it is used as an internal ring. Ownership of the passed ring is assumed by the **OGRPolygon** (p. 206), but otherwise this method operates the same as OGRPolygon::AddRing().

This method has no SFCOM analog.

Parameters:

poNewRing ring to be added to the polygon.

References OGRGeometry::getCoordinateDimension().

Referenced by OGRGeometryFactory::createFromFgf(), OGRMultiPolygon::importFromWkt(), and OGRBuildPolygonFromEdges().

16.25.3.3 int OGRPolygon::Centroid (OGRPoint * *poPoint*) const [virtual]

Compute the polygon centroid.

The centroid location is applied to the passed in **OGRPoint** (p. 198) object.

Returns:

OGRERR_NONE on success or OGRERR_FAILURE on error.

Implements **OGRSurface** (p. 272).

References OGRPoint::getGeometryType(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::setX(), OGRPoint::setY(), and wkbPoint.

16.25.3.4 OGRGeometry * OGRPolygon::clone () const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR_G_Clone()** (p. 377).

Returns:

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. 129).

References addRing(), OGRGeometry::assignSpatialReference(), and OGRGeometry::getSpatialReference().

16.25.3.5 void OGRPolygon::closeRings () [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. 129).

16.25.3.6 void OGRPolygon::empty () [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR_G_Empty()** (p. 379).

Implements **OGRGeometry** (p. 132).

16.25.3.7 OGRBoolean OGRPolygon::Equals (OGRGeometry * *poOtherGeom*) const [virtual]

Returns two if two geometries are equivalent.

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

Returns:

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. 132).

References OGRLineString::Equals(), getExteriorRing(), getGeometryType(), OGRGeometry::getGeometryType(), getInteriorRing(), and getNumInteriorRings().

16.25.3.8 OGRErr OGRPolygon::exportToWkb (OGRwkbByteOrder *eByteOrder*, unsigned char * *pabyData*) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR_G_ExportToWkb()** (p. 380).

Parameters:

eByteOrder One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.

pabyData a buffer into which the binary representation is written. This buffer must be at least **OGRGeometry::WkbSize()** (p. 143) byte in size.

Returns:

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 133).

References OGRLinearRing::_exportToWkb(), OGRLinearRing::_WkbSize(), OGRGeometry::getCoordinateDimension(), and getGeometryType().

16.25.3.9 OGRErr OGRPolygon::exportToWkt (char ** *ppszDstText*) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR_G_ExportToWkt()** (p. 380).

Parameters:

ppszDstText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

Implements **OGRGeometry** (p. 134).

References OGRLineString::exportToWkt(), OGRGeometry::getCoordinateDimension(), getExteriorRing(), IsEmpty(), and OGRLineString::setCoordinateDimension().

16.25.3.10 void OGRPolygon::flattenTo2D () [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR_G_FlattenTo2D()** (p. 381).

Implements **OGRGeometry** (p. 134).

16.25.3.11 double OGRPolygon::get_Area () const [virtual]

Compute area of polygon.

The area is computed as the area of the outer ring less the area of all internal rings.

Returns:

computed area.

Implements **OGRSurface** (p. 272).

References OGRLinearRing::get_Area(), getExteriorRing(), getInteriorRing(), and getNumInteriorRings().

Referenced by OGRMultiPolygon::get_Area().

16.25.3.12 int OGRPolygon::getDimension () const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. 134)).

This method is the same as the C function **OGR_G_GetDimension()** (p. 382).

Returns:

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. 135).

16.25.3.13 void OGRPolygon::getEnvelope (OGREnvelope * *psEnvelope*) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR_G_GetEnvelope()** (p. 382).

Parameters:

psEnvelope the structure in which to place the results.

Implements **OGRGeometry** (p. 135).

References OGRLineString::getEnvelope(), OGREnvelope::MaxX, OGREnvelope::MaxY, OGREnvelope::MinX, and OGREnvelope::MinY.

16.25.3.14 OGRLinearRing * OGRPolygon::getExteriorRing ()

Fetch reference to external polygon ring.

Note that the returned ring pointer is to an internal data object of the **OGRPolygon** (p. 206). 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. 129) 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. 206) is empty.

Referenced by OGRGeometry::dumpReadable(), Equals(), exportToWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), and get_Area().

16.25.3.15 const char * OGRPolygon::getGeometryName () const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR_G_GetGeometryName()** (p. 382).

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. 135).

16.25.3.16 OGRwkbGeometryType OGRPolygon::getGeometryType () const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR_G_GetGeometryType()** (p. 383).

Returns:

the geometry type code.

Implements **OGRGeometry** (p. 136).

References OGRGeometry::getCoordinateDimension(), wkbPolygon, and wkbPolygon25D.

Referenced by Equals(), and exportToWkb().

16.25.3.17 OGRLinearRing * OGRPolygon::getInteriorRing (int *iRing*)

Fetch reference to indicated internal ring.

Note that the returned ring pointer is to an internal data object of the **OGRPolygon** (p. 206). 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. 129) method to make a separate copy within the application.

Relates to the SFCOM IPolygon::get_InternalRing() method.

Parameters:

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

Returns:

pointer to external ring. May be NULL if the **OGRPolygon** (p. 206) is empty.

Referenced by OGRGeometry::dumpReadable(), Equals(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), and get_Area().

16.25.3.18 int OGRPolygon::getNumInteriorRings () const

Fetch the number of internal rings.

Relates to the SFCOM IPolygon::get_NumInteriorRings() method.

Returns:

count of internal rings, zero or more.

Referenced by OGRGeometry::dumpReadable(), Equals(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), get_Area(), and OGRBuildPolygonFromEdges().

16.25.3.19 OGRErr OGRPolygon::importFromWkb (unsigned char * *pabyData*, int *nSize* = -1) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 385).

Parameters:

pabyData the binary input data.

nSize the size of pabyData in bytes, or zero if not known.

Returns:

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

Implements **OGRGeometry** (p. 136).

References OGRLinearRing::_importFromWkb(), OGRLinearRing::_WkbSize(), and wkbPolygon.

16.25.3.20 OGRErr OGRPolygon::importFromWkt (char ** *ppszInput*) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. 154) 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. 386).

Parameters:

ppszInput pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_-UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

Implements **OGRGeometry** (p. 137).

References OGRLineString::setPoints().

16.25.3.21 OGRBoolean OGRPolygon::IsEmpty () const [virtual]

Returns TRUE (non-zero) if the object has no points. Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

Returns:

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. 138).

Referenced by exportToWkt().

16.25.3.22 int OGRPolygon::PointOnSurface (OGRPoint * *poPoint*) const [virtual]

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

NOTE: Only implemented when GEOS included in build.

Parameters:

poPoint point to be set with an internal point.

Returns:

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implements **OGRSurface** (p. 273).

References OGRPoint::getGeometryType(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::setX(), OGRPoint::setY(), and wkbPoint.

16.25.3.23 void OGRPolygon::segmentize (double *dfMaxLength*) [virtual]

Modify the geometry such it has no segment longer then the given distance. Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR_G_Segmentize()** (p. 388)

Parameters:

hGeom handle on the geometry to segmentize

dfMaxLength the maximum distance between 2 points after segmentization

Reimplemented from **OGRGeometry** (p. 140).

16.25.3.24 void OGRPolygon::setCoordinateDimension (int *nNewDimension*) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

Parameters:

nNewDimension New coordinate dimension value, either 2 or 3.

Reimplemented from **OGRGeometry** (p. 140).

16.25.3.25 OGRErr OGRPolygon::transform (OGRCoordinateTransformation * *poCT*) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. 88) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 224) of the **OGRCoordinateTransformation** (p. 88) will be assigned to the geometry.

This method is the same as the C function **OGR_G_Transform()** (p. 389).

Parameters:

poCT the transformation to apply.

Returns:

OGRERR_NONE on success or an error code.

Implements **OGRGeometry** (p. 141).

References OGRGeometry::assignSpatialReference(), OGRCoordinateTransformation::GetTargetCS(), and OGRLineString::transform().

16.25.3.26 int OGRPolygon::WkbSize () const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR_G_WkbSize()** (p. 390).

Returns:

size of binary representation in bytes.

Implements **OGRGeometry** (p. 143).

References OGRLinearRing::_WkbSize(), and OGRGeometry::getCoordinateDimension().

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

- **ogr_geometry.h**
- ogrpolygon.cpp

16.26 OGRRawPoint Class Reference

`#include <ogr_geometry.h>`

16.26.1 Detailed Description

Simple container for a position.

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

- **ogr_geometry.h**

16.27 OGRSFDriver Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- virtual const char * **GetName** ()=0
- virtual **OGRDataSource** * **Open** (const char *pszName, int bUpdate=FALSE)=0
- virtual int **TestCapability** (const char *)=0
- virtual **OGRDataSource** * **CreateDataSource** (const char *pszName, char **=NULL)
- virtual OGRErr **DeleteDataSource** (const char *pszName)

16.27.1 Detailed Description

Represents an operational format driver.

One **OGRSFDriver** (p. 217) derived class will normally exist for each file format registered for use, regardless of whether a file has or will be opened. The list of available drivers is normally managed by the **OGRSFDriverRegistrar** (p. 220).

16.27.2 Member Function Documentation

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

This method attempts to create a new data source based on the passed driver. The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

This method is the same as the C function **OGR_Dr_CreateDataSource()** (p. 347).

Note:

This method does **NOT** attach driver instance to the returned data source, so caller should expect that **OGRDataSource::GetDriver()** (p. 94) will return NULL pointer. In order to attach driver to the returned data soruce, it is required to use C function OGR_Dr_CreateDataSource. This behavior is related to fix of issue reported in `Ticket #1233`.

Parameters:

pszName the name for the new data source.

papszOptions a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns:

NULL is returned on failure, or a new **OGRDataSource** (p. 92) on success.

Referenced by OGR_Dr_CreateDataSource().

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

Destroy a datasource.

Destroy the named datasource. Normally it would be safest if the datasource was not open at the time.

Whether this is a supported operation on this driver case be tested using **TestCapability()** (p. 219) on ODrCDeleteDataSource.

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

Parameters:

pszDataSource the name of the datasource to delete.

Returns:

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

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

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

This method is the same as the C function **OGR_Dr_GetName()** (p. 348).

Returns:

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

Referenced by OGRSFDriverRegistrar::Open(), and OGRSFDriverRegistrar::RegisterDriver().

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

Attempt to open file with this driver.

This method is what **OGRSFDriverRegistrar** (p. 220) uses to implement its **Open()** (p. 218) method. See it for more details.

Note, drivers do not normally set their own m_poDriver value, so a direct call to this method (instead of indirectly via **OGRSFDriverRegistrar** (p. 220)) will usually result in a datasource that does not know what driver it relates to if GetDriver() is called on the datasource. The application may directly call SetDriver() after opening with this method to avoid this problem.

This method is the same as the C function **OGR_Dr_Open()** (p. 348).

Parameters:

pszName the name of the file, or data source to try and open.

bUpdate TRUE if update access is required, otherwise FALSE (the default).

Returns:

NULL on error or if the pass name is not supported by this driver, otherwise a pointer to an **OGRDataSource** (p. 92). This **OGRDataSource** (p. 92) should be closed by deleting the object when it is no longer needed.

Referenced by OGRSFDriverRegistrar::Open().

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

Test if capability is available.

One of the following data source capability names can be passed into this method, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

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

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

This method is the same as the C function **OGR_Dr_TestCapability**() (p. 349).

Parameters:

pszCapability the capability to test.

Returns:

TRUE if capability available otherwise FALSE.

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

- **ogrsf_frmts.h**
- ogrsf_frmts.dox
- ogrsfdriver.cpp

16.28 OGRSFDriverRegistrar Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- void **RegisterDriver** (**OGRSFDriver** *poDriver)
- int **GetDriverCount** (void)
- **OGRSFDriver** * **GetDriver** (int iDriver)
- void **AutoLoadDrivers** ()

Static Public Member Functions

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

16.28.1 Detailed Description

Singleton manager for **OGRSFDriver** (p. 217) instances that will be used to try and open datasources. Normally the registrar is populated with standard drivers using the **OGRRegisterAll()** (p. 408) function and does not need to be directly accessed. The driver registrar and all registered drivers may be cleaned up on shutdown using **OGRCleanupAll()** (p. 406).

16.28.2 Member Function Documentation

16.28.2.1 void OGRSFDriverRegistrar::AutoLoadDrivers ()

Auto-load GDAL drivers from shared libraries.

This function will automatically load drivers from shared libraries. It searches the "driver path" for .so (or .dll) files that start with the prefix "ogr_X.so". It then tries to load them and then tries to call a function within them called RegisterOGRX() where the 'X' is the same as the remainder of the shared library basename, or failing that to call GDALRegisterMe().

There are a few rules for the driver path. If the GDAL_DRIVER_PATH environment variable it set, it is taken to be a list of directories to search separated by colons on unix, or semi-colons on Windows.

If that is not set the following defaults are used:

- Linux/Unix: <prefix>/lib/gdalplugins is searched or /usr/local/lib/gdalplugins if the install prefix is not known.
- MacOSX: <prefix>/PlugIns is searched, or /usr/local/lib/gdalplugins if the install prefix is not known. Also, the framework directory /Library/Application Support/GDAL/PlugIns is searched.
- Win32: <prefix>/lib/gdalplugins if the prefix is known (normally it is not), otherwise the gdalplugins subdirectory of the directory containing the currently running executable is used.

References CPLFormFilename(), CPLGetBasename(), CPLGetDirname(), CPLGetExecPath(), CPLGetExtension(), and CPLGetSymbol().

Referenced by OGRRegisterAll().

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

Fetch the indicated driver.

This method is the same as the C function **OGRGetDriver()** (p. 406).

Parameters:

iDriver the driver index, from 0 to **GetDriverCount()** (p. 221)-1.

Returns:

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

Referenced by OGRGetDriver().

16.28.2.3 int OGRSFDriverRegistrar::GetDriverCount (void)

Fetch the number of registered drivers.

This method is the same as the C function **OGRGetDriverCount()** (p. 407).

Returns:

the drivers count.

Referenced by OGRGetDriverCount().

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

Return the driver manager, creating one if none exist.

Returns:

the driver manager.

Fetch registrar.

This static method should be used to fetch the singleton registrar. It will create a registrar if there is not already one in existance.

Returns:

the current driver registrar.

Referenced by OGRRegisterAll(), OGRRegisterDriver(), Open(), and OGRDataSource::Release().

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

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

This method loops through all the drivers registered with the driver manager trying each until one succeeds with the given data source. This method is static. Applications don't normally need to use any other

OGRSFDriverRegistrar (p. 220) methods directly, not do they normally need to have a pointer to an **OGRSFDriverRegistrar** (p. 220) instance.

If this method fails, **CPLGetLastErrorMsg()** (p. 299) can be used to check if there is an error message explaining why.

This method is the same as the C function **OGROpen()** (p. 407).

Parameters:

pszName the name of the file, or data source to open.

bUpdate FALSE for read-only access (the default) or TRUE for read-write access.

ppoDriver if non-NULL, this argument will be updated with a pointer to the driver which was used to open the data source.

Returns:

NULL on error or if the pass name is not supported by this driver, otherwise a pointer to an **OGRDataSource** (p. 92). This **OGRDataSource** (p. 92) should be closed by deleting the object when it is no longer needed.

Example:

```
OGRDataSource (p. 92) *poDS;

poDS = OGRSFDriverRegistrar::Open (p. 221)( "polygon.shp" );
if( poDS == NULL )
{
    return;
}

... use the data source ...

delete poDS;
```

References OGRDataSource::GetDriver(), OGRSFDriver::GetName(), GetRegistrar(), OGRDataSource::m_poDriver, nDrivers, OGRSFDriver::Open(), papoDrivers, and OGRDataSource::Reference().

Referenced by OGROpen().

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

Add a driver to the list of registered drivers.

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

This method is the same as the C function **OGRRegisterDriver()** (p. 408).

Parameters:

poDriver the driver to add.

References OGRSFDriver::GetName().

Referenced by OGRRegisterDriver().

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

- **ogrsf_frmts.h**
- ogrsf_frmts.dox
- ogrsfdriverregistrar.cpp

16.29 OGRSpatialReference Class Reference

`#include <ogr_spatialref.h>`

Public Member Functions

- **OGRSpatialReference** (const char *=NULL)
- virtual ~**OGRSpatialReference** ()
- int **Reference** ()
- int **Dereference** ()
- int **GetReferenceCount** () const
- void **Release** ()
- **OGRSpatialReference** * **Clone** () const
- OGRErr **exportToWkt** (char **) const
- OGRErr **exportToProj4** (char **) const
- OGRErr **exportToPCI** (char **, char **, double **) const
- OGRErr **exportToUSGS** (long *, long *, double **, long *) const
- OGRErr **exportToPanorama** (long *, long *, long *, long *, double *) const
- OGRErr **exportToERM** (char *pszProj, char *pszDatum, char *pszUnits)
- OGRErr **exportToMICoordSys** (char **) const
- OGRErr **importFromWkt** (char **)
- OGRErr **importFromProj4** (const char *)
- OGRErr **importFromEPSG** (int)
- OGRErr **importFromEPSGA** (int)
- OGRErr **importFromESRI** (char **)
- OGRErr **importFromPCI** (const char *, const char *=NULL, double *=NULL)
- OGRErr **importFromUSGS** (long iProjSys, long iZone, double *padfPrjParams, long iDatum, int bAnglesInPackedDMSFormat=TRUE)
- OGRErr **importFromPanorama** (long, long, long, double *)
- OGRErr **importFromDict** (const char *pszDict, const char *pszCode)
- OGRErr **importFromURN** (const char *)
- OGRErr **importFromERM** (const char *pszProj, const char *pszDatum, const char *pszUnits)
- OGRErr **importFromUrl** (const char *)
- OGRErr **importFromMICoordSys** (const char *)
- OGRErr **morphToESRI** ()
- OGRErr **morphFromESRI** ()
- OGRErr **Validate** ()
- OGRErr **StripCTParms** (**OGR_SRSNode** *=NULL)
- OGRErr **FixupOrdering** ()
- OGRErr **Fixup** ()
- int **EPSGTreatsAsLatLong** ()
- const char * **GetAxis** (const char *pszTargetKey, int iAxis, OGRAxisOrientation *peOrientation)
- OGRErr **SetAxes** (const char *pszTargetKey, const char *pszXAxisName, OGRAxisOrientation eXAxisOrientation, const char *pszYAxisName, OGRAxisOrientation eYAxisOrientation)
- void **SetRoot** (**OGR_SRSNode** *)
- **OGR_SRSNode** * **GetAttrNode** (const char *)
- const char * **GetAttrValue** (const char *, int=0) const
- OGRErr **SetNode** (const char *, const char *)
- OGRErr **SetLinearUnitsAndUpdateParameters** (const char *pszName, double dfInMeters)
- OGRErr **SetLinearUnits** (const char *pszName, double dfInMeters)

- double **GetLinearUnits** (char **=NULL) const
- OGRErr **SetAngularUnits** (const char *pszName, double dfInRadians)
- double **GetAngularUnits** (char **=NULL) const
- double **GetPrimeMeridian** (char **=NULL) const
- int **IsGeographic** () const
- int **IsProjected** () const
- int **IsLocal** () const
- int **IsSameGeogCS** (const **OGRSpatialReference** *) const
- int **IsSame** (const **OGRSpatialReference** *) const
- void **Clear** ()
- OGRErr **SetLocalCS** (const char *)
- OGRErr **SetProjCS** (const char *)
- OGRErr **SetProjection** (const char *)
- OGRErr **SetGeogCS** (const char *pszGeogName, const char *pszDatumName, const char *pszEllipsoidName, double dfSemiMajor, double dfInvFlattening, const char *pszPMName=NULL, double dfPMOffset=0.0, const char *pszUnits=NULL, double dfConvertToRadians=0.0)
- OGRErr **SetWellKnownGeogCS** (const char *)
- OGRErr **CopyGeogCSFrom** (const **OGRSpatialReference** *poSrcSRS)
- OGRErr **SetFromUserInput** (const char *)
- OGRErr **SetTOWGS84** (double, double, double, double=0.0, double=0.0, double=0.0, double=0.0)
- OGRErr **GetTOWGS84** (double *padfCoef, int nCoeff=7) const
- double **GetSemiMajor** (OGRErr *=NULL) const
- double **GetSemiMinor** (OGRErr *=NULL) const
- double **GetInvFlattening** (OGRErr *=NULL) const
- OGRErr **SetAuthority** (const char *pszTargetKey, const char *pszAuthority, int nCode)
- OGRErr **AutoIdentifyEPSG** ()
- const char * **GetAuthorityCode** (const char *pszTargetKey) const
- const char * **GetAuthorityName** (const char *pszTargetKey) const
- const char * **GetExtension** (const char *pszTargetKey, const char *pszName, const char *pszDefault=NULL) const
- OGRErr **SetProjParm** (const char *, double)
- double **GetProjParm** (const char *, double=0.0, OGRErr *=NULL) const
- OGRErr **SetNormProjParm** (const char *, double)
- double **GetNormProjParm** (const char *, double=0.0, OGRErr *=NULL) const
- OGRErr **SetACEA** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetAE** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetBonne** (double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetCEA** (double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetCS** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEC** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEckert** (int nVariation, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEquirectangular** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **SetEquirectangular2** (double dfCenterLat, double dfCenterLong, double dfPseudoStdParallel1, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGEOS** (double dfCentralMeridian, double dfSatelliteHeight, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGH** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **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)
- OGRErr **SetHOM2PNO** (double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetIWMPolyconic** (double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetKrovak** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLAEA** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCC** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCC1SP** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCCB** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMC** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMercator** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMollweide** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetNZMG** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetOS** (double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetOrthographic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetPolyconic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetPS** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetRobinson** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetSinusoidal** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetStereographic** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetSOC** (double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTM** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMVariant** (const char *pszVariantName, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **SetTMG** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMSO** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTPED** (double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetVDG** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetUTM** (int nZone, int bNorth=TRUE)
- int **GetUTMZone** (int *pbNorth=NULL) const
- OGRErr **SetWagner** (int nVariation, double dfCenterLat, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetStatePlane** (int nZone, int bNAD83=TRUE, const char *pszOverrideUnitName=NULL, double dfOverrideUnit=0.0)

16.29.1 Detailed Description

This class respresents a OpenGIS Spatial Reference System, and contains methods for converting between this object organization and well known text (WKT) format. This object is reference counted as one instance of the object is normally shared between many **OGRGeometry** (p. 127) objects.

Normally application code can fetch needed parameter values for this SRS using **GetAttrValue()** (p. 234), but in special cases the underlying parse tree (or **OGR_SRSNode** (p. 81) objects) can be accessed more directly.

See `the tutorial` for more information on how to use this class.

16.29.2 Constructor & Destructor Documentation

16.29.2.1 OGRSpatialReference::OGRSpatialReference (const char * *pszWKT* = `NULL`)

Constructor.

This constructor takes an optional string argument which if passed should be a WKT representation of an SRS. Passing this is equivelent to not passing it, and then calling **importFromWkt()** (p. 250) with the WKT string.

Note that newly created objects are given a reference count of one.

The C function OSRNewSpatialReference() does the same thing as this constructor.

Parameters:

pszWKT well known text definition to which the object should be initialized, or NULL (the default).

References importFromWkt().

16.29.2.2 OGRSpatialReference::~OGRSpatialReference () `[virtual]`

OGRSpatialReference (p. 224) destructor.

The C function OSRDestroySpatialReference() does the same thing as this method.

16.29.3 Member Function Documentation

16.29.3.1 OGRErr OGRSpatialReference::AutoIdentifyEPSG ()

Set EPSG authority info if possible.

This method inspects a WKT definition, and adds EPSG authority nodes where an aspect of the coordinate system can be easily and safely corresponded with an EPSG identifier. In practice, this method will evolve over time. In theory it can add authority nodes for any object (ie. spheroid, datum, GEOGCS, units, and PROJCS) that could have an authority node. Mostly this is useful to inserting appropriate PROJCS codes for common formulations (like UTM n WGS84).

If it success the **OGRSpatialReference** (p. 224) 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. 299).

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

Returns:

OGRERR_NONE or OGRERR_UNSUPPORTED_SRS.

References GetAuthorityCode(), GetAuthorityName(), GetUTMZone(), IsGeographic(), IsProjected(), and SetAuthority().

16.29.3.2 void OGRSpatialReference::Clear ()

Wipe current definition.

Returns **OGRSpatialReference** (p. 224) to a state with no definition, as it exists when first created. It does not affect reference counts.

Referenced by CopyGeogCSFrom(), importFromERM(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromWkt(), SetFromUserInput(), SetGeogCS(), and SetStatePlane().

16.29.3.3 OGRSpatialReference * OGRSpatialReference::Clone () const

Make a duplicate of this **OGRSpatialReference** (p. 224).

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

Returns:

a new SRS, which becomes the responsibility of the caller.

References OGR_SRSNode::Clone(), and poRoot.

16.29.3.4 OGRErr OGRSpatialReference::CopyGeogCSFrom (const OGRSpatialReference * *poSrcSRS*)

Copy GEOGCS from another **OGRSpatialReference** (p. 224).

The GEOGCS information is copied into this **OGRSpatialReference** (p. 224) from another. If this object has a PROJCS root already, the GEOGCS is installed within it, otherwise it is installed as the root.

Parameters:

poSrcSRS the spatial reference to copy the GEOGCS information from.

Returns:

OGRERR_NONE on success or an error code.

References Clear(), OGR_SRSNode::Clone(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::InsertChild(), and SetRoot().

Referenced by importFromERM(), importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and SetWellKnownGeogCS().

16.29.3.5 int OGRSpatialReference::Dereference ()

Decrements the reference count by one.

The method does the same thing as the C function OSRDereference().

Returns:

the updated reference count.

Referenced by Release().

16.29.3.6 int OGRSpatialReference::EPSGTreatsAsLatLong ()

This method returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.

Currently this returns TRUE for all geographic coordinate systems with an EPSG code set, and AXIS values set defining it as lat, long. Note that coordinate systems with an EPSG code and no axis settings will be assumed to not be lat/long.

FALSE will be returned for all coordinate systems that are not geographic, or that do not have an EPSG code set.

Returns:

TRUE or FALSE.

References GetAttrNode(), GetAuthorityName(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetValue(), and IsGeographic().

16.29.3.7 OGRErr OGRSpatialReference::exportToERM (char * *pszProj*, char * *pszDatum*, char * *pszUnits*)

Convert coordinate system to ERMapper format.

Parameters:

pszProj 32 character buffer to receive projection name.

pszDatum 32 character buffer to recieve datum name.

pszUnits 32 character buffer to receive units name.

Returns:

OGRERR_NONE on success, OGRERR_SRS_UNSUPPORTED if not translation is found, or OGRERR_FAILURE on other failures.

References GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), GetLinearUnits(), GetUTMZone(), importFromDict(), IsGeographic(), and IsProjected().

16.29.3.8 OGRErr OGRSpatialReference::exportToMICoordSys (char ** *ppszResult*) const

Export coordinate system in Mapinfo style CoordSys format.

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

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

Parameters:

ppszResult pointer to which dynamically allocated Mapinfo CoordSys definition will be assigned.

Returns:

OGRERR_NONE on success, OGRERR_FAILURE on failure, OGRERR_UNSUPPORTED_OPERATION if MITAB library was not linked in.

16.29.3.9 OGRErr OGRSpatialReference::exportToPanorama (long * *piProjSys*, long * *piDatum*, long * *piEllips*, long * *piZone*, double * *padfPrjParams*) const

Export coordinate system in "Panorama" GIS projection definition.

This method is the equivalent of the C function OSRExportToPanorama().

Parameters:

piProjSys Pointer to variable, where the projection system code will be returned.

piDatum Pointer to variable, where the coordinate system code will be returned.

piEllips Pointer to variable, where the spheroid code will be returned.

piZone Pointer to variable, where the zone for UTM projection system will be returned.

padfPrjParams an existing 7 double buffer into which the projection parameters will be placed. See **importFromPanorama()** (p. 243) for the list of parameters.

Returns:

OGRERR_NONE on success or an error code on failure.

References GetAttrValue(), GetInvFlattening(), GetNormProjParm(), GetSemiMajor(), GetUTMZone(), and IsLocal().

16.29.3.10 OGRErr OGRSpatialReference::exportToPCI (char ** *ppszProj*, char ** *ppszUnits*, double ** *ppadfPrjParams*) const

Export coordinate system in PCI projection definition.

Converts the loaded coordinate reference system into PCI projection definition to the extent possible. The strings returned in ppszProj, ppszUnits and ppadfPrjParams array should be deallocated by the caller with CPLFree() when no longer needed.

LOCAL_CS coordinate systems are not translatable. An empty string will be returned along with OGRERR_NONE.

This method is the equivelent of the C function OSRExportToPCI().

Parameters:

ppszProj pointer to which dynamically allocated PCI projection definition will be assigned.

ppszUnits pointer to which dynamically allocated units definition will be assigned.

ppadfPrjParams pointer to which dynamically allocated array of 17 projection parameters will be assigned. See **importFromPCI()** (p. 244) for the list of parameters.

Returns:

OGRERR_NONE on success or an error code on failure.

References GetAttrNode(), GetAttrValue(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetInvFlattening(), GetLinearUnits(), GetNormProjParm(), GetSemiMajor(), GetUTMZone(), OGR_SRSNode::GetValue(), and IsLocal().

16.29.3.11 OGRErr OGRSpatialReference::exportToProj4 (char ** *ppszProj4*) const

Export coordinate system in PROJ.4 format.

Converts the loaded coordinate reference system into PROJ.4 format to the extent possible. The string returned in ppszProj4 should be deallocated by the caller with CPLFree() when no longer needed.

LOCAL_CS coordinate systems are not translatable. An empty string will be returned along with OGRERR_NONE.

This method is the equivelent of the C function OSRExportToProj4().

Parameters:

ppszProj4 pointer to which dynamically allocated PROJ.4 definition will be assigned.

Returns:

OGRERR_NONE on success or an error code on failure.

References CPLAtof(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetExtension(), GetInvFlattening(), GetLinearUnits(), GetNormProjParm(), GetSemiMajor(), GetSemiMinor(), GetUTMZone(), OGR_SRSNode::GetValue(), and IsGeographic().

16.29.3.12 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().

Parameters:

piProjSys Pointer to variable, where the projection system code will be returned.

piZone Pointer to variable, where the zone for UTM and State Plane projection systems will be returned.

ppadfPrjParams Pointer to which dynamically allocated array of 15 projection parameters will be assigned. See **importFromUSGS()** (p. 246) for the list of parameters. Caller responsible to free this array.

Returns:

OGRERR_NONE on success or an error code on failure.

References GetAttrValue(), GetInvFlattening(), GetNormProjParm(), GetSemiMajor(), GetUTMZone(), and IsLocal().

16.29.3.13 OGRErr OGRSpatialReference::exportToWkt (char ** *ppszResult*) const

Convert this SRS into WKT format.

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

This method is the same as the C function **OSRExportToWkt()** (p. 421).

Parameters:

ppszResult the resulting string is returned in this pointer.

Returns:

currently OGRERR_NONE is always returned, but the future it is possible error conditions will develop.

References OGR_SRSNode::exportToWkt().

16.29.3.14 OGRErr OGRSpatialReference::Fixup ()

Fixup as needed.

Some mechanisms to create WKT using **OGRSpatialReference** (p. 224), 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()) so that the resulting WKT is valid.

This method should be expected to evolve over time to as problems are discovered. The following are amoung the fixup actions this method will take:

- Fixup the ordering of nodes to match the BNF WKT ordering, using the **FixupOrdering()** (p. 233) method.
- Add missing linear or angular units nodes.

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

Returns:

OGRERR_NONE on success or an error code if something goes wrong.

References CPLAtof(), OGR_SRSNode::FindChild(), FixupOrdering(), GetAttrNode(), SetAngularUnits(), and SetLinearUnits().

Referenced by morphToESRI().

16.29.3.15 OGRErr OGRSpatialReference::FixupOrdering ()

Correct parameter ordering to match CT Specification.

Some mechanisms to create WKT using **OGRSpatialReference** (p. 224), 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().

Returns:

OGRERR_NONE on success or an error code if something goes wrong.

References OGR_SRSNode::FixupOrdering().

Referenced by Fixup(), importFromEPSGA(), importFromPanorama(), importFromPCI(), importFromUSGS(), and morphFromESRI().

16.29.3.16 double OGRSpatialReference::GetAngularUnits (char ** *ppszName* = NULL) const

Fetch angular geographic coordinate system units.

If no units are available, a value of "degree" and SRS_UA_DEGREE_CONV will be assumed. This method only checks directly under the GEOGCS node for units.

This method does the same thing as the C function OSRGetAngularUnits().

Parameters:

ppszName a pointer to be updated with the pointer to the units name. The returned value remains internal to the **OGRSpatialReference** (p. 224) and shouldn't be freed, or modified. It may be invalidated on the next **OGRSpatialReference** (p. 224) call.

Returns:

the value to multiply by angular distances to transform them to radians.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by morphToESRI().

16.29.3.17 OGR_SRSNode * OGRSpatialReference::GetAttrNode (const char * *pszNodePath*)

Find named node in tree.

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

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

Parameters:

pszNodePath the name of the node to search for. May contain multiple components such as "GEOGCS|UNITS".

Returns:

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

References OGR_SRSNode::GetNode().

Referenced by CopyGeogCSFrom(), EPSGTreatsAsLatLong(), exportToPCI(), exportToProj4(), Fixup(), GetAngularUnits(), GetAttrValue(), GetInvFlattening(), GetLinearUnits(), GetPrimeMeridian(), GetProjParm(), GetSemiMajor(), GetTOWGS84(), importFromEPSG(), importFromESRI(), importFromProj4(), IsSame(), morphFromESRI(), morphToESRI(), SetAngularUnits(), SetAuthority(), SetGeogCS(), SetLinearUnits(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetProjCS(), SetProjection(), SetProjParm(), SetStatePlane(), and SetTOWGS84().

16.29.3.18 const char * OGRSpatialReference::GetAttrValue (const char * *pszNodeName*, int *iAttr* = 0) const

Fetch indicated attribute of named node.

This method uses **GetAttrNode()** (p. 233) 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().

Parameters:

pszNodeName the tree node to look for (case insensitive).

iAttr the child of the node to fetch (zero based).

Returns:

the requested value, or NULL if it fails for any reason.

References GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToERM(), exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetUTMZone(), IsSame(), IsSameGeogCS(), morphFromESRI(), morphToESRI(), and SetUTM().

16.29.3.19 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().

Parameters:

pszTargetKey the partial or complete path to the node to get an authority from. ie. "PROJCS", "GEOGCS", "GEOGCS|UNIT" or NULL to search for an authority node on the root element.

Returns:

value code from authority node, or NULL on failure. The value returned is internal and should not be freed or modified.

References OGR_SRSNode::FindChild(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by AutoIdentifyEPSG(), exportToERM(), exportToProj4(), and morphToESRI().

16.29.3.20 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().

Parameters:

pszTargetKey the partial or complete path to the node to get an authority from. ie. "PROJCS", "GEOGCS", "GEOGCS|UNIT" or NULL to search for an authority node on the root element.

Returns:

value code from authority node, or NULL on failure. The value returned is internal and should not be freed or modified.

References OGR_SRSNode::FindChild(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by AutoIdentifyEPSG(), EPSGTreatsAsLatLong(), exportToERM(), exportToProj4(), importFromEPSGA(), and morphToESRI().

16.29.3.21 const char * OGRSpatialReference::GetAxis (const char * *pszTargetKey*, int *iAxis*, OGRAxisOrientation * *peOrientation*)

Fetch the orientation of one axis.

Fetches the the request axis (iAxis - zero based) from the indicated portion of the coordinate system (pszTargetKey) which should be either "GEOGCS" or "PROJCS".

No CPLError is issued on routine failures (such as not finding the AXIS).

This method is equivelent to the C function OSRGetAxis().

Parameters:

pszTargetKey the coordinate system part to query ("PROJCS" or "GEOGCS").

iAxis the axis to query (0 for first, 1 for second).

peOrientation location into which to place the fetch orientation, may be NULL.

Returns:

the name of the axis or NULL on failure.

References OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

16.29.3.22 const char * OGRSpatialReference::GetExtension (const char * *pszTargetKey*, const char * *pszName*, const char * *pszDefault* = NULL) const

Fetch extension value.

Fetch the value of the named EXTENSION item for the identified target node.

Parameters:

pszTargetKey the name or path to the parent node of the EXTENSION.

pszName the name of the extension being fetched.

pszDefault the value to return if the extension is not found.

Returns:

node value if successful or pszDefault on failure.

References OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToProj4().

16.29.3.23 double OGRSpatialReference::GetInvFlattening (OGRErr * *pnErr* = NULL) const

Get spheroid inverse flattening.

This method does the same thing as the C function OSRGetInvFlattening().

Parameters:

pnErr if non-NULL set to OGRERR_FAILURE if no inverse flattening can be found.

Returns:

inverse flattening, or SRS_WGS84_INVFLATTENING if it can't be found.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), and GetSemiMinor().

16.29.3.24 double OGRSpatialReference::GetLinearUnits (char ** *ppszName* = NULL) const

Fetch linear projection units.

If no units are available, a value of "Meters" and 1.0 will be assumed. This method only checks directly under the PROJCS or LOCAL_CS node for units.

This method does the same thing as the C function OSRGetLinearUnits()/

Parameters:

ppszName a pointer to be updated with the pointer to the units name. The returned value remains internal to the **OGRSpatialReference** (p. 224) and shouldn't be freed, or modified. It may be invalidated on the next **OGRSpatialReference** (p. 224) call.

Returns:

the value to multiply by linear distances to transform them to meters.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToERM(), exportToPCI(), exportToProj4(), importFromESRI(), importFromProj4(), IsSame(), morphToESRI(), SetLinearUnitsAndUpdateParameters(), and SetStatePlane().

16.29.3.25 double OGRSpatialReference::GetNormProjParm (const char * *pszName*, double *dfDefaultValue* = 0.0, OGRErr * *pnErr* = NULL) const

Fetch a normalized projection parameter value.

This method is the same as **GetProjParm()** (p. 238) 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().

Parameters:

pszName the name of the parameter to fetch, from the set of SRS_PP codes in **ogr_srs_api.h** (p. 418).

dfDefaultValue the value to return if this parameter doesn't exist.

pnErr place to put error code on failure. Ignored if NULL.

Returns:

value of parameter.

References GetProjParm().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetUTMZone(), morphToESRI(), and SetStatePlane().

16.29.3.26 double OGRSpatialReference::GetPrimeMeridian (char ** *ppszName* = NULL) const

Fetch prime meridian info.

Returns the offset of the prime meridian from greenwich in degrees, and the prime meridian name (if requested). If no PRIMEM value exists in the coordinate system definition a value of "Greenwich" and an offset of 0.0 is assumed.

If the prime meridian name is returned, the pointer is to an internal copy of the name. It should not be freed, altered or depended on after the next OGR call.

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

Parameters:

ppszName return location for prime meridian name. If NULL, name is not returned.

Returns:

the offset to the GEOGCS prime meridian from greenwich in decimal degrees.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

16.29.3.27 double OGRSpatialReference::GetProjParm (const char * *pszName*, double *dfDefaultValue* = 0.0, OGRErr * *pnErr* = NULL) const

Fetch a projection parameter value.

NOTE: This code should be modified to translate non degree angles into degrees based on the GEOGCS unit. This has not yet been done.

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

Parameters:

pszName the name of the parameter to fetch, from the set of SRS_PP codes in **ogr_srs_api.h** (p. 418).

dfDefaultValue the value to return if this parameter doesn't exist.

pnErr place to put error code on failure. Ignored if NULL.

Returns:

value of parameter.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), and OGR_SRSNode::GetValue().

Referenced by GetNormProjParm(), GetUTMZone(), importFromProj4(), IsSame(), morphFromESRI(), morphToESRI(), and SetLinearUnitsAndUpdateParameters().

16.29.3.28 int OGRSpatialReference::GetReferenceCount () const [inline]

Fetch current reference count.

Returns:

the current reference count.

16.29.3.29 double OGRSpatialReference::GetSemiMajor (OGRErr * *pnErr* = NULL) const

Get spheroid semi major axis.

This method does the same thing as the C function OSRGetSemiMajor().

Parameters:

pnErr if non-NULL set to OGRERR_FAILURE if semi major axis can be found.

Returns:

semi-major axis, or SRS_WGS84_SEMIMAJOR if it can't be found.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), and GetSemiMinor().

16.29.3.30 double OGRSpatialReference::GetSemiMinor (OGRErr * *pnErr* = NULL) const

Get spheroid semi minor axis.

This method does the same thing as the C function OSRGetSemiMinor().

Parameters:

pnErr if non-NULL set to OGRERR_FAILURE if semi minor axis can be found.

Returns:

semi-minor axis, or WGS84 semi minor if it can't be found.

References GetInvFlattening(), and GetSemiMajor().

Referenced by exportToProj4().

16.29.3.31 OGRErr OGRSpatialReference::GetTOWGS84 (double * *padfCoeff*, int *nCoeffCount* = 7) const

Fetch TOWGS84 parameters, if available.

Parameters:

padfCoeff array into which up to 7 coefficients are placed.

nCoeffCount size of padfCoeff - defaults to 7.

Returns:

OGRERR_NONE on success, or OGRERR_FAILURE if there is no TOWGS84 node available.

References CPLAtof(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::GetValue().

16.29.3.32 int OGRSpatialReference::GetUTMZone (int * *pbNorth* = NULL) const

Get utm zone information.

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

Parameters:

pbNorth pointer to in to set to TRUE if northern hemisphere, or FALSE if southern.

Returns:

UTM zone number or zero if this isn't a UTM definition.

References GetAttrValue(), GetNormProjParm(), and GetProjParm().

Referenced by AutoIdentifyEPSG(), exportToERM(), exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), and morphToESRI().

16.29.3.33 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. 224).

More complete information on the format of the dictionary files can be found in the epsg.wkt file in the GDAL data tree. The dictionary files are searched for in the "GDAL" domain using CPLFindFile(). Normally this results in searching /usr/local/share/gdal or somewhere similar.

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

Parameters:

pszDictFile the name of the dictionary file to load.

pszCode the code to lookup in the dictionary.

Returns:

OGRERR_NONE on success, or OGRERR_SRS_UNSUPPORTED if the code isn't found, and OGRERR_SRS_FAILURE if something more dramatic goes wrong.

References importFromWkt().

Referenced by exportToERM(), importFromEPSGA(), importFromERM(), importFromURN(), and SetFromUserInput().

16.29.3.34 OGRErr OGRSpatialReference::importFromEPSG (int *nCode*)

Initialize SRS based on EPSG GCS or PCS code.

This method will initialize the spatial reference based on the passed in EPSG GCS or PCS code. The coordinate system definitions are normally read from the EPSG derived support files such as pcs.csv, gcs.csv, pcs.override.csv, gcs.override.csv and falling back to search for a PROJ.4 epsg init file or a definition in epsg.wkt.

These support files are normally searched for in /usr/local/share/gdal or in the directory identified by the GDAL_DATA configuration option. See CPLFindFile() for details.

This method is relatively expensive, and generally involves quite a bit of text file scanning. Reasonable efforts should be made to avoid calling it many times for the same coordinate system.

This method is similar to **importFromEPSGA()** (p. 241) 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).

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

Parameters:

nCode a GCS or PCS code from the horizontal coordinate system table.

Returns:

OGRERR_NONE on success, or an error code on failure.

References GetAttrNode(), importFromEPSGA(), and OGR_SRSNode::StripNodes().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), SetFromUserInput(), SetStatePlane(), and SetWellKnownGeogCS().

16.29.3.35 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 **importFromEPSGA()** (p. 241) except that EPSG preferred axis ordering ∗will∗ be applied for geographic coordinate systems. EPSG normally defines geographic coordinate systems to use lat/long contrary to typical GIS use). See **OGRSpatialReference::importFromEPSG()** (p. 240) for more details on operation of this method.

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

Parameters:

nCode a GCS or PCS code from the horizontal coordinate system table.

Returns:

OGRERR_NONE on success, or an error code on failure.

References FixupOrdering(), GetAuthorityName(), importFromDict(), importFromProj4(), IsGeographic(), IsProjected(), and SetAuthority().

Referenced by importFromEPSG(), importFromURN(), SetFromUserInput(), and SetWellKnownGeogCS().

16.29.3.36 OGRErr OGRSpatialReference::importFromERM (const char ∗ *pszProj*, const char ∗ *pszDatum*, const char ∗ *pszUnits*)

OGR WKT from ERMapper projection definitions.

Generates an **OGRSpatialReference** (p. 224) definition from an ERMapper datum and projection name. Based on the ecw_cs.wkt dictionary file from gdal/data.

Parameters:

pszProj the projection name, such as "NUTM11" or "GEOGRAPHIC".

pszDatum the datum name, such as "NAD83".

pszUnits the linear units "FEET" or "METERS".

Returns:

OGRERR_NONE on success or OGRERR_UNSUPPORTED_SRS if not found.

References Clear(), CopyGeogCSFrom(), importFromDict(), IsLocal(), and SetLinearUnits().

16.29.3.37 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. 224) 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. 242) by an automatical call to **morphFromESRI()** (p. 252).

Currently only GEOGRAPHIC, UTM, STATEPLANE, GREATBRITIAN_GRID, ALBERS, EQUIDISTANT_CONIC, and TRANSVERSE (mercator) projections are supported from old style files.

At this time there is no equivelent exportToESRI() method. Writing old style .prj files is not supported by **OGRSpatialReference** (p. 224). However the **morphToESRI()** (p. 253) and **exportToWkt()** (p. 232) methods can be used to generate output suitable to write to new style (Arc 8) .prj files.

This function is the equilvelent of the C function OSRImportFromESRI().

Parameters:

papszPrj NULL terminated list of strings containing the definition.

Returns:

OGRERR_NONE on success or an error code in case of failure.

References CopyGeogCSFrom(), OGR_SRSNode::DestroyChild(), GetAttrNode(), GetLinearUnits(), importFromEPSG(), importFromWkt(), IsLocal(), IsProjected(), morphFromESRI(), SetACEA(), SetEC(), SetLCC(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetPS(), SetStatePlane(), SetTM(), SetUTM(), and SetWellKnownGeogCS().

16.29.3.38 OGRErr OGRSpatialReference::importFromMICoordSys (const char * *pszCoordSys*)

Import Mapinfo style CoordSys definition.

The **OGRSpatialReference** (p. 224) is initialized from the passed Mapinfo style CoordSys definition string.

This method is the equivelent of the C function OSRImportFromMICoordSys().

Parameters:

pszCoordSys Mapinfo style CoordSys definition string.

Returns:

OGRERR_NONE on success, OGRERR_FAILURE on failure, OGRERR_UNSUPPORTED_OPERATION if MITAB library was not linked in.

16.29.3.39 OGRErr OGRSpatialReference::importFromPanorama (long *iProjSys*, long *iDatum*, long *iEllips*, double * *padfPrjParams*)

Import coordinate system from "Panorama" GIS projection definition.

This method will import projection definition in style, used by "Panorama" GIS.

This function is the equivalent of the C function OSRImportFromPanorama().

Parameters:

iProjSys Input projection system code, used in GIS "Panorama".

Supported Projections

```
1:  Gauss-Kruger (Transverse Mercator)
2:  Lambert Conformal Conic 2SP
5:  Stereographic
6:  Azimuthal Equidistant (Postel)
8:  Mercator
10: Polyconic
13: Polar Stereographic
15: Gnomonic
17: Universal Transverse Mercator (UTM)
18: Wagner I (Kavraisky VI)
19: Mollweide
20: Equidistant Conic
24: Lambert Azimuthal Equal Area
27: Equirectangular
28: Cylindrical Equal Area (Lambert)
29: International Map of the World Polyconic
```

Parameters:

iDatum Input coordinate system.

Supported Datums

```
1: Pulkovo, 1942
2: WGS, 1984
```

Parameters:

iEllips Input spheroid.

Supported Spheroids

```
1: Krassovsky, 1940
2: WGS, 1972
3: International, 1924 (Hayford, 1909)
4: Clarke, 1880
5: Clarke, 1866 (NAD1927)
6: Everest, 1830
7: Bessel, 1841
```

```
8: Airy, 1830
9: WGS, 1984 (GPS)
```

Parameters:

padfPrjParams Array of 7 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
```

Particular projection uses different parameters, unused ones may be set to zero. If NULL supplied instead of array pointer default values will be used (i.e., zeroes).

Returns:

OGRERR_NONE on success or an error code in case of failure.

References Clear(), CopyGeogCSFrom(), FixupOrdering(), importFromEPSG(), IsLocal(), IsProjected(), SetAE(), SetAuthority(), SetCEA(), SetEC(), SetEquirectangular(), SetGeogCS(), SetGnomonic(), SetIWMPolyconic(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMercator(), SetMollweide(), SetPolyconic(), SetPS(), SetStereographic(), SetTM(), SetUTM(), SetWagner(), and SetWellKnownGeogCS().

16.29.3.40 OGRErr OGRSpatialReference::importFromPCI (const char * *pszProj*, const char * *pszUnits* = NULL, double * *padfPrjParams* = NULL)

Import coordinate system from PCI projection definition.

PCI software uses 16-character string to specify coordinate system and datum/ellipsoid. You should supply at least this string to the **importFromPCI()** (p. 244) function.

This function is the equilvelent of the C function OSRImportFromPCI().

Parameters:

pszProj NULL terminated string containing the definition. Looks like "ppppppppppp Ennn" or "ppppppppppp Dnnn", where "ppppppppppp" is a projection code, "Ennn" is an ellipsoid code, "Dnnn" — a datum code.

pszUnits Grid units code ("DEGREE" or "METRE"). If NULL "METRE" will be used.

padfPrjParams Array of 16 coordinate system parameters:

[0] Spheroid semi major axis [1] Spheroid semi minor axis [2] Reference Longitude [3] Reference Latitude [4] First Standard Parallel [5] Second Standard Parallel [6] False Easting [7] False Northing [8] Scale Factor [9] Height above sphere surface [10] Longitude of 1st point on center line [11] Latitude of 1st point on center line [12] Longitude of 2nd point on center line [13] Latitude of 2nd point on center line [14] Azimuth east of north for center line [15] Landsat satellite number [16] Landsat path number

Particular projection uses different parameters, unused ones may be set to zero. If NULL suppliet instead of array pointer default values will be used (i.e., zeroes).

Returns:

OGRERR_NONE on success or an error code in case of failure.

References Clear(), CopyGeogCSFrom(), FixupOrdering(), importFromEPSG(), IsGeographic(), IsLocal(), IsProjected(), SetACEA(), SetAE(), SetAngularUnits(), SetAuthority(), SetEC(), SetEquirectangular2(), SetGeogCS(), SetGnomonic(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMC(), SetMercator(), SetOrthographic(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStatePlane(), SetStereographic(), SetTM(), SetUTM(), SetVDG(), and SetWellKnownGeogCS().

16.29.3.41 OGRErr OGRSpatialReference::importFromProj4 (const char * *pszProj4*)

Import PROJ.4 coordinate string.

The **OGRSpatialReference** (p. 224) is initialized from the passed PROJ.4 style coordinate system string. In addition to many +proj formulations which have OGC equivelents, it is also possible to import "+init=epsg:n" style definitions. These are passed to **importFromEPSG()** (p. 240). Other init strings (such as the state plane zones) are not currently supported.

Example: pszProj4 = "+proj=utm +zone=11 +datum=WGS84"

Some parameters, such as grids, recognised by PROJ.4 may not be well understood and translated into the **OGRSpatialReference** (p. 224) model. It is possible to add the +wktext parameter which is a special keyword that OGR recognises as meaning "embed the entire PROJ.4 string in the WKT and use it literally when converting back to PROJ.4 format".

For example: "+proj=nzmg +lat_0=-41 +lon_0=173 +x_0=2510000 +y_0=6023150 +ellps=intl +units=m +nadgrids=nzgd2kgrid0005.gsb +wktext"

will be translated as :

```
PROJCS["unnamed",
   GEOGCS["International 1909 (Hayford)",
       DATUM["unknown",
           SPHEROID["intl",6378388,297]],
       PRIMEM["Greenwich",0],
       UNIT["degree",0.0174532925199433]],
   PROJECTION["New_Zealand_Map_Grid"],
   PARAMETER["latitude_of_origin",-41],
   PARAMETER["central_meridian",173],
   PARAMETER["false_easting",2510000],
   PARAMETER["false_northing",6023150],
   UNIT["Meter",1],
   EXTENSION["PROJ4","+proj=nzmg +lat_0=-41 +lon_0=173 +x_0=2510000
              +y_0=6023150 +ellps=intl  +units=m +nadgrids=nzgd2kgrid0005.gsb +wktext"]]
```

This method is the equivalent of the C function OSRImportFromProj4().

Parameters:

pszProj4 the PROJ.4 style string.

Returns:

OGRERR_NONE on success or OGRERR_CORRUPT_DATA on failure.

References Clear(), CopyGeogCSFrom(), CPLAtof(), CPLAtofM(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), OGR_SRSNode::GetValue(), importFromEPSG(), IsLocal(), IsProjected(), SetACEA(), SetAE(), SetBonne(),

SetCEA(), SetCS(), SetEC(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGeogCS(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLinearUnits(), SetMC(), SetMercator(), SetMollweide(), SetNormProjParm(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStereographic(), SetTM(), SetTOWGS84(), SetTPED(), SetUTM(), SetVDG(), SetWagner(), and SetWellKnownGeogCS().

Referenced by importFromEPSGA(), and SetFromUserInput().

16.29.3.42 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() function.

Parameters:

pszDefinition text definition to try to deduce SRS from.

Returns:

OGRERR_NONE on success, or an error code with the curl error message if it is unable to dowload data.

References SetFromUserInput().

Referenced by SetFromUserInput().

16.29.3.43 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. 256) which can normally be used for URNs.

Parameters:

pszURN the urn string.

Returns:

OGRERR_NONE on success or an error code.

References importFromDict(), importFromEPSGA(), and SetWellKnownGeogCS().

Referenced by SetFromUserInput().

16.29.3.44 OGRErr OGRSpatialReference::importFromUSGS (long *iProjSys*, long *iZone*, double * *padfPrjParams*, long *iDatum*, int *bAnglesInPackedDMSFormat* = 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. 280) 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(). Note that the bAnglesInPackedDMSFormat parameter is only present in the C++ method. The C function assumes bAnglesInPackedFormat = TRUE.

Parameters:

iProjSys Input projection system code, used in GCTP.

iZone Input zone for UTM and State Plane projection systems. For Southern Hemisphere UTM use a negative zone code. iZone ignored for all other projections.

padfPrjParams Array of 15 coordinate system parameters. These parameters differs for different projections.

Projection Transformation Package Projection Parameters

Code & Projection Id	Array Element							
	0	1	2	3	4	5	6	7
0 Geographic								
1 U T M	Lon/Z	Lat/Z						
2 State Plane								
3 Albers Equal Area	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
4 Lambert Conformal C	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
5 Mercator	SMajor	SMinor			CentMer	TrueScale	FE	FN
6 Polar Stereographic	SMajor	SMinor			LongPol	TrueScale	FE	FN
7 Polyconic	SMajor	SMinor			CentMer	OriginLat	FE	FN
8 Equid. Conic A	SMajor	SMinor	STDPAR		CentMer	OriginLat	FE	FN
Equid. Conic B	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN
9 Transverse Mercator	SMajor	SMinor	Factor		CentMer	OriginLat	FE	FN
10 Stereographic	Sphere				CentLon	CenterLat	FE	FN
11 Lambert Azimuthal	Sphere				CentLon	CenterLat	FE	FN
12 Azimuthal	Sphere				CentLon	CenterLat	FE	FN
13 Gnomonic	Sphere				CentLon	CenterLat	FE	FN
14 Orthographic	Sphere				CentLon	CenterLat	FE	FN
15 Gen. Vert. Near Per	Sphere		Height		CentLon	CenterLat	FE	FN
16 Sinusoidal	Sphere				CentMer		FE	FN
17 Equirectangular	Sphere				CentMer	TrueScale	FE	FN
18 Miller Cylindrical	Sphere				CentMer		FE	FN
19 Van der Grinten	Sphere				CentMer	OriginLat	FE	FN
20 Hotin Oblique Merc A	SMajor	SMinor	Factor			OriginLat	FE	FN
Hotin Oblique Merc B	SMajor	SMinor	Factor	AziAng	AzmthPt	OriginLat	FE	FN
21 Robinson	Sphere				CentMer		FE	FN
22 Space Oblique Merc A	SMajor	SMinor		IncAng	AscLong		FE	FN
Space Oblique Merc B	SMajor	SMinor	Satnum	Path			FE	FN
23 Alaska Conformal	SMajor	SMinor					FE	FN
24 Interrupted Goode	Sphere							
25 Mollweide	Sphere				CentMer		FE	FN
26 Interrupt Mollweide	Sphere							
27 Hammer	Sphere				CentMer		FE	FN
28 Wagner IV	Sphere				CentMer		FE	FN
29 Wagner VII	Sphere				CentMer		FE	FN

```
30 Oblated Equal Area   |Sphere|      |Shapem|Shapen|CentLon|CenterLat|FE|FN
-------------------------------------------------------------------------------
```

Code & Projection Id	Array Element 8	9	10	11	12
0 Geographic					
1 U T M					
2 State Plane					
3 Albers Equal Area					
4 Lambert Conformal C					
5 Mercator					
6 Polar Stereographic					
7 Polyconic					
8 Equid. Conic A	zero				
Equid. Conic B	one				
9 Transverse Mercator					
10 Stereographic					
11 Lambert Azimuthal					
12 Azimuthal					
13 Gnomonic					
14 Orthographic					
15 Gen. Vert. Near Per					
16 Sinusoidal					
17 Equirectangular					
18 Miller Cylindrical					
19 Van der Grinten					
20 Hotin Oblique Merc A	Long1	Lat1	Long2	Lat2	zero
Hotin Oblique Merc B					one
21 Robinson					
22 Space Oblique Merc A	PSRev	LRat	PFlag		zero
Space Oblique Merc B					one
23 Alaska Conformal					
24 Interrupted Goode					
25 Mollweide					
26 Interrupt Mollweide					
27 Hammer					
28 Wagner IV					
29 Wagner VII					
30 Oblated Equal Area	Angle				

where

Lon/Z	Longitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
Lat/Z	Latitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
SMajor	Semi-major axis of ellipsoid. If zero, Clarke 1866 in meters is assumed.
SMinor	Eccentricity squared of the ellipsoid if less than zero, if zero, a spherical form is assumed, or if greater than zero, the semi-minor axis of ellipsoid.
Sphere	Radius of reference sphere. If zero, 6370997 meters is used.
STDPAR	Latitude of the standard parallel
STDPR1	Latitude of the first standard parallel

```
   STDPR2    Latitude of the second standard parallel
   CentMer   Longitude of the central meridian
   OriginLat Latitude of the projection origin
   FE        False easting in the same units as the semi-major axis
   FN        False northing in the same units as the semi-major axis
   TrueScale Latitude of true scale
   LongPol   Longitude down below pole of map
   Factor    Scale factor at central meridian (Transverse Mercator) or
             center of projection (Hotine Oblique Mercator)
   CentLon   Longitude of center of projection
   CenterLat Latitude of center of projection
   Height    Height of perspective point
   Long1     Longitude of first point on center line (Hotine Oblique
             Mercator, format A)
   Long2     Longitude of second point on center line (Hotine Oblique
             Mercator, format A)
   Lat1      Latitude of first point on center line (Hotine Oblique
             Mercator, format A)
   Lat2      Latitude of second point on center line (Hotine Oblique
             Mercator, format A)
   AziAng    Azimuth angle east of north of center line (Hotine Oblique
             Mercator, format B)
   AzmthPt   Longitude of point on central meridian where azimuth occurs
             (Hotine Oblique Mercator, format B)
   IncAng    Inclination of orbit at ascending node, counter-clockwise
             from equator (SOM, format A)
   AscLong   Longitude of ascending orbit at equator (SOM, format A)
   PSRev     Period of satellite revolution in minutes (SOM, format A)
   LRat      Landsat ratio to compensate for confusion at northern end
             of orbit (SOM, format A -- use 0.5201613)
   PFlag     End of path flag for Landsat:  0 = start of path,
             1 = end of path (SOM, format A)
   Satnum    Landsat Satellite Number (SOM, format B)
   Path      Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and
             WRS-2 for Landsat 4, 5 and 6.)  (SOM, format B)
   Shapem    Oblated Equal Area oval shape parameter m
   Shapen    Oblated Equal Area oval shape parameter n
   Angle     Oblated Equal Area oval rotation angle

Array elements 13 and 14 are set to zero. All array elements with blank
fields are set to zero too.
```

Parameters:

iDatum Input spheroid.

If the datum code is negative, the first two values in the parameter array (parm) are used to define the values as follows:

- If padfPrjParams[0] is a non-zero value and padfPrjParams[1] is greater than one, the semimajor axis is set to padfPrjParams[0] and the semiminor axis is set to padfPrjParams[1].
- If padfPrjParams[0] is nonzero and padfPrjParams[1] is greater than zero but less than or equal to one, the semimajor axis is set to padfPrjParams[0] and the semiminor axis is computed from the eccentricity squared value padfPrjParams[1]:

  semiminor = sqrt(1.0 - ES) $*$ semimajor

where

ES = eccentricity squared

- If padfPrjParams[0] is nonzero and padfPrjParams[1] is equal to zero, the semimajor axis and semiminor axis are set to padfPrjParams[0].
- If padfPrjParams[0] equals zero and padfPrjParams[1] is greater than zero, the default Clarke 1866 is used to assign values to the semimajor axis and semiminor axis.
- If padfPrjParams[0] and padfPrjParams[1] equals zero, the semimajor axis is set to 6370997.0 and the semiminor axis is set to zero.

If a datum code is zero or greater, the semimajor and semiminor axis are defined by the datum code as found in the following table:

Supported Datums

```
 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
```

Returns:

OGRERR_NONE on success or an error code in case of failure.

References FixupOrdering(), IsLocal(), IsProjected(), SetACEA(), SetAE(), SetAuthority(), SetEC(), SetEquirectangular2(), SetGeogCS(), SetGnomonic(), SetHOM(), SetHOM2PNO(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMC(), SetMercator(), SetMollweide(), SetOrthographic(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStatePlane(), SetStereographic(), SetTM(), SetUTM(), SetVDG(), SetWagner(), and SetWellKnownGeogCS().

16.29.3.45 OGRErr OGRSpatialReference::importFromWkt (char ** *ppszInput*)

Import from WKT string.

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

This method is the same as the C function **OSRImportFromWkt()** (p. 421).

Parameters:

ppszInput Pointer to pointer to input. The pointer is updated to point to remaining unused input text.

Returns:

OGRERR_NONE if import succeeds, or OGRERR_CORRUPT_DATA if it fails for any reason.

References Clear(), and OGR_SRSNode::importFromWkt().

Referenced by importFromDict(), importFromESRI(), OGRSpatialReference(), SetFromUserInput(), and SetWellKnownGeogCS().

16.29.3.46 int OGRSpatialReference::IsGeographic () const

Check if geographic coordinate system.

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

Returns:

TRUE if this spatial reference is geographic ... that is the root is a GEOGCS node.

Referenced by AutoIdentifyEPSG(), EPSGTreatsAsLatLong(), exportToERM(), exportToProj4(), importFromEPSGA(), importFromPCI(), and SetWellKnownGeogCS().

16.29.3.47 int OGRSpatialReference::IsLocal () const

Check if local coordinate system.

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

Returns:

TRUE if this spatial reference is local ... that is the root is a LOCAL_CS node.

Referenced by exportToPanorama(), exportToPCI(), exportToUSGS(), importFromERM(), importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.48 int OGRSpatialReference::IsProjected () const

Check if projected coordinate system.

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

Returns:

TRUE if this contains a PROJCS node indicating a it is a projected coordinate system.

References OGR_SRSNode::GetValue().

Referenced by AutoIdentifyEPSG(), exportToERM(), importFromEPSGA(), importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and IsSame().

16.29.3.49 int OGRSpatialReference::IsSame (const OGRSpatialReference * *poOtherSRS*) const

These two spatial references describe the same system.

Parameters:

poOtherSRS the SRS being compared to.

Returns:

TRUE if equivelent or FALSE otherwise.

References GetAttrNode(), GetAttrValue(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), GetRoot(), OGR_SRSNode::GetValue(), IsProjected(), and IsSameGeogCS().

16.29.3.50 int OGRSpatialReference::IsSameGeogCS (const OGRSpatialReference * *poOther*) const

Do the GeogCS'es match?

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

Parameters:

poOther the SRS being compared against.

Returns:

TRUE if they are the same or FALSE otherwise.

References CPLAtof(), and GetAttrValue().

Referenced by IsSame().

16.29.3.51 OGRErr OGRSpatialReference::morphFromESRI ()

Convert in place to ESRI WKT format.

The value nodes of this coordinate system as modified in various manners more closely map onto the ESRI concept of WKT format. This includes renaming a variety of projections and arguments, and stripping out nodes note recognised by ESRI (like AUTHORITY and AXIS).

This does the same as the C function OSRMorphFromESRI().

Returns:

OGRERR_NONE unless something goes badly wrong.

References OGR_SRSNode::applyRemapper(), FixupOrdering(), GetAttrNode(), GetAttrValue(), OGR_SRSNode::GetChild(), GetProjParm(), OGR_SRSNode::GetValue(), SetNode(), SetProjParm(), and OGR_SRSNode::SetValue().

Referenced by importFromESRI(), and SetFromUserInput().

16.29.3.52 OGRErr OGRSpatialReference::morphToESRI ()

Convert in place from ESRI WKT format.

The value notes of this coordinate system as 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.

This does the same as the C function OSRMorphToESRI().

Returns:

OGRERR_NONE unless something goes badly wrong.

References OGR_SRSNode::applyRemapper(), OGR_SRSNode::DestroyChild(), Fixup(), GetAngularUnits(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), GetNormProjParm(), GetProjParm(), GetUTMZone(), OGR_SRSNode::GetValue(), SetNode(), OGR_SRSNode::SetValue(), and StripCTParms().

16.29.3.53 int OGRSpatialReference::Reference ()

Increments the reference count by one.

The reference count is used keep track of the number of **OGRGeometry** (p. 127) objects referencing this SRS.

The method does the same thing as the C function OSRReference().

Returns:

the updated reference count.

Referenced by OGRGeometry::assignSpatialReference().

16.29.3.54 void OGRSpatialReference::Release ()

Decrements the reference count by one, and destroy if zero.

The method does the same thing as the C function OSRRelease().

References Dereference().

Referenced by OGRGeometry::assignSpatialReference().

16.29.3.55 OGRErr OGRSpatialReference::SetACEA (double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Albers Conic Equal Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.56 OGRErr OGRSpatialReference::SetAE (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Azimuthal Equidistant

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.57 OGRErr OGRSpatialReference::SetAngularUnits (const char * *pszUnitsName*, double *dfInRadians*)

Set the angular units for the geographic coordinate system.

This method creates a UNITS subnode with the specified values as a child of the GEOGCS node.

This method does the same as the C function OSRSetAngularUnits().

Parameters:

pszUnitsName the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. 418) such as SRS_UA_DEGREE.

dfInRadians the value to multiple by an angle in the indicated units to transform to radians. Some standard conversion factors can be found in **ogr_srs_api.h** (p. 418).

Returns:

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::GetChild(), and OGR_SRSNode::SetValue().

Referenced by Fixup(), and importFromPCI().

16.29.3.58 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().

Parameters:

pszTargetKey the partial or complete path to the node to set an authority on. ie. "PROJCS", "GEOGCS" or "GEOGCS|UNIT".

pszAuthority authority name, such as "EPSG".

nCode code for value with this authority.

Returns:

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), and GetAttrNode().

Referenced by AutoIdentifyEPSG(), importFromEPSGA(), importFromPanorama(), importFromPCI(), and importFromUSGS().

16.29.3.59 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 equivelent to the C function OSRSetAxes().

Parameters:

pszTargetKey either "PROJCS" or "GEOGCS", must already exist in SRS.

pszXAxisName name of first axis, normally "Long" or "Easting".

eXAxisOrientation normally OAO_East.

pszYAxisName name of second axis, normally "Lat" or "Northing".

eYAxisOrientation normally OAO_North.

Returns:

OGRERR_NONE on success or an error code.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), and OGR_SRSNode::FindChild().

16.29.3.60 OGRErr OGRSpatialReference::SetBonne (double *dfStdP1*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Bonne

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.61 OGRErr OGRSpatialReference::SetCEA (double *dfStdP1*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cylindrical Equal Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), and importFromProj4().

16.29.3.62 OGRErr OGRSpatialReference::SetCS (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cassini-Soldner

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.63 OGRErr OGRSpatialReference::SetEC (double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equidistant Conic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.64 OGRErr OGRSpatialReference::SetEckert (int *nVariation*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert I-VI

References SetNormProjParm(), and SetProjection().

16.29.3.65 OGRErr OGRSpatialReference::SetEquirectangular (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equirectangular

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), and importFromProj4().

16.29.3.66 OGRErr OGRSpatialReference::SetEquirectangular2 (double *dfCenterLat*, double *dfCenterLong*, double *dfPseudoStdParallel1*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equirectangular generalized form :

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.67 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. 250).
2. "EPSG:n" - number passed on to **importFromEPSG()** (p. 240).
3. "EPSGA:n" - number passed on to **importFromEPSGA()** (p. 241).
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. 245).
7. filename - file read for WKT, XML or PROJ.4 definition.

8. well known name accepted by **SetWellKnownGeogCS()** (p. 269), 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. 252).

It is expected that this method will be extended in the future to support XML and perhaps a simplified "minilanguage" for indicating common UTM and State Plane definitions.

This method is intended to be flexible, but by it's nature it is imprecise as it must guess information about the format intended. When possible applications should call the specific method appropriate if the input is known to be in a particular format.

This method does the same thing as the OSRSetFromUserInput() function.

Parameters:

pszDefinition text definition to try to deduce SRS from.

Returns:

OGRERR_NONE on success, or an error code if the name isn't recognised, the definition is corrupt, or an EPSG value can't be successfully looked up.

References Clear(), importFromDict(), importFromEPSG(), importFromEPSGA(), importFromProj4(), importFromUrl(), importFromURN(), importFromWkt(), morphFromESRI(), and SetWellKnownGeogCS().

Referenced by importFromUrl().

16.29.3.68 OGRErr OGRSpatialReference::SetGaussSchreiberTMercator (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gauss Schreiber Transverse Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.69 OGRErr OGRSpatialReference::SetGeogCS (const char * *pszGeogName*, const char * *pszDatumName*, const char * *pszSpheroidName*, double *dfSemiMajor*, double *dfInvFlattening*, const char * *pszPMName* = `NULL`, double *dfPMOffset* = `0.0`, const char * *pszAngularUnits* = `NULL`, double *dfConvertToRadians* = `0.0`)

Set geographic coordinate system.

This method is used to set the datum, ellipsoid, prime meridian and angular units for a geographic coordinate system. It can be used on it's own to establish a geographic spatial reference, or applied to a projected coordinate system to establish the underlying geographic coordinate system.

This method does the same as the C function OSRSetGeogCS().

Parameters:

pszGeogName user visible name for the geographic coordinate system (not to serve as a key).

pszDatumName key name for this datum. The OpenGIS specification lists some known values, and otherwise EPSG datum names with a standard transformation are considered legal keys.

pszSpheroidName user visible spheroid name (not to serve as a key)

dfSemiMajor the semi major axis of the spheroid.

dfInvFlattening the inverse flattening for the spheroid. This can be computed from the semi minor axis as 1/f = 1.0 / (1.0 - semiminor/semimajor).

pszPMName the name of the prime merdidian (not to serve as a key) If this is NULL a default value of "Greenwich" will be used.

dfPMOffset the longitude of greenwich relative to this prime meridian.

pszAngularUnits the angular units name (see **ogr_srs_api.h** (p. 418) for some standard names). If NULL a value of "degrees" will be assumed.

dfConvertToRadians value to multiply angular units by to transform them to radians. A value of SRS_UL_DEGREE_CONV will be used if pszAngularUnits is NULL.

Returns:

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), Clear(), CPLAtof(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::InsertChild(), and SetRoot().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.70 OGRErr OGRSpatialReference::SetGEOS (double *dfCentralMeridian*, double *dfSatelliteHeight*, double *dfFalseEasting*, double *dfFalseNorthing*)

Geostationary Satellite

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.71 OGRErr OGRSpatialReference::SetGH (double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Goode Homolosine

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.72 OGRErr OGRSpatialReference::SetGnomonic (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gnomonic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.73 OGRErr OGRSpatialReference::SetGS (double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gall Stereograpic

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.74 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.

This method does the same thing as the C function **OSRSetHOM()** (p. 424).

Parameters:

dfCenterLat Latitude of the projection origin.
dfCenterLong Longitude of the projection origin.
dfAzimuth Azimuth, measured clockwise from North, of the projection centerline.
dfRectToSkew ?.
dfScale Scale factor applies to the projection origin.
dfFalseEasting False easting.
dfFalseNorthing False northing.

Returns:

OGRERR_NONE on success.

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4(), and importFromUSGS().

16.29.3.75 OGRErr OGRSpatialReference::SetHOM2PNO (double *dfCenterLat*, double *dfLat1*, double *dfLong1*, double *dfLat2*, double *dfLong2*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Set a Hotine Oblique Mercator projection using two points on projection centerline.

This method does the same thing as the C function **OSRSetHOM2PNO()** (p. 424).

Parameters:

dfCenterLat Latitude of the projection origin.
dfLat1 Latitude of the first point on center line.
dfLong1 Longitude of the first point on center line.
dfLat2 Latitude of the second point on center line.
dfLong2 Longitude of the second point on center line.
dfScale Scale factor applies to the projection origin.
dfFalseEasting False easting.
dfFalseNorthing False northing.

Returns:

OGRERR_NONE on success.

References SetNormProjParm(), and SetProjection().

Referenced by importFromUSGS().

16.29.3.76 OGRErr OGRSpatialReference::SetIWMPolyconic (double *dfLat1*, double *dfLat2*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

International Map of the World Polyconic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), and importFromProj4().

16.29.3.77 OGRErr OGRSpatialReference::SetKrovak (double *dfCenterLat*, double *dfCenterLong*, double *dfAzimuth*, double *dfPseudoStdParallelLat*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Krovak Oblique Conic Conformal

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.78 OGRErr OGRSpatialReference::SetLAEA (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Azimuthal Equal-Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.79 OGRErr OGRSpatialReference::SetLCC (double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.80 OGRErr OGRSpatialReference::SetLCC1SP (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic 1SP

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.81 OGRErr OGRSpatialReference::SetLCCB (double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic (Belgium)

References SetNormProjParm(), and SetProjection().

16.29.3.82 OGRErr OGRSpatialReference::SetLinearUnits (const char * *pszUnitsName*, double *dfInMeters*)

Set the linear units for the projection.

This method creates a UNITS subnode with the specified values as a child of the PROJCS or LOCAL_CS node.

This method does the same as the C function OSRSetLinearUnits().

Parameters:

pszUnitsName the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. 418) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.

dfInMeters the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in **ogr_srs_api.h** (p. 418).

Returns:

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::GetChild(), and OGR_SRSNode::SetValue().

Referenced by Fixup(), importFromERM(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), SetLinearUnitsAndUpdateParameters(), SetStatePlane(), and SetUTM().

16.29.3.83 OGRErr OGRSpatialReference::SetLinearUnitsAndUpdateParameters (const char * *pszName*, double *dfInMeters*)

Set the linear units for the projection.

This method creates a UNITS subnode with the specified values as a child of the PROJCS or LOCAL_CS node. It works the same as the **SetLinearUnits()** (p. 261) method, but it also updates all existing linear projection parameter values from the old units to the new units.

Parameters:

pszUnitsName the units name to be used. Some preferred units names can be found in **ogr_srs_api.h** (p. 418) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.

dfInMeters the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in **ogr_srs_api.h** (p. 418).

Returns:

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), OGR_SRSNode::GetValue(), SetLinearUnits(), and SetProjParm().

Referenced by importFromESRI().

16.29.3.84 OGRErr OGRSpatialReference::SetLocalCS (const char * *pszName*)

Set the user visible LOCAL_CS name.

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

This method is 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. 261).

Parameters:

pszName the user visible name to assign. Not used as a key.

Returns:

OGRERR_NONE on success.

References GetAttrNode(), and SetNode().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromUSGS(), and SetStatePlane().

16.29.3.85 OGRErr OGRSpatialReference::SetMC (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Miller Cylindrical

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.86 OGRErr OGRSpatialReference::SetMercator (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.87 OGRErr OGRSpatialReference::SetMollweide (double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Mollweide

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), and importFromUSGS().

16.29.3.88 OGRErr OGRSpatialReference::SetNode (const char * *pszNodePath*, const char * *pszNewNodeValue*)

Set attribute value in spatial reference.

Missing intermediate nodes in the path will be created if not already in existance. If the attribute has no children one will be created and assigned the value otherwise the zeroth child will be assigned the value.

This method does the same as the C function OSRSetAttrValue().

Parameters:

pszNodePath full path to attribute to be set. For instance "PROJCS|GEOGCS|UNITS".

pszNewNodeValue value to be assigned to node, such as "meter". This may be NULL if you just want to force creation of the intermediate path.

Returns:

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetValue(), SetRoot(), and OGR_SRSNode::SetValue().

Referenced by morphFromESRI(), morphToESRI(), SetLocalCS(), SetProjCS(), SetProjection(), and SetUTM().

16.29.3.89 OGRErr OGRSpatialReference::SetNormProjParm (const char * *pszName*, double *dfValue*)

Set a projection parameter with a normalized value.

This method is the same as **SetProjParm()** (p. 265) 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().

Parameters:

pszName the parameter name, which should be selected from the macros in **ogr_srs_api.h** (p. 418), such as SRS_PP_CENTRAL_MERIDIAN.

dfValue value to assign.

Returns:

OGRERR_NONE on success.

References SetProjParm().

Referenced by importFromProj4(), SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetSOC(), SetStatePlane(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetUTM(), SetVDG(), and SetWagner().

16.29.3.90 OGRErr OGRSpatialReference::SetNZMG (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

New Zealand Map Grid

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.91 OGRErr OGRSpatialReference::SetOrthographic (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Orthographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.92 OGRErr OGRSpatialReference::SetOS (double *dfOriginLat*, double *dfCMeridian*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Oblique Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.93 OGRErr OGRSpatialReference::SetPolyconic (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Polyconic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.94 OGRErr OGRSpatialReference::SetProjCS (const char * *pszName*)

Set the user visible PROJCS name.

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

This method is will ensure a PROJCS node is created as the root, and set the provided name on it. If used on a GEOGCS coordinate system, the GEOGCS node will be demoted to be a child of the new PROJCS root.

Parameters:

pszName the user visible name to assign. Not used as a key.

Returns:

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::GetValue(), OGR_SRSNode::InsertChild(), and SetNode().

16.29.3.95 OGRErr OGRSpatialReference::SetProjection (const char * *pszProjection*)

Set a projection name.

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

Parameters:

pszProjection the projection name, which should be selected from the macros in **ogr_srs_api.h** (p. 418), such as SRS_PT_TRANSVERSE_MERCATOR.

Returns:

OGRERR_NONE on success.

References GetAttrNode(), OGR_SRSNode::GetValue(), OGR_SRSNode::InsertChild(), and SetNode().

Referenced by SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetSOC(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetUTM(), SetVDG(), and SetWagner().

16.29.3.96 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().

Please check `http://www.remotesensing.org/geotiff/proj_list` pages for legal parameter names for specific projections.

Parameters:

pszParmName the parameter name, which should be selected from the macros in **ogr_srs_api.h** (p. 418), such as SRS_PP_CENTRAL_MERIDIAN.

dfValue value to assign.

Returns:

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), GetAttrNode(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetValue(), and OGR_SRSNode::SetValue().

Referenced by morphFromESRI(), SetLinearUnitsAndUpdateParameters(), and SetNormProjParm().

16.29.3.97 OGRErr OGRSpatialReference::SetPS (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Polar Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.98 OGRErr OGRSpatialReference::SetRobinson (double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Robinson

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.99 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. 81) is is assumed by the **OGRSpatialReference** (p. 224).

Parameters:

poNewRoot object to assign as root.

Referenced by CopyGeogCSFrom(), SetGeogCS(), and SetNode().

16.29.3.100 OGRErr OGRSpatialReference::SetSinusoidal (double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Sinusoidal

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.101 OGRErr OGRSpatialReference::SetSOC (double *dfLatitudeOfOrigin*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Swiss Oblique Cylindrical

References SetNormProjParm(), and SetProjection().

16.29.3.102 OGRErr OGRSpatialReference::SetStatePlane (int *nZone*, int *bNAD83* = `TRUE`, const char * *pszOverrideUnitName* = `NULL`, double *dfOverrideUnit* = `0.0`)

State Plane

Set State Plane projection definition.

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().

Parameters:

nZone State plane zone number, in the USGS numbering scheme (as dinstinct from the Arc/Info and Erdas numbering scheme.

bNAD83 TRUE if the NAD83 zone definition should be used or FALSE if the NAD27 zone definition should be used.

pszOverrideUnitName Linear unit name to apply overriding the legal definition for this zone.

dfOverrideUnit Linear unit conversion factor to apply overriding the legal definition for this zone.

Returns:

OGRERR_NONE on success, or OGRERR_FAILURE on failure, mostly likely due to the EPSG tables not being accessable.

References Clear(), CPLAtof(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), GetLinearUnits(), GetNormProjParm(), importFromEPSG(), SetLinearUnits(), SetLocalCS(), and SetNormProjParm().

Referenced by importFromESRI(), importFromPCI(), and importFromUSGS().

16.29.3.103 OGRErr OGRSpatialReference::SetStereographic (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.104 OGRErr OGRSpatialReference::SetTM (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.105 OGRErr OGRSpatialReference::SetTMG (double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Tunesia Mining Grid

References SetNormProjParm(), and SetProjection().

16.29.3.106 OGRErr OGRSpatialReference::SetTMSO (double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator (South Oriented)

References SetNormProjParm(), and SetProjection().

16.29.3.107 OGRErr OGRSpatialReference::SetTMVariant (const char * *pszVariantName*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator variants.

References SetNormProjParm(), and SetProjection().

16.29.3.108 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. 224) 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().

Parameters:

dfDX X child in meters.

dfDY Y child in meters.

dfDZ Z child in meters.

dfEX X rotation in arc seconds (optional, defaults to zero).

dfEY Y rotation in arc seconds (optional, defaults to zero).

dfEZ Z rotation in arc seconds (optional, defaults to zero).

dfPPM scaling factor (parts per million).

Returns:

OGRERR_NONE on success.

References OGR_SRSNode::AddChild(), OGR_SRSNode::DestroyChild(), OGR_SRSNode::FindChild(), GetAttrNode(), OGR_SRSNode::GetChildCount(), and OGR_SRSNode::InsertChild().

Referenced by importFromProj4().

16.29.3.109 OGRErr OGRSpatialReference::SetTPED (double *dfLat1*, double *dfLong1*, double *dfLat2*, double *dfLong2*, double *dfFalseEasting*, double *dfFalseNorthing*)

Two Point Equidistant

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

16.29.3.110 OGRErr OGRSpatialReference::SetUTM (int *nZone*, int *bNorth* = `TRUE`)

Universal Transverse Mercator

Set UTM projection definition.

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().

Parameters:

nZone UTM zone.

bNorth TRUE for northern hemisphere, or FALSE for southern hemisphere.

Returns:

OGRERR_NONE on success.

References GetAttrValue(), SetLinearUnits(), SetNode(), SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.111 OGRErr OGRSpatialReference::SetVDG (double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

VanDerGrinten

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

16.29.3.112 OGRErr OGRSpatialReference::SetWagner (int *nVariation*, double *dfCenterLat*, double *dfFalseEasting*, double *dfFalseNorthing*)

Wagner I – VII

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), and importFromUSGS().

16.29.3.113 OGRErr OGRSpatialReference::SetWellKnownGeogCS (const char * *pszName*)

Set a GeogCS based on well known name.

This may be called on an empty **OGRSpatialReference** (p. 224) to make a geographic coordinate system, or on something with an existing PROJCS node to set the underlying geographic coordinate system of a projected coordinate system.

The following well known text values are currently supported:

- "WGS84": same as "EPSG:4326" but has no dependence on EPSG data files.
- "WGS72": same as "EPSG:4322" but has no dependence on EPSG data files.
- "NAD27": same as "EPSG:4267" but has no dependence on EPSG data files.
- "NAD83": same as "EPSG:4269" but has no dependence on EPSG data files.
- "EPSG:n": same as doing an ImportFromEPSG(n).

Parameters:

pszName name of well known geographic coordinate system.

Returns:

OGRERR_NONE on success, or OGRERR_FAILURE if the name isn't recognised, the target object is already initialized, or an EPSG value can't be successfully looked up.

References CopyGeogCSFrom(), importFromEPSG(), importFromEPSGA(), importFromWkt(), and IsGeographic().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromURN(), importFromUSGS(), and SetFromUserInput().

16.29.3.114 OGRErr OGRSpatialReference::StripCTParms (OGR_SRSNode * *poCurrent* = NULL)

Strip OGC CT Parameters.

This method will remove all components of the coordinate system that are specific to the OGC CT Specification. That is it will attempt to strip it down to being compatible with the Simple Features 1.0 specification.

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

Parameters:

poCurrent node to operate on. NULL to operate on whole tree.

Returns:

OGRERR_NONE on success or an error code.

References OGR_SRSNode::GetValue(), and OGR_SRSNode::StripNodes().

Referenced by morphToESRI().

16.29.3.115 OGRErr OGRSpatialReference::Validate ()

Validate SRS tokens.

This method attempts to verify that the spatial reference system is well formed, and consists of known tokens. The validation is not comprehensive.

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

Returns:

OGRERR_NONE if all is fine, OGRERR_CORRUPT_DATA if the SRS is not well formed, and OGRERR_UNSUPPORTED_SRS if the SRS is well formed, but contains non-standard PROJECTION[] values.

References CPLAtof(), OGR_SRSNode::GetChild(), OGR_SRSNode::GetChildCount(), OGR_SRSNode::GetNode(), and OGR_SRSNode::GetValue().

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

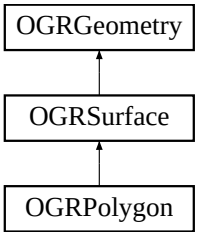
- **ogr_spatialref.h**
- ogr_fromepsg.cpp
- ogr_srs_dict.cpp
- ogr_srs_erm.cpp
- ogr_srs_esri.cpp
- ogr_srs_panorama.cpp
- ogr_srs_pci.cpp
- ogr_srs_proj4.cpp

- ogr_srs_usgs.cpp
- ogr_srs_validate.cpp
- ogr_srs_xml.cpp
- ogrspatialreference.cpp

16.30 OGRSurface Class Reference

`#include <ogr_geometry.h>`

Inheritance diagram for OGRSurface::



Public Member Functions

- virtual double **get_Area** () const =0
- virtual OGRErr **Centroid** (**OGRPoint** ∗poPoint) const =0
- virtual OGRErr **PointOnSurface** (**OGRPoint** ∗poPoint) const =0

16.30.1 Detailed Description

Abstract base class for 2 dimensional objects like polygons.

16.30.2 Member Function Documentation

16.30.2.1 OGRErr OGRSurface::Centroid (OGRPoint ∗ *poPoint*) const `[pure virtual]`

Compute and return centroid of surface. The centroid is not necessarily within the geometry.

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

NOTE: Only implemented when GEOS included in build.

Parameters:

poPoint point to be set with the centroid location.

Returns:

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implemented in **OGRPolygon** (p. 207).

16.30.2.2 double OGRSurface::get_Area () const `[pure virtual]`

Get the area of the surface object.

For polygons the area is computed as the area of the outer ring less the area of all internal rings.

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

Returns:

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

Implemented in **OGRPolygon** (p. 209).

16.30.2.3 OGRErr OGRSurface::PointOnSurface (OGRPoint * *poPoint*) const [pure virtual]

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

NOTE: Only implemented when GEOS included in build.

Parameters:

poPoint point to be set with an internal point.

Returns:

OGRERR_NONE if it succeeds or OGRERR_FAILURE otherwise.

Implemented in **OGRPolygon** (p. 213).

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

- **ogr_geometry.h**
- ogrsurface.cpp

Chapter 17

File Documentation

17.1 cpl_conv.h File Reference

```
#include "cpl_port.h"
#include "cpl_vsi.h"
#include "cpl_error.h"
```

Classes

- struct **CPLSharedFileInfo**
- class **CPLLocaleC**

Functions

- 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** (FILE ∗)
- double **CPLAtof** (const char ∗)
- double **CPLAtofDelim** (const char ∗, char)
- double **CPLStrtod** (const char ∗, char ∗∗)
- double **CPLStrtodDelim** (const char ∗, char ∗∗, char)
- float **CPLStrtof** (const char ∗, char ∗∗)
- float **CPLStrtofDelim** (const char ∗, char ∗∗, char)
- double **CPLAtofM** (const char ∗)
- char ∗ **CPLScanString** (const char ∗, int, int, int)
- double **CPLScanDouble** (const char ∗, int)
- long **CPLScanLong** (const char ∗, int)
- unsigned long **CPLScanULong** (const char ∗, int)
- GUIntBig **CPLScanUIntBig** (const char ∗, int)

- void * **CPLScanPointer** (const char *, int)
- int **CPLPrintString** (char *, const char *, int)
- int **CPLPrintStringFill** (char *, const char *, int)
- int **CPLPrintInt32** (char *, 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 *pszExtension)
- const char * **CPLFormCIFilename** (const char *pszPath, const char *pszBasename, const char *pszExtension)
- const char * **CPLResetExtension** (const char *, const char *)
- const char * **CPLProjectRelativeFilename** (const char *pszProjectDir, const char *pszSecondaryFilename)
- int **CPLIsFilenameRelative** (const char *pszFilename)
- const char * **CPLExtractRelativePath** (const char *, const char *, int *)
- const char * **CPLCleanTrailingSlash** (const char *)
- char ** **CPLCorrespondingPaths** (const char *pszOldFilename, const char *pszNewFilename, char **papszFileList)
- int **CPLCheckForFile** (char *pszFilename, char **papszSiblingList)
- const char * **CPLGenerateTempFilename** (const char *pszStem)
- FILE * **CPLOpenShared** (const char *, const char *, int)
- void **CPLCloseShared** (FILE *)
- CPLSharedFileInfo * **CPLGetSharedList** (int *)
- void **CPLDumpSharedList** (FILE *)
- double **CPLPackedDMSToDec** (double)
- double **CPLDecToPackedDMS** (double dfDec)
- int **CPLUnlinkTree** (const char *)

17.1.1 Detailed Description

Various convenience functions for CPL.

17.1.2 Function Documentation

17.1.2.1 double CPLAtof (const char * *nptr*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. The behaviour is the same as

CPLStrtod(nptr, (char **)NULL);

This function does the same as standard atof(3), but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLAtofDelim()** (p. 277) function if you want to specify custom delimiter.

IMPORTANT NOTE. Existance of this function does not mean you should always use it. Sometimes you should use standard locale aware atof(3) and its family. When you need to process the user's input (for example, command line parameters) use atof(3), because user works in localized environment and her input will be done accordingly the locale set. In particular that means we should not make assumptions about character used as decimal delimiter, it can be either "." or ",". But when you are parsing some ASCII file in predefined format, you most likely need **CPLAtof()** (p. 276), because such files distributed across the systems with different locales and floating point representation shoudl be considered as a part of file format. If the format uses "." as a delimiter the same character must be used when parsing number regardless of actual locale setting.

Parameters:

nptr Pointer to string to convert.

Returns:

Converted value, if any.

References CPLAtof(), and CPLStrtod().

Referenced by CPLAtof(), CPLScanDouble(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::Fixup(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetLinearUnits(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::IsSameGeogCS(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetStatePlane(), and OGRSpatialReference::Validate().

17.1.2.2 double CPLAtofDelim (const char * *nptr*, char *point*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. The behaviour is the same as

CPLStrtodDelim(nptr, (char **)NULL, point);

This function does the same as standard atof(3), but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. 276) function.

Parameters:

nptr Pointer to string to convert.

point Decimal delimiter.

Returns:

Converted value, if any.

References CPLAtofDelim(), and CPLStrtodDelim().

Referenced by CPLAtofDelim().

17.1.2.3 double CPLAtofM (const char * *nptr*)

Converts ASCII string to floating point number using any numeric locale.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. This function does the same as standard atof(), but it allows a variety of locale representations. That is it supports numeric values with either a comma or a period for the decimal delimiter.

PS. The M stands for Multi-lingual.

Parameters:

nptr The string to convert.

Returns:

Converted value, if any. Zero on failure.

References CPLAtofM(), and CPLStrtodDelim().

Referenced by CPLAtofM(), and OGRSpatialReference::importFromProj4().

17.1.2.4 void* CPLCalloc (size_t *nCount*, size_t *nSize*)

Safe version of calloc().

This function is like the C library calloc(), but raises a CE_Fatal error with **CPLError()** (p. 299) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSICalloc() to get the memory, so any hooking of VSICalloc() will apply to **CPLCalloc()** (p. 278) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLCalloc()** (p. 278).

Parameters:

nCount number of objects to allocate.

nSize size (in bytes) of object to allocate.

Returns:

pointer to newly allocated memory, only NULL if nSize * nCount is NULL.

17.1.2.5 int CPLCheckForFile (char * *pszFilename*, char ** *papszSiblingFiles*)

Check for file existance.

The function checks if a named file exists in the filesystem, hopefully in an efficient fashion if a sibling file list is available. It exists primarily to do faster file checking for functions like GDAL open methods that get a list of files from the target directory.

If the sibling file list exists (is not NULL) it is assumed to be a list of files in the same directory as the target file, and it will be checked (case insensitively) for a match. If a match is found, pszFilename is updated with the correct case and TRUE is returned.

If papszSiblingFiles is NULL, a **VSIStatL()** (p. 342) is used to test for the files existance, and no case insensitive testing is done.

Parameters:

pszFilename name of file to check for - filename case updated in some cases.

papszSiblingFiles a list of files in the same directory as pszFilename if available, or NULL. This list should have no path components.

Returns:

TRUE if a match is found, or FALSE if not.

References CPLGetFilename(), and VSIStatL().

17.1.2.6 const char* CPLCleanTrailingSlash (const char * *pszPath*)

Remove trailing forward/backward slash from the path for unix/windows resp.

Returns a string containing the portion of the passed path string with trailing slash removed. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLCleanTrailingSlash( "abc/def/" ) == "abc/def"
CPLCleanTrailingSlash( "abc/def" ) == "abc/def"
CPLCleanTrailingSlash( "c:\abc\def\" ) == "c:\abc\def"
CPLCleanTrailingSlash( "c:\abc\def" ) == "c:\abc\def"
CPLCleanTrailingSlash( "abc" ) == "abc"
```

Parameters:

pszPath the path to be cleaned up

Returns:

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLCleanTrailingSlash().

Referenced by CPLCleanTrailingSlash().

17.1.2.7 void CPLCloseShared (FILE * *fp*)

Close shared file.

Dereferences the indicated file handle, and closes it if the reference count has dropped to zero. A **CPLError()** (p. 299) is issued if the file is not in the shared file list.

Parameters:

fp file handle from **CPLOpenShared()** (p. 287) to deaccess.

References VSIFCloseL().

17.1.2.8 char** CPLCorrespondingPaths (const char * *pszOldFilename*, const char * *pszNewFilename*, char ** *papszFileList*)

Identify corresponding paths.

Given a prototype old and new filename this function will attempt to determine corresponding names for a set of other old filenames that will rename them in a similar manner. This correspondance assumes there are two possibly kinds of renaming going on. A change of path, and a change of filename stem.

If a consistent renaming cannot be established for all the files this function will return indicating an error.

The returned file list becomes owned by the caller and should be destroyed with **CSLDestroy()** (p. 328).

Parameters:

pszOldFilename path to old prototype file.

pszNewFilename path to new prototype file.

papszFileList list of other files associated with pszOldFilename to rename similarly.

Returns:

a list of files corresponding to papszFileList but renamed to correspond to pszNewFilename.

References CPLCorrespondingPaths(), CPLFormFilename(), CPLGetBasename(), CPLGetFilename(), and CPLGetPath().

Referenced by CPLCorrespondingPaths().

17.1.2.9 double CPLDecToPackedDMS (double *dfDec*)

Convert decimal degrees into packed DMS value (DDDMMMSSS.SS).

This function converts a value, specified in decimal degrees into packed DMS angle. The standard packed DMS format is:

degrees * 1000000 + minutes * 1000 + seconds

See also **CPLPackedDMSToDec()** (p. 288).

Parameters:

dfDec Angle in decimal degrees.

Returns:

Angle in packed DMS format.

17.1.2.10 void CPLDumpSharedList (FILE * *fp*)

Report open shared files.

Dumps all open shared files to the indicated file handle. If the file handle is NULL information is sent via the **CPLDebug()** (p. 298) call.

Parameters:

fp File handle to write to.

17.1.2.11 const char* CPLExtractRelativePath (const char * *pszBaseDir*, const char * *pszTarget*, int * *pbGotRelative*)

Get relative path from directory to target file.

Computes a relative path for pszTarget relative to pszBaseDir. Currently this only works if they share a common base path. The returned path is normally into the pszTarget string. It should only be considered valid as long as pszTarget is valid or till the next call to this function, whichever comes first.

Parameters:

pszBaseDir the name of the directory relative to which the path should be computed. pszBaseDir may be NULL in which case the original target is returned without relitivizing.

pszTarget the filename to be changed to be relative to pszBaseDir.

pbGotRelative Pointer to location in which a flag is placed indicating that the returned path is relative to the basename (TRUE) or not (FALSE). This pointer may be NULL if flag is not desired.

Returns:

an adjusted path or the original if it could not be made relative to the pszBaseFile's path.

References CPLExtractRelativePath(), and CPLGetPath().

Referenced by CPLExtractRelativePath().

17.1.2.12 char* CPLFGets (char * *pszBuffer*, int *nBufferSize*, FILE * *fp*)

Reads in at most one less than nBufferSize characters from the fp stream and stores them into the buffer pointed to by pszBuffer. Reading stops after an EOF or a newline. If a newline is read, it is _not_ stored into the buffer. A '\0' is stored after the last character in the buffer. All three types of newline terminators recognized by the **CPLFGets()** (p. 281): single '\r' and '\n' and '\r\n' combination.

Parameters:

pszBuffer pointer to the targeting character buffer.

nBufferSize maximum size of the string to read (not including termonating '\0').

fp file pointer to read from.

Returns:

pointer to the pszBuffer containing a string read from the file or NULL if the error or end of file was encountered.

17.1.2.13 const char* CPLFormCIFilename (const char * *pszPath*, const char * *pszBasename*, const char * *pszExtension*)

Case insensitive file searching, returing full path.

This function tries to return the path to a file regardless of whether the file exactly matches the basename, and extension case, or is all upper case, or all lower case. The path is treated as case sensitive. This function is equivelent to **CPLFormFilename()** (p. 282) on case insensitive file systems (like Windows).

Parameters:

pszPath directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.

pszBasename file basename. May optionally have path and/or extension. May not be NULL.

pszExtension file extension, optionally including the period. May be NULL.

Returns:

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLFormCIFilename(), CPLFormFilename(), and VSIStatL().

Referenced by CPLFormCIFilename().

17.1.2.14 const char* CPLFormFilename (const char * *pszPath*, const char * *pszBasename*, const char * *pszExtension*)

Build a full file path from a passed path, file basename and extension.

The path, and extension are optional. The basename may in fact contain an extension if desired.

```
CPLFormFilename("abc/xyz","def", ".dat" ) == "abc/xyz/def.dat"
CPLFormFilename(NULL,"def", NULL ) == "def"
CPLFormFilename(NULL,"abc/def.dat", NULL ) == "abc/def.dat"
CPLFormFilename("/abc/xyz/","def.dat", NULL ) == "/abc/xyz/def.dat"
```

Parameters:

pszPath directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.

pszBasename file basename. May optionally have path and/or extension. May not be NULL.

pszExtension file extension, optionally including the period. May be NULL.

Returns:

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLFormFilename().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLCorrespondingPaths(), CPLFormCIFilename(), CPLFormFilename(), CPLGenerateTempFilename(), and CPLUnlinkTree().

17.1.2.15 const char* CPLGenerateTempFilename (const char * *pszStem*)

Generate temporary file name.

Returns a filename that may be used for a temporary file. The location of the file tries to follow operating system semantics but may be forced via the CPL_TMPDIR configuration option.

Parameters:

pszStem if non-NULL this will be part of the filename.

Returns:

a filename which is valid till the next CPL call in this thread.

References CPLFormFilename(), and CPLGenerateTempFilename().

Referenced by CPLGenerateTempFilename().

17.1.2.16 const char* CPLGetBasename (const char * *pszFullFilename*)

Extract basename (non-directory, non-extension) portion of filename.

Returns a string containing the file basename portion of the passed name. If there is no basename (passed value ends in trailing directory separator, or filename starts with a dot) an empty string is returned.

```
CPLGetBasename( "abc/def.xyz" ) == "def"
CPLGetBasename( "abc/def" ) == "def"
CPLGetBasename( "abc/def/" ) == ""
```

Parameters:

pszFullFilename the full filename potentially including a path.

Returns:

just the non-directory, non-extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLGetBasename().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLCorrespondingPaths(), and CPLGetBasename().

17.1.2.17 char* CPLGetCurrentDir (void)

Get the current working directory name.

Returns:

a pointer to buffer, containing current working directory path or NULL in case of error. User is responsible to free that buffer after usage with CPLFree() function. If HAVE_GETCWD macro is not defined, the function returns NULL.

References CPLGetCurrentDir().

Referenced by CPLGetCurrentDir().

17.1.2.18 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. 285).

```
CPLGetDirname( "abc/def.xyz" ) == "abc"
CPLGetDirname( "/abc/def/" ) == "/abc/def"
CPLGetDirname( "/" ) == "/"
CPLGetDirname( "/abc/def" ) == "/abc"
CPLGetDirname( "abc" ) == "."
```

Parameters:

pszFilename the filename potentially including a path.

Returns:

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLGetDirname().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLGetDirname().

17.1.2.19 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 platform.

Parameters:

pszPathBuf the buffer into which the path is placed.

nMaxLength the buffer size, MAX_PATH+1 is suggested.

Returns:

FALSE on failure or TRUE on success.

References CPLGetExecPath().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLGetExecPath().

17.1.2.20 const char* CPLGetExtension (const char * *pszFullFilename*)

Extract filename extension from full filename.

Returns a string containing the extention portion of the passed name. If there is no extension (the filename has no dot) an empty string is returned. The returned extension will not include the period.

```
CPLGetExtension( "abc/def.xyz" ) == "xyz"
CPLGetExtension( "abc/def" ) == ""
```

Parameters:

pszFullFilename the full filename potentially including a path.

Returns:

just the extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLGetExtension().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLGetExtension().

17.1.2.21 const char* CPLGetFilename (const char * *pszFullFilename*)

Extract non-directory portion of filename.

Returns a string containing the bare filename portion of the passed filename. If there is no filename (passed value ends in trailing directory separator) an empty string is returned.

```
CPLGetFilename( "abc/def.xyz" ) == "def.xyz"
CPLGetFilename( "/abc/def/" ) == ""
CPLGetFilename( "abc/def" ) == "def"
```

Parameters:

pszFullFilename the full filename potentially including a path.

Returns:

just the non-directory portion of the path (points back into original string).

References CPLGetFilename().

Referenced by CPLCheckForFile(), CPLCorrespondingPaths(), and CPLGetFilename().

17.1.2.22 const char* CPLGetPath (const char * *pszFilename*)

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLGetPath( "abc/def.xyz" ) == "abc"
CPLGetPath( "/abc/def/" ) == "/abc/def"
CPLGetPath( "/" ) == "/"
CPLGetPath( "/abc/def" ) == "/abc"
CPLGetPath( "abc" ) == ""
```

Parameters:

pszFilename the filename potentially including a path.

Returns:

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLGetPath().

Referenced by CPLCorrespondingPaths(), CPLExtractRelativePath(), and CPLGetPath().

17.1.2.23 CPLSharedFileInfo* CPLGetSharedList (int * *pnCount*)

Fetch list of open shared files.

Parameters:

pnCount place to put the count of entries.

Returns:

the pointer to the first in the array of shared file info structures.

17.1.2.24 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. 299) or not are not currently well defined, and will have to be resolved in the future.

Currently **CPLGetSymbol()** (p. 286) doesn't try to:

- prevent the reference count on the library from going up for every request, or given any opportunity to unload the library.
- Attempt to look for the library in non-standard locations.
- Attempt to try variations on the symbol name, like pre-prending or post-pending an underscore.

Some of these issues may be worked on in the future.

Parameters:

pszLibrary the name of the shared library or DLL containing the function. May contain path to file. If not system supplies search paths will be used.

pszSymbolName the name of the function to fetch a pointer to.

Returns:

A pointer to the function if found, or NULL if the function isn't found, or the shared library can't be loaded.

References CPLGetSymbol().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLGetSymbol().

17.1.2.25 int CPLIsFilenameRelative (const char * *pszFilename*)

Is filename relative or absolute?

The test is filesystem convention agnostic. That is it will test for Unix style and windows style path conventions regardless of the actual system in use.

Parameters:

pszFilename the filename with path to test.

Returns:

TRUE if the filename is relative or FALSE if it is absolute.

References CPLIsFilenameRelative().

Referenced by CPLIsFilenameRelative(), and CPLProjectRelativeFilename().

17.1.2.26 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. 299) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSIMalloc() to get the memory, so any hooking of VSIMalloc() will apply to **CPLMalloc()** (p. 287) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLMalloc()** (p. 287).

Parameters:

nSize size (in bytes) of memory block to allocate.

Returns:

pointer to newly allocated memory, only NULL if nSize is zero.

17.1.2.27 FILE* CPLOpenShared (const char * *pszFilename*, const char * *pszAccess*, int *bLarge*)

Open a shared file handle.

Some operating systems have limits on the number of file handles that can be open at one time. This function attempts to maintain a registry of already open file handles, and reuse existing ones if the same file is requested by another part of the application.

Note that access is only shared for access types "r", "rb", "r+" and "rb+". All others will just result in direct VSIOpen() calls. Keep in mind that a file is only reused if the file name is exactly the same. Different names referring to the same file will result in different handles.

The VSIFOpen() or **VSIFOpenL()** (p. 336) function is used to actually open the file, when an existing file handle can't be shared.

Parameters:

pszFilename the name of the file to open.

pszAccess the normal fopen()/VSIFOpen() style access string.

bLarge If TRUE **VSIFOpenL()** (p. 336) (for large files) will be used instead of VSIFOpen().

Returns:

a file handle or NULL if opening fails.

References VSIFOpenL().

17.1.2.28 double CPLPackedDMSToDec (double *dfPacked*)

Convert a packed DMS value (DDDMMMSSS.SS) into decimal degrees.

This function converts a packed DMS angle to seconds. The standard packed DMS format is:

degrees * 1000000 + minutes * 1000 + seconds

Example: ang = 120025045.25 yields deg = 120 min = 25 sec = 45.25

The algorithm used for the conversion is as follows:

1. The absolute value of the angle is used.

2. The degrees are separated out: deg = ang/1000000 (fractional portion truncated)

3. The minutes are separated out: min = (ang - deg * 1000000) / 1000 (fractional portion truncated)

4. The seconds are then computed: sec = ang - deg * 1000000 - min * 1000

5. The total angle in seconds is computed: sec = deg * 3600.0 + min * 60.0 + sec

6. The sign of sec is set to that of the input angle.

Packed DMS values used by the USGS GCTP package and probably by other software.

NOTE: This code does not validate input value. If you give the wrong value, you will get the wrong result.

Parameters:

dfPacked Angle in packed DMS format.

Returns:

Angle in decimal degrees.

17.1.2.29 int CPLPrintDouble (char * *pszBuffer*, const char * *pszFormat*, double *dfValue*, const char * *pszLocale*)

Print double value into specified string buffer. Exponential character flag 'E' (or 'e') will be replaced with 'D', as in Fortran. Resulting string will not to be NULL-terminated.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

pszFormat Format specifier (for example, "%16.9E").

dfValue Numerical value to print.

pszLocale Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. With the pszLocale option we can control what exact locale will be used for printing a numeric value to the string (in most cases it should be C/POSIX).

Returns:

Number of characters printed.

17.1.2.30 int CPLPrintInt32 (char * *pszBuffer*, GInt32 *iValue*, int *nMaxLen*)

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

iValue Numerical value to print.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

17.1.2.31 int CPLPrintPointer (char * *pszBuffer*, void * *pValue*, int *nMaxLen*)

Print pointer value into specified string buffer. This string will not be NULL-terminated.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

pValue Pointer to ASCII encode.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

17.1.2.32 int CPLPrintString (char * *pszDest*, const char * *pszSrc*, int *nMaxLen*)

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest.

Parameters:

pszDest Pointer to the destination string buffer. Should be large enough to hold the resulting string.

pszSrc Pointer to the source buffer.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

17.1.2.33 int CPLPrintStringFill (char * *pszDest*, const char * *pszSrc*, int *nMaxLen*)

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest. Remainder of the destination string will be filled with space characters. This is only difference from the PrintString().

Parameters:

pszDest Pointer to the destination string buffer. Should be large enough to hold the resulting string.

pszSrc Pointer to the source buffer.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

17.1.2.34 int CPLPrintTime (char * *pszBuffer*, int *nMaxLen*, const char * *pszFormat*, const struct tm * *poBrokenTime*, const char * *pszLocale*)

Print specified time value accordingly to the format options and specified locale name. This function does following:

- if locale parameter is not NULL, the current locale setting will be stored and replaced with the specified one;
- format time value with the strftime(3) function;
- restore back current locale, if was saved.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

pszFormat Controls the output format. Options are the same as for strftime(3) function.

poBrokenTime Pointer to the broken-down time structure. May be requested with the VSIGMTime() and VSILocalTime() functions.

pszLocale Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. Be aware that it may be unsuitable to use current locale for printing time, because all names will be printed in your native language, as well as time format settings also may be ajusted differently from the C/POSIX defaults. To solve these problems this option was introdiced.

Returns:

Number of characters printed.

17.1.2.35 int CPLPrintUIntBig (char * *pszBuffer*, GUIntBig *iValue*, int *nMaxLen*)

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

iValue Numerical value to print.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

17.1.2.36 const char* CPLProjectRelativeFilename (const char * *pszProjectDir*, const char * *pszSecondaryFilename*)

Find a file relative to a project file.

Given the path to a "project" directory, and a path to a secondary file referenced from that project, build a path to the secondary file that the current application can use. If the secondary path is already absolute, rather than relative, then it will be returned unaltered.

Examples:

```
CPLProjectRelativeFilename("abc/def","tmp/abc.gif") == "abc/def/tmp/abc.gif"
CPLProjectRelativeFilename("abc/def","/tmp/abc.gif") == "/tmp/abc.gif"
CPLProjectRelativeFilename("/xy", "abc.gif") == "/xy/abc.gif"
CPLProjectRelativeFilename("/abc/def","../abc.gif") == "/abc/def/../abc.gif"
CPLProjectRelativeFilename("C:\WIN","abc.gif") == "C:\WIN\abc.gif"
```

Parameters:

pszProjectDir the directory relative to which the secondary files path should be interpreted.

pszSecondaryFilename the filename (potentially with path) that is to be interpreted relative to the project directory.

Returns:

a composed path to the secondary file. The returned string is internal and should not be altered, freed, or depending on past the next CPL call.

References CPLIsFilenameRelative(), and CPLProjectRelativeFilename().

Referenced by CPLProjectRelativeFilename().

17.1.2.37 const char* CPLReadLine (FILE * *fp*)

Simplified line reading from text file.

Read a line of text from the given file handle, taking care to capture CR and/or LF and strip off ... equivelent of DKReadLine(). Pointer to an internal buffer is returned. The application shouldn't free it, or depend on it's value past the next call to **CPLReadLine()** (p. 291).

Note that **CPLReadLine()** (p. 291) uses VSIFGets(), so any hooking of VSI file services should apply to **CPLReadLine()** (p. 291) as well.

CPLReadLine() (p. 291) maintains an internal buffer, which will appear as a single block memory leak in some circumstances. **CPLReadLine()** (p. 291) may be called with a NULL FILE * at any time to free this working buffer.

Parameters:

fp file pointer opened with VSIFOpen().

Returns:

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

17.1.2.38 const char* CPLReadLineL (FILE * *fp*)

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. 291), but reading from a large file API handle.

Parameters:

fp file pointer opened with **VSIFOpenL()** (p. 336).

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 VSIFReadL(), VSIFSeekL(), and VSIFTellL().

17.1.2.39 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. 299) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSIRealloc() to get the memory, so any hooking of VSIRealloc() will apply to **CPLRealloc()** (p. 292) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLRealloc()** (p. 292).

It is also safe to pass NULL in as the existing memory block for **CPLRealloc()** (p. 292), in which case it uses VSIMalloc() to allocate a new block.

Parameters:

pData existing memory block which should be copied to the new block.

nNewSize new size (in bytes) of memory block to allocate.

Returns:

pointer to allocated memory, only NULL if nNewSize is zero.

17.1.2.40 const char* CPLResetExtension (const char * *pszPath*, const char * *pszExt*)

Replace the extension with the provided one.

Parameters:

pszPath the input path, this string is not altered.

pszExt the new extension to apply to the given path.

Returns:

an altered filename with the new extension. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLResetExtension().

Referenced by CPLResetExtension().

17.1.2.41 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. 276) to convert string to double value, so it uses a comma as a decimal delimiter.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

Double value, converted from its ASCII form.

References CPLAtof().

17.1.2.42 long CPLScanLong (const char * *pszString*, int *nMaxLength*)

Scan up to a maximum number of characters from a string and convert the result to a long.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

Long value, converted from its ASCII form.

17.1.2.43 void* CPLScanPointer (const char * *pszString*, int *nMaxLength*)

Extract pointer from string.

Scan up to a maximum number of characters from a string and convert the result to a pointer.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

pointer value, converted from its ASCII form.

17.1.2.44 char* CPLScanString (const char * *pszString*, int *nMaxLength*, int *bTrimSpaces*, int *bNormalize*)

Scan up to a maximum number of characters from a given string, allocate a buffer for a new string and fill it with scanned characters.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to read. Less characters will be read if a null character is encountered.

bTrimSpaces If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.

bNormalize If TRUE, replace ':' symbol with the '_'. It is needed if resulting string will be used in CPL dictionaries.

Returns:

Pointer to the resulting string buffer. Caller responsible to free this buffer with CPLFree().

17.1.2.45 GUIntBig CPLScanUIntBig (const char * *pszString*, int *nMaxLength*)

Extract big integer from string.

Scan up to a maximum number of characters from a string and convert the result to a GUIntBig.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

GUIntBig value, converted from its ASCII form.

17.1.2.46 unsigned long CPLScanULong (const char * *pszString*, int *nMaxLength*)

Scan up to a maximum number of characters from a string and convert the result to a unsigned long.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

Unsigned long value, converted from its ASCII form.

17.1.2.47 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. 299) instead of returning NULL. It uses VSIStrdup(), so any hooking of that function will apply to **CPLStrdup()** (p. 295) as well. Memory allocated with **CPLStrdup()** (p. 295) can be freed with CPLFree() or VSIFree().

It is also safe to pass a NULL string into **CPLStrdup()** (p. 295). **CPLStrdup()** (p. 295) will allocate and return a zero length string (as opposed to a NULL string).

Parameters:

pszString input string to be duplicated. May be NULL.

Returns:

pointer to a newly allocated copy of the string. Free with CPLFree() or VSIFree().

17.1.2.48 char* CPLStrlwr (char * *pszString*)

Convert each characters of the string to lower case.

For example, "ABcdE" will be converted to "abcde". This function is locale dependent.

Parameters:

pszString input string to be converted.

Returns:

pointer to the same string, pszString.

17.1.2.49 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. 296) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. 276) function.

Parameters:

nptr Pointer to string to convert.

endptr If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

Returns:

Converted value, if any.

References CPLStrtod(), and CPLStrtodDelim().

Referenced by CPLAtof(), and CPLStrtod().

17.1.2.50 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. 276) function.

Parameters:

nptr Pointer to string to convert.

endptr If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

point Decimal delimiter.

Returns:

Converted value, if any.

References CPLStrtodDelim().

Referenced by CPLAtofDelim(), CPLAtofM(), CPLStrtod(), CPLStrtodDelim(), and CPLStrtofDelim().

17.1.2.51 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. 297) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. 276) function.

Parameters:

nptr Pointer to string to convert.

endptr If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

Returns:

Converted value, if any.

References CPLStrtof(), and CPLStrtofDelim().

Referenced by CPLStrtof().

17.1.2.52 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. 276) function.

Parameters:

nptr Pointer to string to convert.

endptr If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

point Decimal delimiter.

Returns:

Converted value, if any.

References CPLStrtodDelim(), and CPLStrtofDelim().

Referenced by CPLStrtof(), and CPLStrtofDelim().

17.1.2.53 int CPLUnlinkTree (const char * *pszPath*)

Returns:

0 on successful completion, -1 if function fails.

References CPLFormFilename(), VSIRmdir(), and VSIUnlink().

17.2 cpl_error.h File Reference

```
#include "cpl_port.h"
```

Functions

- void **CPLError** (CPLErr eErrClass, int err_no, const char *fmt,...)
- void **CPLErrorReset** (void)
- int **CPLGetLastErrorNo** (void)
- CPLErr **CPLGetLastErrorType** (void)
- const char * **CPLGetLastErrorMsg** (void)
- CPLErrorHandler **CPLSetErrorHandler** (CPLErrorHandler)
- void **CPLPushErrorHandler** (CPLErrorHandler)
- void **CPLPopErrorHandler** (void)
- void **CPLDebug** (const char *, const char *,...)
- void **_CPLAssert** (const char *, const char *, int)

17.2.1 Detailed Description

CPL error handling services.

17.2.2 Function Documentation

17.2.2.1 void _CPLAssert (const char * *pszExpression*, const char * *pszFile*, int *iLine*)

Report failure of a logical assertion.

Applications would normally use the CPLAssert() macro which expands into code calling **_CPLAssert()** (p. 298) only if the condition fails. **_CPLAssert()** (p. 298) will generate a CE_Fatal error call to **CPLError()** (p. 299), 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. 298) directly.

17.2.2.2 void CPLDebug (const char * *pszCategory*, const char * *pszFormat*, ...)

Display a debugging message.

The category argument is used in conjunction with the CPL_DEBUG environment variable to establish if the message should be displayed. If the CPL_DEBUG environment variable is not set, no debug messages are emitted (use CPLError(CE_Warning,...) to ensure messages are displayed). If CPL_DEBUG is set, but is an empty string or the word "ON" then all debug messages are shown. Otherwise only messages whose category appears somewhere within the CPL_DEBUG value are displayed (as determinted by strstr()).

Categories are usually an identifier for the subsystem producing the error. For instance "GDAL" might be used for the GDAL core, and "TIFF" for messages from the TIFF translator.

Parameters:

pszCategory name of the debugging message category.

pszFormat printf() style format string for message to display. Remaining arguments are assumed to be for format.

17.2.2.3 void CPLError (CPLErr *eErrClass*, int *err_no*, const char * *fmt*, ...)

Report an error.

This function reports an error in a manner that can be hooked and reported appropriate by different applications.

The effect of this function can be altered by applications by installing a custom error handling using **CPLSetErrorHandler()** (p. 300).

The eErrClass argument can have the value CE_Warning indicating that the message is an informational warning, CE_Failure indicating that the action failed, but that normal recover mechanisms will be used or CE_Fatal meaning that a fatal error has occured, and that **CPLError()** (p. 299) should not return.

The default behaviour of **CPLError()** (p. 299) is to report errors to stderr, and to abort() after reporting a CE_Fatal error. It is expected that some applications will want to supress 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. 299) and **CPLGetLastErrorMsg()** (p. 299).

Parameters:

eErrClass one of CE_Warning, CE_Failure or CE_Fatal.

err_no the error number (CPLE_*) from **cpl_error.h** (p. 298).

fmt a printf() style format string. Any additional arguments will be treated as arguments to fill in this format in a manner similar to printf().

17.2.2.4 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.

17.2.2.5 const char* CPLGetLastErrorMsg (void)

Get the last error message.

Fetches the last error message posted with **CPLError()** (p. 299), that hasn't been cleared by **CPLErrorReset()** (p. 299). 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.

17.2.2.6 int CPLGetLastErrorNo (void)

Fetch the last error number.

This is the error number, not the error class.

Returns:

the error number of the last error to occur, or CPLE_None (0) if there are no posted errors.

17.2.2.7 CPLErr CPLGetLastErrorType (void)

Fetch the last error type.

This is the error class, not the error number.

Returns:

the error number of the last error to occur, or CE_None (0) if there are no posted errors.

17.2.2.8 void CPLPopErrorHandler (void)

Pop error handler off stack.

Discards the current error handler on the error handler stack, and restores the one in use before the last **CPLPushErrorHandler()** (p. 300) call. This method has no effect if there are no error handlers on the current threads error handler stack.

17.2.2.9 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 untill removed with **CPLPopErrorHandler()** (p. 300).

The **CPLSetErrorHandler()** (p. 300) docs have further information on how CPLError handlers work.

Parameters:

pfnErrorHandlerNew new error handler function.

17.2.2.10 CPLErrorHandler CPLSetErrorHandler (CPLErrorHandler *pfnErrorHandlerNew*)

Install custom error handler.

Allow the library's user to specify his own error handler function. A valid error handler is a C function with the following prototype:

```
void MyErrorHandler(CPLErr eErrClass, int err_no, const char *msg)
```

Pass NULL to come back to the default behavior. The default behaviour (CPLDefaultErrorHandler()) is to write the message to stderr.

The msg will be a partially formatted error message not containing the "ERROR %d:" portion emitted by the default handler. Message formatting is handled by **CPLError()** (p. 299) before calling the handler. If the error handler function is passed a CE_Fatal class error and returns, then **CPLError()** (p. 299) will call abort(). Applications wanting to interrupt this fatal behaviour will have to use longjmp(), or a C++ exception to indirectly exit the function.

Another standard error handler is CPLQuietErrorHandler() which doesn't make any attempt to report the passed error or warning messages but will process debug messages via CPLDefaultErrorHandler.

Note that error handlers set with **CPLSetErrorHandler()** (p. 300) 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 precidence over the global error handlers set with **CPLSetErrorHandler()** (p. 300). Generally speaking **CPLSetErrorHandler()** (p. 300) would be used to set a desired global error handler, while **CPLPushErrorHandler()** (p. 300) would be used to install a temporary local error handler, such as CPLQuietErrorHandler() to suppress error reporting in a limited segment of code.

Parameters:

pfnErrorHandlerNew new error handler function.

Returns:

returns the previously installed error handler.

17.3 cpl_hash_set.h File Reference

```
#include "cpl_port.h"
```

Functions

- CPLHashSet * **CPLHashSetNew** (CPLHashSetHashFunc fnHashFunc, CPLHashSetEqualFunc fnEqualFunc, CPLHashSetFreeEltFunc fnFreeEltFunc)
- void **CPLHashSetDestroy** (CPLHashSet *set)
- int **CPLHashSetSize** (const CPLHashSet *set)
- void **CPLHashSetForeach** (CPLHashSet *set, CPLHashSetIterEltFunc fnIterFunc, void *user_data)
- int **CPLHashSetInsert** (CPLHashSet *set, void *elt)
- void * **CPLHashSetLookup** (CPLHashSet *set, const void *elt)
- int **CPLHashSetRemove** (CPLHashSet *set, const void *elt)
- unsigned long **CPLHashSetHashPointer** (const void *elt)
- int **CPLHashSetEqualPointer** (const void *elt1, const void *elt2)
- unsigned long **CPLHashSetHashStr** (const void *pszStr)
- int **CPLHashSetEqualStr** (const void *pszStr1, const void *pszStr2)

17.3.1 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.

17.3.2 Function Documentation

17.3.2.1 void CPLHashSetDestroy (CPLHashSet * *set*)

Destroys an allocated hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

Parameters:

set the hash set

References _CPLList::pData, and _CPLList::psNext.

17.3.2.2 int CPLHashSetEqualPointer (const void * *elt1*, const void * *elt2*)

Equality function for arbitrary pointers

Parameters:

elt1 the first arbitrary pointer to compare

elt2 the second arbitrary pointer to compare

Returns:

TRUE if the pointers are equal

17.3.2.3 int CPLHashSetEqualStr (const void * *elt1*, const void * *elt2*)

Equality function for strings

Parameters:

elt1 the first string to compare. May be NULL.

elt2 the second string to compare. May be NULL.

Returns:

TRUE if the strings are equal

17.3.2.4 void CPLHashSetForeach (CPLHashSet * *set*, CPLHashSetIterEltFunc *fnIterFunc*, void * *user_data*)

Walk through the hash set and runs the provided function on all the elements

This function is provided the user_data argument of CPLHashSetForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the hash set must *NOT* be modified during the walk.

Parameters:

set the hash set.

fnIterFunc the function called on each element.

user_data the user data provided to the function.

References _CPLList::pData, and _CPLList::psNext.

17.3.2.5 unsigned long CPLHashSetHashPointer (const void * *elt*)

Hash function for an arbitrary pointer

Parameters:

elt the arbitrary pointer to hash

Returns:

the hash value of the pointer

17.3.2.6 unsigned long CPLHashSetHashStr (const void * *elt*)

Hash function for a zero-terminated string

Parameters:

elt the string to hash. May be NULL.

Returns:

the hash value of the string

17.3.2.7 int CPLHashSetInsert (CPLHashSet * *set*, void * *elt*)

Inserts an element into a hash set.

If the element was already inserted in the hash set, the previous element is replaced by the new element. If a free function was provided, it is used to free the previously inserted element

Parameters:

set the hash set

elt the new element to insert in the hash set

Returns:

TRUE if the element was not already in the hash set

17.3.2.8 void* CPLHashSetLookup (CPLHashSet * *set*, const void * *elt*)

Returns the element found in the hash set corresponding to the element to look up The element must not be modified.

Parameters:

set the hash set

elt the element to look up in the hash set

Returns:

the element found in the hash set or NULL

17.3.2.9 CPLHashSet* CPLHashSetNew (CPLHashSetHashFunc *fnHashFunc*, CPLHashSetEqualFunc *fnEqualFunc*, CPLHashSetFreeEltFunc *fnFreeEltFunc*)

Creates a new hash set

The hash function must return a hash value for the elements to insert. If fnHashFunc is NULL, CPLHashSetHashPointer will be used.

The equal function must return if two elements are equal. If fnEqualFunc is NULL, CPLHashSetEqualPointer will be used.

The free function is used to free elements inserted in the hash set, when the hash set is destroyed, when elements are removed or replaced. If fnFreeEltFunc is NULL, elements inserted into the hash set will not be freed.

Parameters:

fnHashFunc hash function. May be NULL.

fnEqualFunc equal function. May be NULL.

fnFreeEltFunc element free function. May be NULL.

Returns:

a new hash set

17.3.2.10 int CPLHashSetRemove (CPLHashSet * *set*, const void * *elt*)

Removes an element from a hash set

Parameters:

set the hash set

elt the new element to remove from the hash set

Returns:

TRUE if the element was in the hash set

References _CPLList::pData, and _CPLList::psNext.

17.3.2.11 int CPLHashSetSize (const CPLHashSet * *set*)

Returns the number of elements inserted in the hash set

Note: this is not the internal size of the hash set

Parameters:

set the hash set

Returns:

the number of elements in the hash set

17.4 cpl_list.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **_CPLList**

Typedefs

- typedef struct **_CPLList CPLList**

Functions

- **CPLList** ∗ **CPLListAppend** (**CPLList** ∗psList, void ∗pData)
- **CPLList** ∗ **CPLListInsert** (**CPLList** ∗psList, void ∗pData, int nPosition)
- **CPLList** ∗ **CPLListGetLast** (**CPLList** ∗psList)
- **CPLList** ∗ **CPLListGet** (**CPLList** ∗psList, int nPosition)
- int **CPLListCount** (**CPLList** ∗psList)
- **CPLList** ∗ **CPLListRemove** (**CPLList** ∗psList, int nPosition)
- void **CPLListDestroy** (**CPLList** ∗psList)
- **CPLList** ∗ **CPLListGetNext** (**CPLList** ∗psElement)
- void ∗ **CPLListGetData** (**CPLList** ∗psElement)

17.4.1 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.

17.4.2 Typedef Documentation

17.4.2.1 typedef struct _CPLList CPLList

List element structure.

17.4.3 Function Documentation

17.4.3.1 CPLList∗ CPLListAppend (CPLList ∗ *psList*, void ∗ *pData*)

Append an object list and return a pointer to the modified list. If the input list is NULL, then a new list is created.

Parameters:

psList pointer to list head.

pData pointer to inserted data object. May be NULL.

Returns:

pointer to the head of modified list.

References _CPLList::pData, and _CPLList::psNext.

17.4.3.2 int CPLListCount (CPLList * *psList*)

Return the number of elements in a list.

Parameters:

psList pointer to list head.

Returns:

number of elements in a list.

References _CPLList::psNext.

17.4.3.3 void CPLListDestroy (CPLList * *psList*)

Destroy a list. Caller responsible for freeing data objects contained in list elements.

Parameters:

psList pointer to list head.

References _CPLList::psNext.

17.4.3.4 CPLList* CPLListGet (CPLList * *psList*, int *nPosition*)

Return the pointer to the specified element in a list.

Parameters:

psList pointer to list head.
nPosition the index of the element in the list, 0 being the first element

Returns:

pointer to the specified element in a list.

References _CPLList::psNext.

17.4.3.5 void* CPLListGetData (CPLList * *psElement*)

Return pointer to the data object contained in given list element.

Parameters:

psElement pointer to list element.

Returns:

pointer to the data object contained in given list element.

References _CPLList::pData.

17.4.3.6 CPLList* CPLListGetLast (CPLList * *psList*)

Return the pointer to last element in a list.

Parameters:

psList pointer to list head.

Returns:

pointer to last element in a list.

References _CPLList::psNext.

17.4.3.7 CPLList* CPLListGetNext (CPLList * *psElement*)

Return the pointer to next element in a list.

Parameters:

psElement pointer to list element.

Returns:

pointer to the list element preceded by the given element.

References _CPLList::psNext.

17.4.3.8 CPLList* CPLListInsert (CPLList * *psList*, void * *pData*, int *nPosition*)

Insert an object into list at specified position (zero based). If the input list is NULL, then a new list is created.

Parameters:

psList pointer to list head.

pData pointer to inserted data object. May be NULL.

nPosition position number to insert an object.

Returns:

pointer to the head of modified list.

References _CPLList::pData, and _CPLList::psNext.

17.4.3.9 CPLList* CPLListRemove (CPLList * *psList*, int *nPosition*)

Remove the element from the specified position (zero based) in a list. Data object contained in removed element must be freed by the caller first.

Parameters:

psList pointer to list head.

nPosition position number to delet an element.

Returns:

pointer to the head of modified list.

References _CPLList::psNext.

17.5 cpl_minixml.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **CPLXMLNode**

Enumerations

- enum **CPLXMLNodeType** {
  CXT_Element = 0, **CXT_Text** = 1, **CXT_Attribute** = 2, **CXT_Comment** = 3,
  CXT_Literal = 4 }

Functions

- **CPLXMLNode** * **CPLParseXMLString** (const char *)
  Parse an XML string into tree form.
- void **CPLDestroyXMLNode** (**CPLXMLNode** *)
  Destroy a tree.
- **CPLXMLNode** * **CPLGetXMLNode** (**CPLXMLNode** *poRoot, const char *pszPath)
  Find node by path.
- **CPLXMLNode** * **CPLSearchXMLNode** (**CPLXMLNode** *poRoot, const char *pszTarget)
  Search for a node in document.
- const char * **CPLGetXMLValue** (**CPLXMLNode** *poRoot, const char *pszPath, const char *pszDefault)
  Fetch element/attribute value.
- **CPLXMLNode** * **CPLCreateXMLNode** (**CPLXMLNode** *poParent, **CPLXMLNodeType** eType, const char *pszText)
  Create an document tree item.
- char * **CPLSerializeXMLTree** (**CPLXMLNode** *psNode)
  Convert tree into string document.
- void **CPLAddXMLChild** (**CPLXMLNode** *psParent, **CPLXMLNode** *psChild)
  Add child node to parent.
- int **CPLRemoveXMLChild** (**CPLXMLNode** *psParent, **CPLXMLNode** *psChild)
  Remove child node from parent.
- void **CPLAddXMLSibling** (**CPLXMLNode** *psOlderSibling, **CPLXMLNode** *psNewSibling)
  Add new sibling.

- **CPLXMLNode** ∗ **CPLCreateXMLElementAndValue** (**CPLXMLNode** ∗psParent, const char ∗pszName, const char ∗pszValue)
  Create an element and text value.
- **CPLXMLNode** ∗ **CPLCloneXMLTree** (**CPLXMLNode** ∗psTree)
  Copy tree.
- int **CPLSetXMLValue** (**CPLXMLNode** ∗psRoot, const char ∗pszPath, const char ∗pszValue)
  Set element value by path.
- void **CPLStripXMLNamespace** (**CPLXMLNode** ∗psRoot, const char ∗pszNameSpace, int bRecurse)
  Strip indicated namespaces.
- void **CPLCleanXMLElementName** (char ∗)
  Make string into safe XML token.
- **CPLXMLNode** ∗ **CPLParseXMLFile** (const char ∗pszFilename)
  Parse XML file into tree.
- int **CPLSerializeXMLTreeToFile** (**CPLXMLNode** ∗psTree, const char ∗pszFilename)
  Write document tree to a file.

17.5.1 Detailed Description

Definitions for CPL mini XML Parser/Serializer.

17.5.2 Enumeration Type Documentation

17.5.2.1 enum CPLXMLNodeType

Enumerator:

CXT_Element Node is an element
CXT_Text Node is a raw text value
CXT_Attribute Node is attribute
CXT_Comment Node is an XML comment.
CXT_Literal Node is a special literal

17.5.3 Function Documentation

17.5.3.1 void CPLAddXMLChild (CPLXMLNode ∗ *psParent*, CPLXMLNode ∗ *psChild*)

Add child node to parent.

The passed child is added to the list of children of the indicated parent. Normally the child is added at the end of the parents child list, but attributes (CXT_Attribute) will be inserted after any other attributes but before any other element type. Ownership of the child node is effectively assumed by the parent node. If the child has siblings (it's psNext is not NULL) they will be trimmed, but if the child has children they are carried with it.

Parameters:

psParent the node to attach the child to. May not be NULL.

psChild the child to add to the parent. May not be NULL. Should not be a child of any other parent.

References CXT_Attribute, CPLXMLNode::eType, CPLXMLNode::psChild, and CPLXMLNode::psNext.

17.5.3.2 void CPLAddXMLSibling (CPLXMLNode * *psOlderSibling*, CPLXMLNode * *psNewSibling*)

Add new sibling.

The passed psNewSibling is added to the end of siblings of the psOlderSibling node. That is, it is added to the end of the psNext chain. There is no special handling if psNewSibling is an attribute. If this is required, use **CPLAddXMLChild()** (p. 311).

Parameters:

psOlderSibling the node to attach the sibling after.

psNewSibling the node to add at the end of psOlderSiblings psNext chain.

References CPLXMLNode::psNext.

17.5.3.3 void CPLCleanXMLElementName (char * *pszTarget*)

Make string into safe XML token.

Modifiies a string in place to try and make it into a legal XML token that can be used as an element name. This is accomplished by changing any characters not legal in a token into an underscore.

NOTE: This function should implement the rules in section 2.3 of `http://www.w3.org/TR/xml11/` but it doesn't yet do that properly. We only do a rough approximation of that.

Parameters:

pszTarget the string to be adjusted. It is altered in place.

17.5.3.4 CPLXMLNode* CPLCloneXMLTree (CPLXMLNode * *psTree*)

Copy tree.

Creates a deep copy of a **CPLXMLNode** (p. 79) tree.

Parameters:

psTree the tree to duplicate.

Returns:

a copy of the whole tree.

References CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

17.5.3.5 CPLXMLNode* CPLCreateXMLElementAndValue (CPLXMLNode * *psParent*, const char * *pszName*, const char * *pszValue*)

Create an element and text value.

This is function is a convenient short form for:

```
CPLXMLNode *psTextNode;
CPLXMLNode *psElementNode;

psElementNode = CPLCreateXMLNode( psParent, CXT_Element, pszName );
psTextNode = CPLCreateXMLNode( psElementNode, CXT_Text, pszValue );

return psElementNode;
```

It creates a CXT_Element node, with a CXT_Text child, and attaches the element to the passed parent.

Parameters:

psParent the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.

pszName the element name to create.

pszValue the text to attach to the element. Must not be NULL.

Returns:

the pointer to the new element node.

References CXT_Element, and CXT_Text.

17.5.3.6 CPLXMLNode* CPLCreateXMLNode (CPLXMLNode * *poParent*, CPLXMLNodeType *eType*, const char * *pszText*)

Create an document tree item.

Create a single **CPLXMLNode** (p. 79) object with the desired value and type, and attach it as a child of the indicated parent.

Parameters:

poParent the parent to which this node should be attached as a child. May be NULL to keep as free standing.

eType the type of the newly created node

pszText the value of the newly created node

Returns:

the newly created node, now owned by the caller (or parent node).

References CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

17.5.3.7 void CPLDestroyXMLNode (CPLXMLNode * *psNode*)

Destroy a tree.

This function frees resources associated with a **CPLXMLNode** (p. 79) and all its children nodes.

Parameters:

psNode the tree to free.

References CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

17.5.3.8 CPLXMLNode* CPLGetXMLNode (CPLXMLNode * *psRoot*, const char * *pszPath*)

Find node by path.

Searches the document or subdocument indicated by psRoot for an element (or attribute) with the given path. The path should consist of a set of element names separated by dots, not including the name of the root element (psRoot). If the requested element is not found NULL is returned.

Attribute names may only appear as the last item in the path.

The search is done from the root nodes children, but all intermediate nodes in the path must be specified. Seaching for "name" would only find a name element or attribute if it is a direct child of the root, not at any level in the subdocument.

If the pszPath is prefixed by "=" then the search will begin with the root node, and it's siblings, instead of the root nodes children. This is particularly useful when searching within a whole document which is often prefixed by one or more "junk" nodes like the <?xml> declaration.

Parameters:

psRoot the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.

pszPath the list of element names in the path (dot separated).

Returns:

the requested element node, or NULL if not found.

References CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

17.5.3.9 const char* CPLGetXMLValue (CPLXMLNode * *psRoot*, const char * *pszPath*, const char * *pszDefault*)

Fetch element/attribute value.

Searches the document for the element/attribute value associated with the path. The corresponding node is internally found with **CPLGetXMLNode()** (p. 314) (see there for details on path handling). Once found, the value is considered to be the first CXT_Text child of the node.

If the attribute/element search fails, or if the found node has not value then the passed default value is returned.

The returned value points to memory within the document tree, and should not be altered or freed.

Parameters:

psRoot the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.

pszPath the list of element names in the path (dot separated). An empty path means get the value of the psRoot node.

pszDefault the value to return if a corresponding value is not found, may be NULL.

Returns:

the requested value or pszDefault if not found.

References CXT_Attribute, CXT_Element, CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

17.5.3.10 CPLXMLNode* CPLParseXMLFile (const char * *pszFilename*)

Parse XML file into tree.

The named file is opened, loaded into memory as a big string, and parsed with **CPLParseXMLString()** (p. 315). Errors in reading the file or parsing the XML will be reported by **CPLError()** (p. 299).

The "large file" API is used, so XML files can come from virtualized files.

Parameters:

pszFilename the file to open.

Returns:

NULL on failure, or the document tree on success.

References VSIFCloseL(), VSIFOpenL(), VSIFReadL(), VSIFSeekL(), and VSIFTellL().

17.5.3.11 CPLXMLNode* CPLParseXMLString (const char * *pszString*)

Parse an XML string into tree form.

The passed document is parsed into a **CPLXMLNode** (p. 79) tree representation. If the document is not well formed XML then NULL is returned, and errors are reported via **CPLError()** (p. 299). No validation beyond wellformedness is done. The **CPLParseXMLFile()** (p. 315) convenience function can be used to parse from a file.

The returned document tree is is owned by the caller and should be freed with **CPLDestroyXMLNode()** (p. 314) when no longer needed.

If the document has more than one "root level" element then those after the first will be attached to the first as siblings (via the psNext pointers) even though there is no common parent. A document with no XML structure (no angle brackets for instance) would be considered well formed, and returned as a single CXT_Text node.

Parameters:

pszString the document to parse.

Returns:

parsed tree or NULL on error.

References CXT_Attribute, CXT_Comment, CXT_Element, CXT_Literal, CXT_Text, and CPLXMLNode::pszValue.

17.5.3.12 int CPLRemoveXMLChild (CPLXMLNode * *psParent*, CPLXMLNode * *psChild*)

Remove child node from parent.

The passed child is removed from the child list of the passed parent, but the child is not destroyed. The child retains ownership of it's own children, but is cleanly removed from the child list of the parent.

Parameters:

psParent the node to the child is attached to.

psChild the child to remove.

Returns:

TRUE on success or FALSE if the child was not found.

References CPLXMLNode::psChild, and CPLXMLNode::psNext.

17.5.3.13 CPLXMLNode* CPLSearchXMLNode (CPLXMLNode * *psRoot*, const char * *pszElement*)

Search for a node in document.

Searches the children (and potentially siblings) of the documented passed in for the named element or attribute. To search following siblings as well as children, prefix the pszElement name with an equal sign. This function does an in-order traversal of the document tree. So it will first match against the current node, then it's first child, that childs first child, and so on.

Use **CPLGetXMLNode()** (p. 314) to find a specific child, or along a specific node path.

Parameters:

psRoot the subtree to search. This should be a node of type CXT_Element. NULL is safe.

pszElement the name of the element or attribute to search for.

Returns:

The matching node or NULL on failure.

References CXT_Attribute, CXT_Element, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

17.5.3.14 char* CPLSerializeXMLTree (CPLXMLNode * *psNode*)

Convert tree into string document.

This function converts a **CPLXMLNode** (p. 79) tree representation of a document into a flat string representation. White space indentation is used visually preserve the tree structure of the document. The returned document becomes owned by the caller and should be freed with CPLFree() when no longer needed.

Parameters:

psNode

Returns:

the document on success or NULL on failure.

References CPLXMLNode::psNext.

17.5.3.15 int CPLSerializeXMLTreeToFile (CPLXMLNode * *psTree*, const char * *pszFilename*)

Write document tree to a file.

The passed document tree is converted into one big string (with **CPLSerializeXMLTree()** (p. 316)) and then written to the named file. Errors writing the file will be reported by **CPLError()** (p. 299). The source document tree is not altered. If the output file already exists it will be overwritten.

Parameters:

psTree the document tree to write.

pszFilename the name of the file to write to.

Returns:

TRUE on success, FALSE otherwise.

References VSIFCloseL(), VSIFOpenL(), and VSIFWriteL().

17.5.3.16 int CPLSetXMLValue (CPLXMLNode * *psRoot*, const char * *pszPath*, const char * *pszValue*)

Set element value by path.

Find (or create) the target element or attribute specified in the path, and assign it the indicated value.

Any path elements that do not already exist will be created. The target nodes value (the first CXT_Text child) will be replaced with the provided value.

If the target node is an attribute instead of an element, the name should be prefixed with a #.

Example: CPLSetXMLValue("Citation.Id.Description", "DOQ dataset"); CPLSetXMLValue("Citation.Id.Description.#name", "doq");

Parameters:

psRoot the subdocument to be updated.

pszPath the dot seperated path to the target element/attribute.

pszValue the text value to assign.

Returns:

TRUE on success.

References CXT_Attribute, CXT_Element, CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

17.5.3.17 void CPLStripXMLNamespace (CPLXMLNode * *psRoot*, const char * *pszNamespace*, int *bRecurse*)

Strip indicated namespaces.

The subdocument (psRoot) is recursively examined, and any elements with the indicated namespace prefix will have the namespace prefix stripped from the element names. If the passed namespace is NULL, then all namespace prefixes will be stripped.

Nodes other than elements should remain unaffected. The changes are made "in place", and should not alter any node locations, only the pszValue field of affected nodes.

Parameters:

psRoot the document to operate on.

pszNamespace the name space prefix (not including colon), or NULL.

bRecurse TRUE to recurse over whole document, or FALSE to only operate on the passed node.

References CXT_Attribute, CXT_Element, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

17.6 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**

17.6.1 Detailed Description

ODBC Abstraction Layer (C++).

17.7 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>
```

Defines

- #define **CPL_LSBINT16PTR**(x) ((∗(GByte∗)(x)) | ((∗(GByte∗)((x)+1)) << 8))
- #define **CPL_LSBINT32PTR**(x)

17.7.1 Detailed Description

Core portability definitions for CPL.

17.7.2 Define Documentation

17.7.2.1 #define CPL_LSBINT16PTR(x) ((∗(GByte∗)(x)) | ((∗(GByte∗)((x)+1)) << 8))

Return a Int16 from the 2 bytes ordered in LSB order at address x

17.7.2.2 #define CPL_LSBINT32PTR(x)

Value:

```
((*(GByte*)(x)) | ((*(GByte*)((x)+1)) << 8) | \
                              ((*(GByte*)((x)+2)) << 16) | ((*(GByte*)((x)+3)) << 24))
```

Return a Int32 from the 4 bytes ordered in LSB order at address x

17.8 cpl_quad_tree.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **CPLRectObj**

Functions

- CPLQuadTree * **CPLQuadTreeCreate** (const CPLRectObj *pGlobalBounds, CPLQuadTreeGetBoundsFunc pfnGetBounds)
- void **CPLQuadTreeDestroy** (CPLQuadTree *hQuadtree)
- void **CPLQuadTreeSetBucketCapacity** (CPLQuadTree *hQuadtree, int nBucketCapacity)
- int **CPLQuadTreeGetAdvisedMaxDepth** (int nExpectedFeatures)
- void **CPLQuadTreeSetMaxDepth** (CPLQuadTree *hQuadtree, int nMaxDepth)
- void **CPLQuadTreeInsert** (CPLQuadTree *hQuadtree, void *hFeature)
- void ** **CPLQuadTreeSearch** (const CPLQuadTree *hQuadtree, const CPLRectObj *pAoi, int *pnFeatureCount)
- void **CPLQuadTreeForeach** (const CPLQuadTree *hQuadtree, CPLQuadTreeForeachFunc pfnForeach, void *pUserData)

17.8.1 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

17.8.2 Function Documentation

17.8.2.1 CPLQuadTree* CPLQuadTreeCreate (const CPLRectObj * *pGlobalBounds*, CPLQuadTreeGetBoundsFunc *pfnGetBounds*)

Create a new quadtree

Parameters:

pGlobalBounds a pointer to the global extent of all the elements that will be inserted

pfnGetBounds a user provided function to get the bounding box of the inserted elements

Returns:

a newly allocated quadtree

17.8.2.2 void CPLQuadTreeDestroy (CPLQuadTree * *hQuadTree*)

Destroy a quadtree

Parameters:

hQuadTree the quad tree to destroy

17.8.2.3 void CPLQuadTreeForeach (const CPLQuadTree * *hQuadTree*, CPLQuadTreeForeachFunc *pfnForeach*, void * *pUserData*)

Walk through the quadtree and runs the provided function on all the elements

This function is provided with the user_data argument of pfnForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the quadtree must *NOT* be modified during the walk.

Parameters:

hQuadTree the quad tree

pfnForeach the function called on each element.

pUserData the user data provided to the function.

17.8.2.4 int CPLQuadTreeGetAdvisedMaxDepth (int *nExpectedFeatures*)

Returns the optimal depth of a quadtree to hold nExpectedFeatures

Parameters:

nExpectedFeatures the expected maximum number of elements to be inserted

Returns:

the optimal depth of a quadtree to hold nExpectedFeatures

17.8.2.5 void CPLQuadTreeInsert (CPLQuadTree * *hQuadTree*, void * *hFeature*)

Insert a feature into a quadtree

Parameters:

hQuadTree the quad tree

hFeature the feature to insert

17.8.2.6 void** CPLQuadTreeSearch (const CPLQuadTree * *hQuadTree*, const CPLRectObj * *pAoi*, int * *pnFeatureCount*)

Returns all the elements inserted whose bounding box intersects the provided area of interest

Parameters:

hQuadTree the quad tree

pAoi the pointer to the area of interest

pnFeatureCount the user data provided to the function.

Returns:

an array of features that must be freed with CPLFree

17.8.2.7 void CPLQuadTreeSetBucketCapacity (CPLQuadTree * *hQuadTree*, int *nBucketCapacity*)

Set the maximum capacity of a node of a quadtree. The default value is 8. Note that the maximum capacity will only be honoured if the features inserted have a point geometry. Otherwise it may be exceeded.

Parameters:

hQuadTree the quad tree

nBucketCapacity the maximum capactiy of a node of a quadtree

17.8.2.8 void CPLQuadTreeSetMaxDepth (CPLQuadTree * *hQuadTree*, int *nMaxDepth*)

Set the maximum depth of a quadtree. By default, quad trees have no maximum depth, but a maximum bucket capacity.

Parameters:

hQuadTree the quad tree

nMaxDepth the maximum depth allowed

17.9 cpl_string.h File Reference

```
#include "cpl_vsi.h"
#include "cpl_error.h"
#include "cpl_conv.h"
#include <string>
```

Classes

- class **CPLString**

Functions

- int **CSLCount** (char **papszStrList)
- void **CSLDestroy** (char **papszStrList)
- char ** **CSLDuplicate** (char **papszStrList)
- char ** **CSLMerge** (char **papszOrig, char **papszOverride)

  Merge two lists.

- char ** **CSLTokenizeString2** (const char *pszString, const char *pszDelimeter, int nCSLTFlags)
- char ** **CSLLoad** (const char *pszFname)
- int **CSLFindString** (char **, const char *)
- int **CSLPartialFindString** (char **papszHaystack, const char *pszNeedle)
- int **CSLFindName** (char **papszStrList, const char *pszName)
- int **CSLTestBoolean** (const char *pszValue)
- const char * **CPLParseNameValue** (const char *pszNameValue, char **ppszKey)
- char ** **CSLSetNameValue** (char **papszStrList, const char *pszName, const char *pszValue)
- void **CSLSetNameValueSeparator** (char **papszStrList, const char *pszSeparator)
- char * **CPLEscapeString** (const char *pszString, int nLength, int nScheme)
- char * **CPLUnescapeString** (const char *pszString, int *pnLength, int nScheme)
- char * **CPLBinaryToHex** (int nBytes, const GByte *pabyData)
- GByte * **CPLHexToBinary** (const char *pszHex, int *pnBytes)
- CPLValueType **CPLGetValueType** (const char *pszValue)
- char * **CPLRecodeFromWChar** (const wchar_t *pwszSource, const char *pszSrcEncoding, const char *pszDstEncoding)
- wchar_t * **CPLRecodeToWChar** (const char *pszSource, const char *pszSrcEncoding, const char *pszDstEncoding)

17.9.1 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. 330), and CSLFetchNameValue()).

17.9.2 Function Documentation

17.9.2.1 char* CPLBinaryToHex (int *nBytes*, const GByte * *pabyData*)

Binary to hexadecimal translation.

Parameters:

nBytes number of bytes of binary data in pabyData.

pabyData array of data bytes to translate.

Returns:

hexadecimal translation, zero terminated. Free with CPLFree().

17.9.2.2 char* CPLEscapeString (const char * *pszInput*, int *nLength*, int *nScheme*)

Apply escaping to string to preserve special characters.

This function will "escape" a variety of special characters to make the string suitable to embed within a string constant or to write within a text stream but in a form that can be reconstitued to it's original form. The escaping will even preserve zero bytes allowing preservation of raw binary data.

CPLES_BackslashQuotable(0): This scheme turns a binary string into a form suitable to be placed within double quotes as a string constant. The backslash, quote, '\0' and newline characters are all escaped in the usual C style.

CPLES_XML(1): This scheme converts the '<', '<' and '&' characters into their XML/HTML equivelent (>, < and &) making a string safe to embed as CDATA within an XML element. The '\0' is not escaped and should not be included in the input.

CPLES_URL(2): Everything except alphanumerics and the underscore are converted to a percent followed by a two digit hex encoding of the character (leading zero supplied if needed). This is the mechanism used for encoding values to be passed in URLs.

CPLES_SQL(3): All single quotes are replaced with two single quotes. Suitable for use when constructing literal values for SQL commands where the literal will be enclosed in single quotes.

CPLES_CSV(4): If the values contains commas, 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. 327) currently does not support this format, only **CPLEscapeString()** (p. 325). See cpl_csv.cpp for csv parsing support.

Parameters:

pszInput the string to escape.

nLength The number of bytes of data to preserve. If this is -1 the strlen(pszString) function will be used to compute the length.

nScheme the encoding scheme to use.

Returns:

an escaped, zero terminated string that should be freed with CPLFree() when no longer needed.

17.9.2.3 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.

Parameters:

pszValue the string to analyze

Returns:

returns the type of the value contained in the string.

17.9.2.4 GByte* CPLHexToBinary (const char * *pszHex*, int * *pnBytes*)

Hexadecimal to binary translation

Parameters:

pszHex the input hex encoded string.

pnBytes the returned count of decoded bytes placed here.

Returns:

returns binary buffer of data - free with CPLFree().

17.9.2.5 const char* CPLParseNameValue (const char * *pszNameValue*, char ** *ppszKey*)

Parse NAME=VALUE string into name and value components.

Note that if ppszKey is non-NULL, the key (or name) portion will be allocated using VSIMalloc(), and returned in that pointer. It is the applications responsibility to free this string, but the application should not modify or free the returned value portion.

This function also support "NAME:VALUE" strings and will strip white space from around the delimeter when forming name and value strings.

Eventually CSLFetchNameValue() and friends may be modified to use **CPLParseNameValue()** (p. 326).

Parameters:

pszNameValue string in "NAME=VALUE" format.

ppszKey optional pointer though which to return the name portion.

Returns:

the value portion (pointing into original string).

17.9.2.6 char* CPLRecodeFromWChar (const wchar_t * *pwszSource*, const char * *pszSrcEncoding*, const char * *pszDstEncoding*)

Convert wchar_t string to UTF-8.

Convert a wchar_t string into a multibyte utf-8 string. The only guaranteed supported source encoding is CPL_ENC_UCS2, and the only guaranteed supported destination encodings are CPL_ENC_UTF8, CPL_-ENC_ASCII and CPL_ENC_ISO8859_1. In some cases (ie. using iconv()) other encodings may also be supported.

Note that the wchar_t type varies in size on different systems. On win32 it is normally 2 bytes, and on unix 4 bytes.

If an error occurs an error may, or may not be posted with **CPLError()** (p. 299).

Parameters:

pwszSource the source wchar_t string, terminated with a 0 wchar_t.

pszSrcEncoding the source encoding, typically CPL_ENC_UCS2.

pszDstEncoding the destination encoding, typically CPL_ENC_UTF8.

Returns:

a zero terminated multi-byte string which should be freed with CPLFree(), or NULL if an error occurs.

17.9.2.7 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. 299).

Parameters:

pszSource input multi-byte character string.

pszSrcEncoding source encoding, typically CPL_ENC_UTF8.

pszDstEncoding destination encoding, typically CPL_ENC_UCS2.

Returns:

the zero terminated wchar_t string (to be freed with CPLFree()) or NULL on error.

17.9.2.8 char* CPLUnescapeString (const char * *pszInput*, int * *pnLength*, int *nScheme*)

Unescape a string.

This function does the opposite of **CPLEscapeString()** (p. 325). Given a string with special values escaped according to some scheme, it will return a new copy of the string returned to it's original form.

Parameters:

pszInput the input string. This is a zero terminated string.

pnLength location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.

nScheme the escaped scheme to undo (see **CPLEscapeString()** (p. 325) for a list).

Returns:

a copy of the unescaped string that should be freed by the application using CPLFree() when no longer needed.

17.9.2.9 int CSLCount (char ** *papszStrList*)

Return number of items in a string list.

Returns the number of items in a string list, not counting the terminating NULL. Passing in NULL is safe, and will result in a count of zero.

Lists are counted by iterating through them so long lists will take more time than short lists. Care should be taken to avoid using **CSLCount()** (p. 328) as an end condition for loops as it will result in O(n^2) behavior.

Parameters:

papszStrList the string list to count.

Returns:

the number of entries.

17.9.2.10 void CSLDestroy (char ** *papszStrList*)

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

Parameters:

papszStrList the list to free.

17.9.2.11 char** CSLDuplicate (char ** *papszStrList*)

Clone a string list.

Efficiently allocates a copy of a string list. The returned list is owned by the caller and should be freed with **CSLDestroy()** (p. 328).

Parameters:

papszStrList the input string list.

Returns:

newly allocated copy.

17.9.2.12 int CSLFindName (char ** *papszStrList*, const char * *pszName*)

Find StringList entry with given key name.

Parameters:

papszStrList the string list to search.

pszName the key value to look for (case insensitive).

Returns:

-1 on failure or the list index of the first occurance matching the given key.

17.9.2.13 int CSLFindString (char ** *papszList*, const char * *pszTarget*)

Find a string within a string list.

Returns the index of the entry in the string list that contains the target string. The string in the string list must be a full match for the target, but the search is case insensitive.

Parameters:

papszList the string list to be searched.

pszTarget the string to be searched for.

Returns:

the index of the string within the list or -1 on failure.

17.9.2.14 char** CSLLoad (const char * *pszFname*)

Load a text file into a string list.

The VSI*L API is used, so **VSIFOpenL()** (p. 336) 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. 292)).

If reading the file fails a **CPLError()** (p. 299) will be issued and NULL returned.

Parameters:

pszFname the name of the file to read.

Returns:

a string list with the files lines, now owned by caller.

References VSIFCloseL(), VSIFEofL(), and VSIFOpenL().

17.9.2.15 char** CSLMerge (char ** *papszOrig*, char ** *papszOverride*)

Merge two lists.

The two lists are merged, ensuring that if any keys appear in both that the value from the second (papszOverride) list take precidence.

Parameters:

papszOrig the original list, being modified.

papszOverride the list of items being merged in. This list is unaltered and remains owned by the caller.

Returns:

updated list.

17.9.2.16 int CSLPartialFindString (char ** *papszHaystack*, const char * *pszNeedle*)

Find a substring within a string list.

Returns the index of the entry in the string list that contains the target string as a substring. The search is case sensitive (unlike **CSLFindString()** (p. 329)).

Parameters:

papszHaystack the string list to be searched.

pszNeedle the substring to be searched for.

Returns:

the index of the string within the list or -1 on failure.

17.9.2.17 char** CSLSetNameValue (char ** *papszList*, const char * *pszName*, const char * *pszValue*)

Assign value to name in StringList.

Set the value for a given name in a StringList of "Name=Value" pairs ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

If there is already a value for that name in the list then the value is changed, otherwise a new "Name=Value" pair is added.

Parameters:

papszList the original list, the modified version is returned.

pszName the name to be assigned a value. This should be a well formed token (no spaces or very special characters).

pszValue the value to assign to the name. This should not contain any newlines (CR or LF) but is otherwise pretty much unconstrained. If NULL any corresponding value will be removed.

Returns:

modified stringlist.

17.9.2.18 void CSLSetNameValueSeparator (char ** *papszList*, const char * *pszSeparator*)

Replace the default separator (":" or "=") with the passed separator in the given name/value list.

Note that if a separator other than ":" or "=" is used, the resulting list will not be manipulatable by the CSL name/value functions any more.

The **CPLParseNameValue()** (p. 326) function is used to break the existing lines, and it also strips white space from around the existing delimiter, thus the old separator, and any white space will be replaced by the new separator. For formatting purposes it may be desireable to include some white space in the new separator. eg. ": " or " = ".

Parameters:

papszList the list to update. Component strings may be freed but the list array will remain at the same location.

pszSeparator the new separator string to insert.

17.9.2.19 int CSLTestBoolean (const char * *pszValue*)

Test what boolean value contained in the string.

If pszValue is "NO", "FALSE", "OFF" or "0" will be returned FALSE. Otherwise, TRUE will be returned.

Parameters:

pszValue the string should be tested.

Returns:

TRUE or FALSE.

17.9.2.20 char** CSLTokenizeString2 (const char * *pszString*, const char * *pszDelimiters*, int *nCSLTFlags*)

Tokenize a string.

This function will split a string into tokens based on specified' delimeter(s) with a variety of options. The returned result is a string list that should be freed with **CSLDestroy()** (p. 328) when no longer needed.

The available parsing options are:

- CSLT_ALLOWEMPTYTOKENS: allow the return of empty tokens when two delimiters in a row occur with no other text between them. If not set, empty tokens will be discarded;
- CSLT_STRIPLEADSPACES: strip leading space characters from the token (as reported by isspace());
- CSLT_STRIPENDSPACES: strip ending space characters from the token (as reported by isspace());
- CSLT_HONOURSTRINGS: double quotes can be used to hold values that should not be broken into multiple tokens;
- CSLT_PRESERVEQUOTES: string quotes are carried into the tokens when this is set, otherwise they are removed;
- CSLT_PRESERVEESCAPES: if set backslash escapes (for backslash itself, and for literal double quotes) will be preserved in the tokens, otherwise the backslashes will be removed in processing.

Example:

Parse a string into tokens based on various white space (space, newline, tab) and then print out results and cleanup. Quotes may be used to hold white space in tokens.

```
char **papszTokens;
int i;

papszTokens =
    CSLTokenizeString2( pszCommand, " \t\n",
                        CSLT_HONOURSTRINGS | CSLT_ALLOWEMPTYTOKENS );

for( i = 0; papszTokens != NULL && papszTokens[i] != NULL; i++ )
    printf( "arg %d: '%s'", papszTokens[i] );
CSLDestroy( papszTokens );
```

Parameters:

pszString the string to be split into tokens.

pszDelimiters one or more characters to be used as token delimeters.

nCSLTFlags an ORing of one or more of the CSLT_ flag values.

Returns:

a string list of tokens owned by the caller.

17.10 cpl_vsi.h File Reference

```
#include "cpl_port.h"
#include <unistd.h>
#include <sys/stat.h>
```

Functions

- FILE * **VSIFOpenL** (const char *, const char *)
  Open file.
- int **VSIFCloseL** (FILE *)
  Close file.
- int **VSIFSeekL** (FILE *, vsi_l_offset, int)
  Seek to requested offset.
- vsi_l_offset **VSIFTellL** (FILE *)
  Tell current file offset.
- size_t **VSIFReadL** (void *, size_t, size_t, FILE *)
  Read bytes from file.
- size_t **VSIFWriteL** (const void *, size_t, size_t, FILE *)
  Write bytes to file.
- int **VSIFEofL** (FILE *)
  Test for end of file.
- int **VSIFFlushL** (FILE *)
  Flush pending writes to disk.
- int **VSIFPrintfL** (FILE *, const char *,...)
  Formatted write to file.
- int **VSIStatL** (const char *, VSIStatBufL *)
  Get filesystem object info.
- void * **VSIMalloc2** (size_t nSize1, size_t nSize2)
- void * **VSIMalloc3** (size_t nSize1, size_t nSize2, size_t nSize3)
- char ** **VSIReadDir** (const char *)
  Read names in a directory.
- int **VSIMkdir** (const char *pathname, long mode)
  Create a directory.
- int **VSIRmdir** (const char *pathname)
  Delete a directory.

- int **VSIUnlink** (const char *pathname)

  Delete a file.

- int **VSIRename** (const char *oldpath, const char *newpath)

  Rename a file.

- void **VSIInstallMemFileHandler** (void)

  Install "memory" file system handler.

- void **VSIInstallGZipFileHandler** (void)

  Install GZip file system handler.

- void **VSIInstallZipFileHandler** (void)

  Install ZIP file system handler.

- FILE * **VSIFileFromMemBuffer** (const char *pszFilename, GByte *pabyData, vsi_l_offset nDataLength, int bTakeOwnership)

  Create memory "file" from a buffer.

- GByte * **VSIGetMemFileBuffer** (const char *pszFilename, vsi_l_offset *pnDataLength, int bUnlinkAndSeize)

  Fetch buffer underlying memory file.

17.10.1 Detailed Description

Standard C Covers

The VSI functions are intended to be hookable aliases for Standard C I/O, memory allocation and other system functions. They are intended to allow virtualization of disk I/O so that non file data sources can be made to appear as files, and so that additional error trapping and reporting can be interested. The memory access API is aliased so that special application memory management services can be used.

Is is intended that each of these functions retains exactly the same calling pattern as the original Standard C functions they relate to. This means we don't have to provide custom documentation, and also means that the default implementation is very simple.

17.10.2 Function Documentation

17.10.2.1 int VSIFCloseL (FILE * *fp*)

Close file.

This function closes the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fclose() function.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. 336).

Returns:

0 on success or -1 on failure.

References VSIFCloseL().

Referenced by CPLCloseShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), CSLLoad(), and VSIFCloseL().

17.10.2.2 int VSIFEofL (FILE * *fp*)

Test for end of file.

Returns TRUE (non-zero) if the file read/write offset is currently at the end of the file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. 336).

Returns:

TRUE if at EOF else FALSE.

References VSIFEofL().

Referenced by CSLLoad(), and VSIFEofL().

17.10.2.3 int VSIFFlushL (FILE * *fp*)

Flush pending writes to disk.

For files in write or update mode and on filesystem types where it is applicable, all pending output on the file is flushed to the physical disk.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. 336).

Returns:

0 on success or -1 on error.

References VSIFFlushL().

Referenced by VSIFFlushL().

17.10.2.4 FILE* VSIFileFromMemBuffer (const char * *pszFilename*, GByte * *pabyData*, vsi_l_offset *nDataLength*, int *bTakeOwnership*)

Create memory "file" from a buffer.

A virtual memory file is created from the passed buffer with the indicated filename. Under normal conditions the filename would need to be absolute and within the /vsimem/ portion of the filesystem.

If bTakeOwnership is TRUE, then the memory file system handler will take ownership of the buffer, freeing it when the file is deleted. Otherwise it remains the responsibility of the caller, but should not be freed as long as it might be accessed as a file. In no circumstances does this function take a copy of the pabyData contents.

Parameters:

pszFilename the filename to be created.

pabyData the data buffer for the file.

nDataLength the length of buffer in bytes.

bTakeOwnership TRUE to transfer "ownership" of buffer or FALSE.

Returns:

open file handle on created file (see **VSIFOpenL()** (p. 336)).

References VSIFileFromMemBuffer(), and VSIInstallMemFileHandler().

Referenced by VSIFileFromMemBuffer().

17.10.2.5 FILE* 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 "FILE *" returned by this function is not really 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.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

Parameters:

pszFilename the file to open.

pszAccess access requested (ie. "r", "r+", "w".

Returns:

NULL on failure, or the file handle.

References VSIFOpenL().

Referenced by CPLOpenShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), CSLLoad(), and VSIFOpenL().

17.10.2.6 int VSIFPrintfL (FILE * *fp*, const char * *pszFormat*, ...)

Formatted write to file.

Provides fprintf() style formatted output to a VSI*L file. This formats an internal buffer which is written using **VSIFWriteL()** (p. 338).

Analog of the POSIX fprintf() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. 336).

pszFormat the printf style format string.

Returns:

the number of bytes written or -1 on an error.

References VSIFPrintfL(), and VSIFWriteL().

Referenced by VSIFPrintfL().

17.10.2.7 size_t VSIFReadL (void * *pBuffer*, size_t *nSize*, size_t *nCount*, FILE * *fp*)

Read bytes from file.

Reads nCount objects of nSize bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters:

pBuffer the buffer into which the data should be read (at least nCount * nSize bytes in size.

nSize size of objects to read in bytes.

nCount number of objects to read.

fp file handle opened with **VSIFOpenL()** (p. 336).

Returns:

number of objects successfully read.

References VSIFReadL().

Referenced by CPLParseXMLFile(), CPLReadLineL(), and VSIFReadL().

17.10.2.8 int VSIFSeekL (FILE * *fp*, vsi_l_offset *nOffset*, int *nWhence*)

Seek to requested offset.

Seek to the desired offset (nOffset) in the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. 336).

nOffset offset in bytes.

nWhence one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns:

0 on success or -1 one failure.

References VSIFSeekL().

Referenced by CPLParseXMLFile(), CPLReadLineL(), and VSIFSeekL().

17.10.2.9 vsi_l_offset VSIFTellL (FILE * *fp*)

Tell current file offset.

Returns the current file read/write offset in bytes from the beginning of the file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftell() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. 336).

Returns:

file offset in bytes.

References VSIFTellL().

Referenced by CPLParseXMLFile(), CPLReadLineL(), and VSIFTellL().

17.10.2.10 size_t VSIFWriteL (const void * *pBuffer*, size_t *nSize*, size_t *nCount*, FILE * *fp*)

Write bytes to file.

Writess nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

Parameters:

pBuffer the buffer from which the data should be written (at least nCount * nSize bytes in size.

nSize size of objects to read in bytes.

nCount number of objects to read.

fp file handle opened with **VSIFOpenL()** (p. 336).

Returns:

number of objects successfully written.

References VSIFWriteL().

Referenced by CPLSerializeXMLTreeToFile(), VSIFPrintfL(), and VSIFWriteL().

17.10.2.11 GByte* VSIGetMemFileBuffer (const char * *pszFilename*, vsi_l_offset * *pnDataLength*, int *bUnlinkAndSeize*)

Fetch buffer underlying memory file.

This function returns a pointer to the memory buffer underlying a virtual "in memory" file. If bUnlinkAndSeize is TRUE the filesystem object will be deleted, and ownership of the buffer will pass to the caller otherwise the underlying file will remain in existance.

Parameters:

pszFilename the name of the file to grab the buffer of.

pnDataLength (file) length returned in this variable.

bUnlinkAndSeize TRUE to remove the file, or FALSE to leave unaltered.

Returns:

pointer to memory buffer or NULL on failure.

References VSIGetMemFileBuffer().

Referenced by VSIGetMemFileBuffer().

17.10.2.12 void VSIInstallGZipFileHandler (void)

Install GZip file system handler.

A special file handler is installed that allows reading on-the-fly 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`

References VSIInstallGZipFileHandler().

Referenced by VSIInstallGZipFileHandler().

17.10.2.13 void VSIInstallMemFileHandler (void)

Install "memory" file system handler.

A special file handler is installed that allows block of memory to be treated as files. All portions of the file system underneath the base path "/vsimem/" will be handled by this driver.

Normal VSI*L functions can be used freely to create and destroy memory arrays treating them as if they were real file system objects. Some additional methods exist to efficient create memory file system objects without duplicating original copies of the data or to "steal" the block of memory associated with a memory file.

At this time the memory handler does not properly handle directory semantics for the memory portion of the filesystem. The **VSIReadDir()** (p. 341) function is not supported though this will be corrected in the future.

Calling this function repeatedly should do no harm, though it is not necessary. It is already called the first time a virtualizable file access function (ie. **VSIFOpenL()** (p. 336), VSIMkDir(), etc) is called.

This code example demonstrates using GDAL to translate from one memory buffer to another.

```
GByte *ConvertBufferFormat( GByte *pabyInData, vsi_l_offset nInDataLength,
                            vsi_l_offset *pnOutDataLength )
{
    // create memory file system object from buffer.
    VSIFCloseL( VSIFileFromMemBuffer( "/vsimem/work.dat", pabyInData,
                                      nInDataLength, FALSE ) );

    // Open memory buffer for read.
    GDALDatasetH hDS = GDALOpen( "/vsimem/work.dat", GA_ReadOnly );

    // Get output format driver.
    GDALDriverH hDriver = GDALGetDriverByName( "GTiff" );
    GDALDatasetH hOutDS;

    hOutDS = GDALCreateCopy( hDriver, "/vsimem/out.tif", hDS, TRUE, NULL,
                             NULL, NULL );

    // close source file, and "unlink" it.
    GDALClose( hDS );
    VSIUnlink( "/vsimem/work.dat" );

    // seize the buffer associated with the output file.

    return VSIGetMemFileBuffer( "/vsimem/out.tif", pnOutDataLength, TRUE );
}
```

References VSIInstallMemFileHandler().

Referenced by VSIFileFromMemBuffer(), and VSIInstallMemFileHandler().

17.10.2.14 void VSIInstallZipFileHandler (void)

Install ZIP file system handler.

A special file handler is installed that allows reading on-the-fly in ZIP (.zip) archives. All portions of the file system underneath the base path "/vsizip/" will be handled by this driver.

Additional documentation is to be found at `http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip`

References VSIInstallZipFileHandler().

Referenced by VSIInstallZipFileHandler().

17.10.2.15 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. 299). If nSize1 == 0 || nSize2 == 0, a NULL pointer will also be returned. CPLFree() or VSIFree() can be used to free memory allocated by this function.

References VSIMalloc2().

Referenced by VSIMalloc2().

17.10.2.16 void* VSIMalloc3 (size_t *nSize1*, size_t *nSize2*, size_t *nSize3*)

VSIMalloc3 allocates (nSize1 * nSize2 * nSize3) bytes. In case of overflow of the multiplication, or if memory allocation fails, a NULL pointer is returned and a CE_Failure error is raised with **CPLError()**

(p. 299). If nSize1 == 0 || nSize2 == 0 || nSize3 == 0, a NULL pointer will also be returned. CPLFree() or VSIFree() can be used to free memory allocated by this function.

References VSIMalloc3().

Referenced by VSIMalloc3().

17.10.2.17 int VSIMkdir (const char * *pszPathname*, long *mode*)

Create a directory.

Create a new directory with the indicated mode. The mode is ignored on some platforms. A reasonable default mode value would be 0666. This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX mkdir() function.

Parameters:

pszPathname the path to the directory to create.

mode the permissions mode.

Returns:

0 on success or -1 on an error.

References VSIMkdir().

Referenced by VSIMkdir().

17.10.2.18 char** VSIReadDir (const char * *pszPath*)

Read names in a directory.

This function abstracts access to directory contains. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with **CSLDestroy()** (p. 328) when no longer needed.

Note that no error is issued via **CPLError()** (p. 299) if the directory path is invalid, though NULL is returned.

This function used to be known as CPLReadDir(), but the old name is now deprecated.

Parameters:

pszPath the relative, or absolute path of a directory to read.

Returns:

The list of entries in the directory, or NULL if the directory doesn't exist.

References VSIReadDir().

Referenced by VSIReadDir().

17.10.2.19 int VSIRename (const char * *oldpath*, const char * *newpath*)

Rename a file.

Renames a file object in the file system. It should be possible to rename a file onto a new filesystem, but it is safest if this function is only used to rename files that remain in the same directory.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rename() function.

Parameters:

oldpath the name of the file to be renamed.

newpath the name the file should be given.

Returns:

0 on success or -1 on an error.

References VSIRename().

Referenced by VSIRename().

17.10.2.20 int VSIRmdir (const char * *pszDirname*)

Delete a directory.

Deletes a directory object from the file system. On some systems the directory must be empty before it can be deleted.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rmdir() function.

Parameters:

pszDirname the path of the directory to be deleted.

Returns:

0 on success or -1 on an error.

References VSIRmdir().

Referenced by CPLUnlinkTree(), and VSIRmdir().

17.10.2.21 int VSIStatL (const char * *pszFilename*, VSIStatBufL * *psStatBuf*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSIStatBufL structure. For portability only the st_size (size in bytes), and st_mode (file type). This method is similar to VSIStat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function.

Parameters:

pszFilename the path of the filesystem object to be queried.

psStatBuf the structure to load with information.

Returns:

0 on success or -1 on an error.

References VSIStatL().

Referenced by CPLCheckForFile(), CPLFormCIFilename(), and VSIStatL().

17.10.2.22 int VSIUnlink (const char * *pszFilename*)

Delete a file.

Deletes a file object from the file system.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX unlink() function.

Parameters:

pszFilename the path of the file to be deleted.

Returns:

0 on success or -1 on an error.

References VSIUnlink().

Referenced by CPLUnlinkTree(), and VSIUnlink().

17.11 ogr_api.h File Reference

```
#include "ogr_core.h"
```

Functions

- OGRErr **OGR_G_CreateFromWkb** (unsigned char *, OGRSpatialReferenceH, OGRGeometryH *, int)
- OGRErr **OGR_G_CreateFromWkt** (char **, OGRSpatialReferenceH, OGRGeometryH *)
- void **OGR_G_DestroyGeometry** (OGRGeometryH)
- OGRGeometryH **OGR_G_CreateGeometry** (**OGRwkbGeometryType**)
- int **OGR_G_GetDimension** (OGRGeometryH)
- int **OGR_G_GetCoordinateDimension** (OGRGeometryH)
- OGRGeometryH **OGR_G_Clone** (OGRGeometryH)
- void **OGR_G_GetEnvelope** (OGRGeometryH, **OGREnvelope** *)
- OGRErr **OGR_G_ImportFromWkb** (OGRGeometryH, unsigned char *, int)
- OGRErr **OGR_G_ExportToWkb** (OGRGeometryH, OGRwkbByteOrder, unsigned char *)
- int **OGR_G_WkbSize** (OGRGeometryH hGeom)
- OGRErr **OGR_G_ImportFromWkt** (OGRGeometryH, char **)
- OGRErr **OGR_G_ExportToWkt** (OGRGeometryH, char **)
- **OGRwkbGeometryType OGR_G_GetGeometryType** (OGRGeometryH)
- const char * **OGR_G_GetGeometryName** (OGRGeometryH)
- void **OGR_G_DumpReadable** (OGRGeometryH, FILE *, const char *)
- void **OGR_G_FlattenTo2D** (OGRGeometryH)
- void **OGR_G_AssignSpatialReference** (OGRGeometryH, OGRSpatialReferenceH)
- OGRSpatialReferenceH **OGR_G_GetSpatialReference** (OGRGeometryH)
- OGRErr **OGR_G_Transform** (OGRGeometryH, OGRCoordinateTransformationH)
- OGRErr **OGR_G_TransformTo** (OGRGeometryH, OGRSpatialReferenceH)
- void **OGR_G_Segmentize** (OGRGeometryH hGeom, double dfMaxLength)
- int **OGR_G_Intersects** (OGRGeometryH, OGRGeometryH)
- int **OGR_G_Equals** (OGRGeometryH, OGRGeometryH)
- double **OGR_G_GetArea** (OGRGeometryH)
- void **OGR_G_Empty** (OGRGeometryH)
- int **OGR_G_IsEmpty** (OGRGeometryH)
- int **OGR_G_IsSimple** (OGRGeometryH)
- int **OGR_G_GetPointCount** (OGRGeometryH)
- double **OGR_G_GetX** (OGRGeometryH, int)
- double **OGR_G_GetY** (OGRGeometryH, int)
- double **OGR_G_GetZ** (OGRGeometryH, int)
- void **OGR_G_GetPoint** (OGRGeometryH, int iPoint, double *, double *, double *)
- void **OGR_G_SetPoint** (OGRGeometryH, int iPoint, double, double, double)
- void **OGR_G_SetPoint_2D** (OGRGeometryH, int iPoint, double, double)
- void **OGR_G_AddPoint** (OGRGeometryH, double, double, double)
- void **OGR_G_AddPoint_2D** (OGRGeometryH, double, double)
- int **OGR_G_GetGeometryCount** (OGRGeometryH)
- OGRGeometryH **OGR_G_GetGeometryRef** (OGRGeometryH, int)
- OGRErr **OGR_G_AddGeometry** (OGRGeometryH, OGRGeometryH)
- OGRErr **OGR_G_AddGeometryDirectly** (OGRGeometryH, OGRGeometryH)
- OGRErr **OGR_G_RemoveGeometry** (OGRGeometryH, int, int)

- OGRGeometryH **OGRBuildPolygonFromEdges** (OGRGeometryH hLinesAsCollection, int bBestEffort, int bAutoClose, double dfTolerance, OGRErr *peErr)
- OGRFieldDefnH **OGR_Fld_Create** (const char *, **OGRFieldType**)
- void **OGR_Fld_Destroy** (OGRFieldDefnH)
- void **OGR_Fld_SetName** (OGRFieldDefnH, const char *)
- const char * **OGR_Fld_GetNameRef** (OGRFieldDefnH)
- **OGRFieldType OGR_Fld_GetType** (OGRFieldDefnH)
- void **OGR_Fld_SetType** (OGRFieldDefnH, **OGRFieldType**)
- **OGRJustification OGR_Fld_GetJustify** (OGRFieldDefnH)
- void **OGR_Fld_SetJustify** (OGRFieldDefnH, **OGRJustification**)
- int **OGR_Fld_GetWidth** (OGRFieldDefnH)
- void **OGR_Fld_SetWidth** (OGRFieldDefnH, int)
- int **OGR_Fld_GetPrecision** (OGRFieldDefnH)
- void **OGR_Fld_SetPrecision** (OGRFieldDefnH, int)
- void **OGR_Fld_Set** (OGRFieldDefnH, const char *, **OGRFieldType**, int, int, **OGRJustification**)
- const char * **OGR_GetFieldTypeName** (**OGRFieldType**)
- OGRFeatureDefnH **OGR_FD_Create** (const char *)
- void **OGR_FD_Destroy** (OGRFeatureDefnH)
- void **OGR_FD_Release** (OGRFeatureDefnH)
- const char * **OGR_FD_GetName** (OGRFeatureDefnH)
- int **OGR_FD_GetFieldCount** (OGRFeatureDefnH)
- OGRFieldDefnH **OGR_FD_GetFieldDefn** (OGRFeatureDefnH, int)
- int **OGR_FD_GetFieldIndex** (OGRFeatureDefnH, const char *)
- void **OGR_FD_AddFieldDefn** (OGRFeatureDefnH, OGRFieldDefnH)
- **OGRwkbGeometryType OGR_FD_GetGeomType** (OGRFeatureDefnH)
- void **OGR_FD_SetGeomType** (OGRFeatureDefnH, **OGRwkbGeometryType**)
- int **OGR_FD_Reference** (OGRFeatureDefnH)
- int **OGR_FD_Dereference** (OGRFeatureDefnH)
- int **OGR_FD_GetReferenceCount** (OGRFeatureDefnH)
- OGRFeatureH **OGR_F_Create** (OGRFeatureDefnH)
- void **OGR_F_Destroy** (OGRFeatureH)
- OGRFeatureDefnH **OGR_F_GetDefnRef** (OGRFeatureH)
- OGRErr **OGR_F_SetGeometryDirectly** (OGRFeatureH, OGRGeometryH)
- OGRErr **OGR_F_SetGeometry** (OGRFeatureH, OGRGeometryH)
- OGRGeometryH **OGR_F_GetGeometryRef** (OGRFeatureH)
- OGRFeatureH **OGR_F_Clone** (OGRFeatureH)
- int **OGR_F_Equal** (OGRFeatureH, OGRFeatureH)
- int **OGR_F_GetFieldCount** (OGRFeatureH)
- OGRFieldDefnH **OGR_F_GetFieldDefnRef** (OGRFeatureH, int)
- int **OGR_F_GetFieldIndex** (OGRFeatureH, const char *)
- int **OGR_F_IsFieldSet** (OGRFeatureH, int)
- void **OGR_F_UnsetField** (OGRFeatureH, int)
- **OGRField** * **OGR_F_GetRawFieldRef** (OGRFeatureH, int)
- int **OGR_F_GetFieldAsInteger** (OGRFeatureH, int)
- double **OGR_F_GetFieldAsDouble** (OGRFeatureH, int)
- const char * **OGR_F_GetFieldAsString** (OGRFeatureH, int)
- const int * **OGR_F_GetFieldAsIntegerList** (OGRFeatureH, int, int *)
- const double * **OGR_F_GetFieldAsDoubleList** (OGRFeatureH, int, int *)
- char ** **OGR_F_GetFieldAsStringList** (OGRFeatureH, int)
- GByte * **OGR_F_GetFieldAsBinary** (OGRFeatureH, int, int *)

- int **OGR_F_GetFieldAsDateTime** (OGRFeatureH, int, int *, int *, int *, int *, int *, int *, int *)
- void **OGR_F_SetFieldInteger** (OGRFeatureH, int, int)
- void **OGR_F_SetFieldDouble** (OGRFeatureH, int, double)
- void **OGR_F_SetFieldString** (OGRFeatureH, int, const char *)
- void **OGR_F_SetFieldIntegerList** (OGRFeatureH, int, int, int *)
- void **OGR_F_SetFieldDoubleList** (OGRFeatureH, int, int, double *)
- void **OGR_F_SetFieldStringList** (OGRFeatureH, int, char **)
- void **OGR_F_SetFieldRaw** (OGRFeatureH, int, **OGRField** *)
- void **OGR_F_SetFieldBinary** (OGRFeatureH, int, int, GByte *)
- void **OGR_F_SetFieldDateTime** (OGRFeatureH, int, int, int, int, int, int, int, int)
- long **OGR_F_GetFID** (OGRFeatureH)
- OGRErr **OGR_F_SetFID** (OGRFeatureH, long)
- void **OGR_F_DumpReadable** (OGRFeatureH, FILE *)
- OGRErr **OGR_F_SetFrom** (OGRFeatureH, OGRFeatureH, int)
- const char * **OGR_F_GetStyleString** (OGRFeatureH)
- void **OGR_F_SetStyleString** (OGRFeatureH, const char *)
- void **OGR_F_SetStyleStringDirectly** (OGRFeatureH, char *)
- OGRGeometryH **OGR_L_GetSpatialFilter** (OGRLayerH)
- void **OGR_L_SetSpatialFilter** (OGRLayerH, OGRGeometryH)
- void **OGR_L_SetSpatialFilterRect** (OGRLayerH, double, double, double, double)
- OGRErr **OGR_L_SetAttributeFilter** (OGRLayerH, const char *)
- void **OGR_L_ResetReading** (OGRLayerH)
- OGRFeatureH **OGR_L_GetNextFeature** (OGRLayerH)
- OGRFeatureH **OGR_L_GetFeature** (OGRLayerH, long)
- OGRErr **OGR_L_SetFeature** (OGRLayerH, OGRFeatureH)
- OGRErr **OGR_L_CreateFeature** (OGRLayerH, OGRFeatureH)
- OGRErr **OGR_L_DeleteFeature** (OGRLayerH, long)
- OGRFeatureDefnH **OGR_L_GetLayerDefn** (OGRLayerH)
- OGRSpatialReferenceH **OGR_L_GetSpatialRef** (OGRLayerH)
- int **OGR_L_GetFeatureCount** (OGRLayerH, int)
- OGRErr **OGR_L_GetExtent** (OGRLayerH, **OGREnvelope** *, int)
- int **OGR_L_TestCapability** (OGRLayerH, const char *)
- OGRErr **OGR_L_CreateField** (OGRLayerH, OGRFieldDefnH, int)
- OGRErr **OGR_L_StartTransaction** (OGRLayerH)
- OGRErr **OGR_L_CommitTransaction** (OGRLayerH)
- OGRErr **OGR_L_RollbackTransaction** (OGRLayerH)
- void **OGR_DS_Destroy** (OGRDataSourceH)
- const char * **OGR_DS_GetName** (OGRDataSourceH)
- int **OGR_DS_GetLayerCount** (OGRDataSourceH)
- OGRLayerH **OGR_DS_GetLayer** (OGRDataSourceH, int)
- OGRLayerH **OGR_DS_GetLayerByName** (OGRDataSourceH, const char *)
- OGRLayerH **OGR_DS_CreateLayer** (OGRDataSourceH, const char *, OGRSpatialReferenceH, **OGRwkbGeometryType**, char **)
- int **OGR_DS_TestCapability** (OGRDataSourceH, const char *)
- OGRLayerH **OGR_DS_ExecuteSQL** (OGRDataSourceH, const char *, OGRGeometryH, const char *)
- void **OGR_DS_ReleaseResultSet** (OGRDataSourceH, OGRLayerH)
- const char * **OGR_Dr_GetName** (OGRSFDriverH)
- OGRDataSourceH **OGR_Dr_Open** (OGRSFDriverH, const char *, int)
- int **OGR_Dr_TestCapability** (OGRSFDriverH, const char *)

- OGRDataSourceH **OGR_Dr_CreateDataSource** (OGRSFDriverH, const char *, char **)
- OGRDataSourceH **OGROpen** (const char *, int, OGRSFDriverH *)
- void **OGRRegisterDriver** (OGRSFDriverH)
- int **OGRGetDriverCount** (void)
- OGRSFDriverH **OGRGetDriver** (int)
- void **OGRRegisterAll** (void)
- void **OGRCleanupAll** (void)
- OGRStyleMgrH **OGR_SM_Create** (void *hStyleTable)
- void **OGR_SM_Destroy** (OGRStyleMgrH hSM)
- const char * **OGR_SM_InitFromFeature** (OGRStyleMgrH hSM, OGRFeatureH hFeat)
- int **OGR_SM_InitStyleString** (OGRStyleMgrH hSM, const char *pszStyleString)
- int **OGR_SM_GetPartCount** (OGRStyleMgrH hSM, const char *pszStyleString)
- OGRStyleToolH **OGR_SM_GetPart** (OGRStyleMgrH hSM, int nPartId, const char *pszStyleString)
- int **OGR_SM_AddPart** (OGRStyleMgrH hSM, OGRStyleToolH hST)
- OGRStyleToolH **OGR_ST_Create** (**OGRSTClassId** eClassId)
- void **OGR_ST_Destroy** (OGRStyleToolH hST)
- **OGRSTClassId OGR_ST_GetType** (OGRStyleToolH hST)
- **OGRSTUnitId OGR_ST_GetUnit** (OGRStyleToolH hST)
- void **OGR_ST_SetUnit** (OGRStyleToolH hST, **OGRSTUnitId** eUnit, double dfGroundPaperScale)
- const char * **OGR_ST_GetParamStr** (OGRStyleToolH hST, int eParam, int *bValueIsNull)
- int **OGR_ST_GetParamNum** (OGRStyleToolH hST, int eParam, int *bValueIsNull)
- double **OGR_ST_GetParamDbl** (OGRStyleToolH hST, int eParam, int *bValueIsNull)
- void **OGR_ST_SetParamStr** (OGRStyleToolH hST, int eParam, const char *pszValue)
- void **OGR_ST_SetParamNum** (OGRStyleToolH hST, int eParam, int nValue)
- const char * **OGR_ST_GetStyleString** (OGRStyleToolH hST)
- int **OGR_ST_GetRGBFromString** (OGRStyleToolH hST, const char *pszColor, int *pnRed, int *pnGreen, int *pnBlue, int *pnAlpha)

17.11.1 Detailed Description

C API and defines for **OGRFeature** (p. 101), **OGRGeometry** (p. 127), and **OGRDataSource** (p. 92) related classes.

See also: **ogr_geometry.h** (p. 416), **ogr_feature.h** (p. 415), **ogrsf_frmts.h** (p. 429), **ogr_featurestyle.h** (p. **??**)

17.11.2 Function Documentation

17.11.2.1 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.

This function is the same as the C++ method **OGRSFDriver::CreateDataSource()** (p. 217).

Parameters:

hDriver handle to the driver on which data source creation is based.

pszName the name for the new data source.

papszOptions a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns:

NULL is returned on failure, or a new **OGRDataSource** (p. 92) handle on success.

References OGRSFDriver::CreateDataSource(), OGRDataSource::GetDriver(), OGR_Dr_-CreateDataSource(), and OGRDataSource::SetDriver().

Referenced by OGR_Dr_CreateDataSource().

17.11.2.2 const char * OGR_Dr_GetName (OGRSFDriverH *hDriver*)

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

This function is the same as the C++ method **OGRSFDriver::GetName()** (p. 218).

Parameters:

hDriver handle to the the driver to get the name from.

Returns:

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

References OGR_Dr_GetName().

Referenced by OGR_Dr_GetName().

17.11.2.3 OGRDataSourceH OGR_Dr_Open (OGRSFDriverH *hDriver*, const char * *pszName*, int *bUpdate*)

Attempt to open file with this driver.

This function is the same as the C++ method **OGRSFDriver::Open()** (p. 218).

Parameters:

hDriver handle to the driver that is used to open file.

pszName the name of the file, or data source to try and open.

bUpdate TRUE if update access is required, otherwise FALSE (the default).

Returns:

NULL on error or if the pass name is not supported by this driver, otherwise an handle to an **OGR-DataSource** (p. 92). This **OGRDataSource** (p. 92) should be closed by deleting the object when it is no longer needed.

References OGRDataSource::GetDriver(), OGR_Dr_Open(), and OGRDataSource::SetDriver().

Referenced by OGR_Dr_Open().

17.11.2.4 int OGR_Dr_TestCapability (OGRSFDriverH *hDriver*, const char * *pszCap*)

Test if capability is available.

One of the following data source capability names can be passed into this function, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

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

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

This function is the same as the C++ method **OGRSFDriver::TestCapability()** (p. 219).

Parameters:

hDriver handle to the driver to test the capability against.

pszCap the capability to test.

Returns:

TRUE if capability available otherwise FALSE.

References OGR_Dr_TestCapability().

Referenced by OGR_Dr_TestCapability().

17.11.2.5 OGRLayerH OGR_DS_CreateLayer (OGRDataSourceH *hDS*, const char * *pszName*, OGRSpatialReferenceH *hSpatialRef*, OGRwkbGeometryType *eType*, char ** *papszOptions*)

This function attempts to create a new layer on the data source with the indicated name, coordinate system, geometry type. The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

This function is the same as the C++ method **OGRDataSource::CreateLayer()** (p. 92).

Parameters:

hDS The dataset handle.

pszName the name for the new layer. This should ideally not match any existing layer on the datasource.

hSpatialRef handle to the coordinate system to use for the new layer, or NULL if no coordinate system is available.

eType the geometry type for the layer. Use wkbUnknown if there are no constraints on the types geometry to be written.

papszOptions a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: `http://www.gdal.org/ogr/ogr_formats.html`

Returns:

NULL is returned on failure, or a new **OGRLayer** (p. 160) handle on success.

Example:

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

...

        OGRLayerH *hLayer;
        char     *papszOptions;

        if( OGR_DS_TestCapability( hDS, ODsCCreateLayer ) )
        {
            ...
        }

        papszOptions = CSLSetNameValue( papszOptions, "DIM", "2" );
        hLayer = OGR_DS_CreateLayer( hDS, "NewLayer", NULL, wkbUnknown,
                                     papszOptions );
        CSLDestroy( papszOptions );

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

References OGR_DS_CreateLayer().

Referenced by OGR_DS_CreateLayer().

17.11.2.6 void OGR_DS_Destroy (OGRDataSourceH *hDataSource*)

Closes opened datasource and releases allocated resources.

Parameters:

hDataSource handle to allocated datasource object.

References OGR_DS_Destroy().

Referenced by OGR_DS_Destroy().

17.11.2.7 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. 160) handle representing a results set from the query. Note that this **OGRLayer** (p. 160) is in addition to the layers in the data store and must be destroyed with OGR_DS_ReleaseResultsSet() before the data source is closed (destroyed).

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

This function is the same as the C++ method **OGRDataSource::ExecuteSQL()** (p. 94);

Parameters:

hDS handle to the data source on which the SQL query is executed.

pszSQLCommand the SQL statement to execute.

hSpatialFilter handle to a geometry which represents a spatial filter.

pszDialect allows control of the statement dialect. By default it is assumed to be "generic" SQL, whatever that is.

Returns:

an handle to a **OGRLayer** (p. 160) containing the results of the query. Deallocate with OGR_DS_-ReleaseResultsSet().

References OGR_DS_ExecuteSQL().

Referenced by OGR_DS_ExecuteSQL().

17.11.2.8 OGRLayerH OGR_DS_GetLayer (OGRDataSourceH *hDS*, int *iLayer*)

Fetch a layer by index. The returned layer remains owned by the **OGRDataSource** (p. 92) and should not be deleted by the application.

This function is the same as the C++ method **OGRDataSource::GetLayer()** (p. 95).

Parameters:

hDS handle to the data source from which to get the layer.

iLayer a layer number between 0 and **OGR_DS_GetLayerCount()** (p. 352)-1.

Returns:

an handle to the layer, or NULL if iLayer is out of range or an error occurs.

References OGR_DS_GetLayer().

Referenced by OGR_DS_GetLayer().

17.11.2.9 OGRLayerH OGR_DS_GetLayerByName (OGRDataSourceH *hDS*, const char * *pszLayerName*)

Fetch a layer by name. The returned layer remains owned by the **OGRDataSource** (p. 92) and should not be deleted by the application.

This function is the same as the C++ method **OGRDataSource::GetLayerByName()** (p. 95).

Parameters:

hDS handle to the data source from which to get the layer.

pszLayerName Layer the layer name of the layer to fetch.

Returns:

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

References OGR_DS_GetLayerByName().

Referenced by OGR_DS_GetLayerByName().

17.11.2.10 int OGR_DS_GetLayerCount (OGRDataSourceH *hDS*)

Get the number of layers in this data source.

This function is the same as the C++ method **OGRDataSource::GetLayerCount()** (p. 95).

Parameters:

hDS handle to the data source from which to get the number of layers.

Returns:

layer count.

References OGR_DS_GetLayerCount().

Referenced by OGR_DS_GetLayerCount().

17.11.2.11 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. 217) that this data source was opened with, but it need not be exactly the same string that was used to open the data source. Normally this is a filename.

This function is the same as the C++ method **OGRDataSource::GetName()** (p. 95).

Parameters:

hDS handle to the data source to get the name from.

Returns:

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

References OGR_DS_GetName().

Referenced by OGR_DS_GetName().

17.11.2.12 void OGR_DS_ReleaseResultSet (OGRDataSourceH *hDS*, OGRLayerH *hLayer*)

Release results of **OGR_DS_ExecuteSQL()** (p. 350).

This function should only be used to deallocate OGRLayers resulting from an **OGR_DS_ExecuteSQL()** (p. 350) call on the same **OGRDataSource** (p. 92). Failure to deallocate a results set before destroying the **OGRDataSource** (p. 92) may cause errors.

This function is the same as the C++ method OGRDataSource::ReleaseResultsSet().

Parameters:

hDS an handle to the data source on which was executed an SQL query.

hLayer handle to the result of a previous **OGR_DS_ExecuteSQL()** (p. 350) call.

References OGR_DS_ReleaseResultSet().

Referenced by OGR_DS_ReleaseResultSet().

17.11.2.13 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.

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

This function is the same as the C++ method **OGRDataSource::TestCapability()** (p. 98).

Parameters:

hDS handle to the data source against which to test the capability.

pszCapability the capability to test.

Returns:

TRUE if capability available otherwise FALSE.

References OGR_DS_TestCapability().

Referenced by OGR_DS_TestCapability().

17.11.2.14 OGRFeatureH OGR_F_Clone (OGRFeatureH *hFeat*)

Duplicate feature.

The newly created feature is owned by the caller, and will have it's own reference to the **OGRFeatureDefn** (p. 116).

This function is the same as the C++ method **OGRFeature::Clone()** (p. 102).

Parameters:

hFeat handle to the feature to clone.

Returns:

an handle to the new feature, exactly matching this feature.

References OGR_F_Clone().

Referenced by OGR_F_Clone().

17.11.2.15 OGRFeatureH OGR_F_Create (OGRFeatureDefnH *hDefn*)

Feature factory.

Note that the **OGRFeature** (p. 101) will increment the reference count of it's defining **OGRFeatureDefn** (p. 116). Destruction of the **OGRFeatureDefn** (p. 116) 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. 102).

Parameters:

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

Returns:

an handle to the new feature object with null fields and no geometry.

References OGR_F_Create().

Referenced by OGR_F_Create().

17.11.2.16 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. 103).

Parameters:

hFeat handle to the feature to destroy.

References OGR_F_Destroy().

Referenced by OGR_F_Destroy().

17.11.2.17 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. 103).

Parameters:

hFeat handle to the feature to dump.

fpOut the stream to write to, such as strout.

References OGR_F_DumpReadable().

Referenced by OGR_F_DumpReadable().

17.11.2.18 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. 116), 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. 103).

Parameters:

hFeat handle to one of the feature.

hOtherFeat handle to the other feature to test this one against.

Returns:

TRUE if they are equal, otherwise FALSE.

References OGR_F_Equal().

Referenced by OGR_F_Equal().

17.11.2.19 OGRFeatureDefnH OGR_F_GetDefnRef (OGRFeatureH *hFeat*)

Fetch feature definition.

This function is the same as the C++ method **OGRFeature::GetDefnRef()** (p. 104).

Parameters:

hFeat handle to the feature to get the feature definition from.

Returns:

an handle to the feature definition object on which feature depends.

References OGR_F_GetDefnRef().

Referenced by OGR_F_GetDefnRef().

17.11.2.20 long OGR_F_GetFID (OGRFeatureH *hFeat*)

Get feature identifier.

This function is the same as the C++ method **OGRFeature::GetFID()** (p. 104).

Parameters:

hFeat handle to the feature from which to get the feature identifier.

Returns:

feature id or OGRNullFID if none has been assigned.

References OGR_F_GetFID().

Referenced by OGR_F_GetFID().

17.11.2.21 GByte* OGR_F_GetFieldAsBinary (OGRFeatureH *hFeat*, int *iField*, int * *pnBytes*)

Fetch field value as binary.

Currently this method only works for OFTBinary fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsBinary()** (p. 104).

Parameters:

hFeat handle to the feature that owned the field.

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

pnBytes location to place count of bytes returned.

Returns:

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGR_F_GetFieldAsBinary().

Referenced by OGR_F_GetFieldAsBinary().

17.11.2.22 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. 105).

Parameters:

hFeat handle to the feature that owned the field.

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

int pnYear (including century)

int pnMonth (1-12)

int pnDay (1-31)

int pnHour (0-23)

int pnMinute (0-59)

int pnSecond (0-59)

int pnTZFlag (0=unknown, 1=localtime, 100=GMT, see data model for details)

Returns:

TRUE on success or FALSE on failure.

References OGR_F_GetFieldAsDateTime().

Referenced by OGR_F_GetFieldAsDateTime().

17.11.2.23 double OGR_F_GetFieldAsDouble (OGRFeatureH *hFeat*, int *iField*)

Fetch field value as a double.

OFTString features will be translated using atof(). OFTInteger fields will be cast to double. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsDouble()** (p. 105).

Parameters:

hFeat handle to the feature that owned the field.

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

Returns:

the field value.

References OGR_F_GetFieldAsDouble().

Referenced by OGR_F_GetFieldAsDouble().

17.11.2.24 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. 105).

Parameters:

hFeat handle to the feature that owned the field.

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

pnCount an integer to put the list count (number of doubles) into.

Returns:

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief. If *pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGR_F_GetFieldAsDoubleList().

Referenced by OGR_F_GetFieldAsDoubleList().

17.11.2.25 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. 106).

Parameters:

hFeat handle to the feature that owned the field.

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

Returns:

the field value.

References OGR_F_GetFieldAsInteger().

Referenced by OGR_F_GetFieldAsInteger().

17.11.2.26 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. 106).

Parameters:

hFeat handle to the feature that owned the field.

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

pnCount an integer to put the list count (number of integers) into.

Returns:

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief. If *pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGR_F_GetFieldAsIntegerList().

Referenced by OGR_F_GetFieldAsIntegerList().

17.11.2.27 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. 107).

Parameters:

hFeat handle to the feature that owned the field.

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

Returns:

the field value. This string is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGR_F_GetFieldAsString().

Referenced by OGR_F_GetFieldAsString().

17.11.2.28 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. 328).

This function is the same as the C++ method **OGRFeature::GetFieldAsStringList()** (p. 107).

Parameters:

hFeat handle to the feature that owned the field.

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

Returns:

the field value. This list is internal, and should not be modified, or freed. It's lifetime may be very brief.

References OGR_F_GetFieldAsStringList().

Referenced by OGR_F_GetFieldAsStringList().

17.11.2.29 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. 116).

This function is the same as the C++ method **OGRFeature::GetFieldCount()** (p. 107).

Parameters:

hFeat handle to the feature to get the fields count from.

Returns:

count of fields.

References OGR_F_GetFieldCount().

Referenced by OGR_F_GetFieldCount().

17.11.2.30 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. 108).

Parameters:

hFeat handle to the feature on which the field is found.

i the field to fetch, from 0 to GetFieldCount()-1.

Returns:

an handle to the field definition (from the **OGRFeatureDefn** (p. 116)). This is an internal reference, and should not be deleted or modified.

References OGR_F_GetFieldDefnRef().

Referenced by OGR_F_GetFieldDefnRef().

17.11.2.31 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. 118) method.

This function is the same as the C++ method **OGRFeature::GetFieldIndex()** (p. 108).

Parameters:

hFeat handle to the feature on which the field is found.

pszName the name of the field to search for.

Returns:

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

References OGR_F_GetFieldIndex().

Referenced by OGR_F_GetFieldIndex().

17.11.2.32 OGRGeometryH OGR_F_GetGeometryRef (OGRFeatureH *hFeat*)

Fetch an handle to feature geometry.

This function is the same as the C++ method **OGRFeature::GetGeometryRef()** (p. 108).

Parameters:

hFeat handle to the feature to get geometry from.

Returns:

an handle to internal feature geometry. This object should not be modified.

References OGR_F_GetGeometryRef().

Referenced by OGR_F_GetGeometryRef().

17.11.2.33 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. 109).

Parameters:

hFeat handle to the feature on which field is found.

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

Returns:

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

References OGR_F_GetRawFieldRef().

Referenced by OGR_F_GetRawFieldRef().

17.11.2.34 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. 109).

Parameters:

hFeat handle to the feature to get the style from.

Returns:

a reference to a representation in string format, or NULL if there isn't one.

References OGR_F_GetStyleString().

Referenced by OGR_F_GetStyleString().

17.11.2.35 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. 109).

Parameters:

hFeat handle to the feature on which the field is.

iField the field to test.

Returns:

TRUE if the field has been set, otherwise false.

References OGR_F_IsFieldSet().

Referenced by OGR_F_IsFieldSet().

17.11.2.36 OGRErr OGR_F_SetFID (OGRFeatureH *hFeat*, long *nFID*)

Set the feature identifier.

For specific types of features this operation may fail on illegal features ids. Generally it always succeeds. Feature ids should be greater than or equal to zero, with the exception of OGRNullFID (-1) indicating that the feature id is unknown.

This function is the same as the C++ method **OGRFeature::SetFID()** (p. 109).

Parameters:

hFeat handle to the feature to set the feature id to.

nFID the new feature identifier value to assign.

Returns:

On success OGRERR_NONE, or on failure some other value.

References OGR_F_SetFID().

Referenced by OGR_F_SetFID().

17.11.2.37 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. 113).

Parameters:

hFeat handle to the feature that owned the field.

iField the field to set, from 0 to GetFieldCount()-1.

nBytes the number of bytes in pabyData array.

pabyData the data to apply.

References OGR_F_SetFieldBinary().

Referenced by OGR_F_SetFieldBinary().

17.11.2.38 void OGR_F_SetFieldDateTime (OGRFeatureH *hFeat*, int *iField*, int *nYear*, int *nMonth*, int *nDay*, int *nHour*, int *nMinute*, int *nSecond*, int *nTZFlag*)

Set field to datetime.

This method currently only has an effect for OFTDate, OFTTime and OFTDateTime fields.

Parameters:

hFeat handle to the feature that owned the field.

iField the field to set, from 0 to GetFieldCount()-1.

nYear (including century)

nMonth (1-12)

nDay (1-31)

nHour (0-23)

nMinute (0-59)

nSecond (0-59)

nTZFlag (0=unknown, 1=localtime, 100=GMT, see data model for details)

References OGR_F_SetFieldDateTime().

Referenced by OGR_F_SetFieldDateTime().

17.11.2.39 void OGR_F_SetFieldDouble (OGRFeatureH *hFeat*, int *iField*, double *dfValue*)

Set field to double value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 113).

Parameters:

hFeat handle to the feature that owned the field.

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

dfValue the value to assign.

References OGR_F_SetFieldDouble().

Referenced by OGR_F_SetFieldDouble().

17.11.2.40 void OGR_F_SetFieldDoubleList (OGRFeatureH *hFeat*, int *iField*, int *nCount*, double * *padfValues*)

Set field to list of doubles value.

This function currently on has an effect of OFTRealList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 113).

Parameters:

hFeat handle to the feature that owned the field.

iField the field to set, from 0 to GetFieldCount()-1.

nCount the number of values in the list being assigned.

padfValues the values to assign.

References OGR_F_SetFieldDoubleList().

Referenced by OGR_F_SetFieldDoubleList().

17.11.2.41 void OGR_F_SetFieldInteger (OGRFeatureH *hFeat*, int *iField*, int *nValue*)

Set field to integer value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 113).

Parameters:

hFeat handle to the feature that owned the field.

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

nValue the value to assign.

References OGR_F_SetFieldInteger().

Referenced by OGR_F_SetFieldInteger().

17.11.2.42 void OGR_F_SetFieldIntegerList (OGRFeatureH *hFeat*, int *iField*, int *nCount*, int * *panValues*)

Set field to list of integers value.

This function currently on has an effect of OFTIntegerList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 113).

Parameters:

hFeat handle to the feature that owned the field.

iField the field to set, from 0 to GetFieldCount()-1.

nCount the number of values in the list being assigned.

panValues the values to assign.

References OGR_F_SetFieldIntegerList().

Referenced by OGR_F_SetFieldIntegerList().

17.11.2.43 void OGR_F_SetFieldRaw (OGRFeatureH *hFeat*, int *iField*, OGRField * *psValue*)

Set field.

The passed value **OGRField** (p. 121) 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. 113).

Parameters:

hFeat handle to the feature that owned the field.

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

psValue handle on the value to assign.

References OGR_F_SetFieldRaw().

Referenced by OGR_F_SetFieldRaw().

17.11.2.44 void OGR_F_SetFieldString (OGRFeatureH *hFeat*, int *iField*, const char * *pszValue*)

Set field to string value.

OFTInteger fields will be set based on an atoi() conversion of the string. OFTReal fields will be set based on an atof() conversion of the string. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 113).

Parameters:

hFeat handle to the feature that owned the field.

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

pszValue the value to assign.

References OGR_F_SetFieldString().

Referenced by OGR_F_SetFieldString().

17.11.2.45 void OGR_F_SetFieldStringList (OGRFeatureH *hFeat*, int *iField*, char ** *papszValues*)

Set field to list of strings value.

This function currently on has an effect of OFTStringList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. 113).

Parameters:

hFeat handle to the feature that owned the field.

iField the field to set, from 0 to GetFieldCount()-1.

papszValues the values to assign.

References OGR_F_SetFieldStringList().

Referenced by OGR_F_SetFieldStringList().

17.11.2.46 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. 116). 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. 113).

Parameters:

hFeat handle to the feature to set to.

hOtherFeat handle to the feature from which geometry, and field values will be copied.

bForgiving TRUE if the operation should continue despite lacking output fields matching some of the source fields.

Returns:

OGRERR_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References OGR_F_SetFrom().

Referenced by OGR_F_SetFrom().

17.11.2.47 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. 113).

Parameters:

hFeat handle to the feature on which new geometry is applied to.

hGeom handle to the new geometry to apply to feature.

Returns:

OGRERR_NONE if successful, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. 116) (checking not yet implemented).

References OGR_F_SetGeometry().

Referenced by OGR_F_SetGeometry().

17.11.2.48 OGRErr OGR_F_SetGeometryDirectly (OGRFeatureH *hFeat*, OGRGeometryH *hGeom*)

Set feature geometry.

This function updates the features geometry, and operate exactly as SetGeometry(), except that this function assumes ownership of the passed geometry.

This function is the same as the C++ method **OGRFeature::SetGeometryDirectly** (p. 114).

Parameters:

hFeat handle to the feature on which to apply the geometry.

hGeom handle to the new geometry to apply to feature.

Returns:

OGRERR_NONE if successful, or OGR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. 116) (checking not yet implemented).

References OGR_F_SetGeometryDirectly().

Referenced by OGR_F_SetGeometryDirectly().

17.11.2.49 void OGR_F_SetStyleString (OGRFeatureH *hFeat*, const char * *pszStyle*)

Set feature style string. This method operate exactly as **OGR_F_SetStyleStringDirectly()** (p. 366) 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. 114).

Parameters:

hFeat handle to the feature to set style to.

pszStyle the style string to apply to this feature, cannot be NULL.

References OGR_F_SetStyleString().

Referenced by OGR_F_SetStyleString().

17.11.2.50 void OGR_F_SetStyleStringDirectly (OGRFeatureH *hFeat*, char * *pszStyle*)

Set feature style string. This method operate exactly as **OGR_F_SetStyleString()** (p. 366) except that it assumes ownership of the passed string.

This function is the same as the C++ method **OGRFeature::SetStyleStringDirectly()** (p. 114).

Parameters:

hFeat handle to the feature to set style to.

pszStyle the style string to apply to this feature, cannot be NULL.

References OGR_F_SetStyleStringDirectly().

Referenced by OGR_F_SetStyleStringDirectly().

17.11.2.51 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. 115).

Parameters:

hFeat handle to the feature on which the field is.

iField the field to unset.

References OGR_F_UnsetField().

Referenced by OGR_F_UnsetField().

17.11.2.52 void OGR_FD_AddFieldDefn (OGRFeatureDefnH *hDefn*, OGRFieldDefnH *hNewField*)

Add a new field definition to the passed feature definition.

This function should only be called while there are no **OGRFeature** (p. 101) objects in existance based on this **OGRFeatureDefn** (p. 116). The **OGRFieldDefn** (p. 122) passed in is copied, and remains the responsibility of the caller.

This function is the same as the C++ method **OGRFeatureDefn::AddFieldDefn** (p. 117).

Parameters:

hDefn handle to the feature definition to add the field definition to.

hNewField handle to the new field definition.

References OGR_FD_AddFieldDefn().

Referenced by OGR_FD_AddFieldDefn().

17.11.2.53 OGRFeatureDefnH OGR_FD_Create (const char * *pszName*)

Create a new feature definition object to held the field definitions.

The **OGRFeatureDefn** (p. 116) 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. 116).

Parameters:

pszName the name to be assigned to this layer/class. It does not need to be unique.

Returns:

handle to the newly created feature definition.

References OGR_FD_Create().

Referenced by OGR_FD_Create().

17.11.2.54 int OGR_FD_Dereference (OGRFeatureDefnH *hDefn*)

Decrements the reference count by one.

This function is the same as the C++ method **OGRFeatureDefn::Dereference()** (p. 117).

Parameters:

hDefn handle to the feature definition on witch **OGRFeature** (p. 101) are based on.

Returns:

the updated reference count.

References OGR_FD_Dereference().

Referenced by OGR_FD_Dereference().

17.11.2.55 void OGR_FD_Destroy (OGRFeatureDefnH *hDefn*)

Destroy a feature definition object and release all memory associated with it.

This function is the same as the C++ method OGRFeatureDefn::∼OGRFeatureDefn().

Parameters:

hDefn handle to the feature definition to be destroyed.

References OGR_FD_Destroy().

Referenced by OGR_FD_Destroy().

17.11.2.56 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. 117).

Parameters:

hDefn handle to the feature definition to get the fields count from.

Returns:

count of fields.

References OGR_FD_GetFieldCount().

Referenced by OGR_FD_GetFieldCount().

17.11.2.57 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. 118).

Parameters:

hDefn handle to the feature definition to get the field definition from.

iField the field to fetch, between 0 and GetFieldCount()-1.

Returns:

an handle to an internal field definition object. This object should not be modified or freed by the application.

References OGR_FD_GetFieldDefn().

Referenced by OGR_FD_GetFieldDefn().

17.11.2.58 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. 118).

Parameters:

hDefn handle to the feature definition to get field index from.

pszFieldName the field name to search for.

Returns:

the field index, or -1 if no match found.

References OGR_FD_GetFieldIndex().

Referenced by OGR_FD_GetFieldIndex().

17.11.2.59 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. 118).

Parameters:

hDefn handle to the feature definition to get the geometry type from.

Returns:

the base type for all geometry related to this definition.

References OGR_FD_GetGeomType().

Referenced by OGR_FD_GetGeomType().

17.11.2.60 const char* OGR_FD_GetName (OGRFeatureDefnH *hDefn*)

Get name of the **OGRFeatureDefn** (p. 116) passed as an argument.

This function is the same as the C++ method **OGRFeatureDefn::GetName()** (p. 119).

Parameters:

hDefn handle to the feature definition to get the name from.

Returns:

the name. This name is internal and should not be modified, or freed.

References OGR_FD_GetName().

Referenced by OGR_FD_GetName().

17.11.2.61 int OGR_FD_GetReferenceCount (OGRFeatureDefnH *hDefn*)

Fetch current reference count.

This function is the same as the C++ method **OGRFeatureDefn::GetReferenceCount()** (p. 119).

Parameters:

hDefn hanlde to the feature definition on witch **OGRFeature** (p. 101) are based on.

Returns:

the current reference count.

References OGR_FD_GetReferenceCount().

Referenced by OGR_FD_GetReferenceCount().

17.11.2.62 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. 101) objects referencing this definition.

This function is the same as the C++ method **OGRFeatureDefn::Reference()** (p. 119).

Parameters:

hDefn handle to the feature definition on witch **OGRFeature** (p. 101) are based on.

Returns:

the updated reference count.

References OGR_FD_Reference().

Referenced by OGR_FD_Reference().

17.11.2.63 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. 119).

Parameters:

hDefn handle to the feature definition to be released.

References OGR_FD_Release().

Referenced by OGR_FD_Release().

17.11.2.64 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. 119).

Parameters:

hDefn handle to the layer or feature definition to set the geometry type to.

eType the new type to assign.

References OGR_FD_SetGeomType().

Referenced by OGR_FD_SetGeomType().

17.11.2.65 OGRFieldDefnH OGR_Fld_Create (const char * *pszName*, OGRFieldType *eType*)

Create a new field definition.

This function is the same as the CPP method **OGRFieldDefn::OGRFieldDefn()** (p. 122).

Parameters:

pszName the name of the new field definition.

eType the type of the new field definition.

Returns:

handle to the new field definition.

References OGR_Fld_Create().

Referenced by OGR_Fld_Create().

17.11.2.66 void OGR_Fld_Destroy (OGRFieldDefnH *hDefn*)

Destroy a field definition.

Parameters:

hDefn handle to the field definition to destroy.

References OGR_Fld_Destroy().

Referenced by OGR_Fld_Destroy().

17.11.2.67 OGRJustification OGR_Fld_GetJustify (OGRFieldDefnH *hDefn*)

Get the justification for this field.

This function is the same as the CPP method **OGRFieldDefn::GetJustify()** (p. 123).

Parameters:

hDefn handle to the field definition to get justification from.

Returns:

the justification.

References OGR_Fld_GetJustify().

Referenced by OGR_Fld_GetJustify().

17.11.2.68 const char* OGR_Fld_GetNameRef (OGRFieldDefnH *hDefn*)

Fetch name of this field.

This function is the same as the CPP method **OGRFieldDefn::GetNameRef()** (p. 123).

Parameters:

hDefn handle to the field definition.

Returns:

the name of the field definition.

References OGR_Fld_GetNameRef().

Referenced by OGR_Fld_GetNameRef().

17.11.2.69 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. 123).

Parameters:

hDefn handle to the field definition to get precision from.

Returns:

the precision.

References OGR_Fld_GetPrecision().

Referenced by OGR_Fld_GetPrecision().

17.11.2.70 OGRFieldType OGR_Fld_GetType (OGRFieldDefnH *hDefn*)

Fetch type of this field.

This function is the same as the CPP method **OGRFieldDefn::GetType()** (p. 124).

Parameters:

hDefn handle to the field definition to get type from.

Returns:

field type.

References OGR_Fld_GetType().

Referenced by OGR_Fld_GetType().

17.11.2.71 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. 124).

Parameters:

hDefn handle to the field definition to get width from.

Returns:

the width, zero means no specified width.

References OGR_Fld_GetWidth().

Referenced by OGR_Fld_GetWidth().

17.11.2.72 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. 124).

Parameters:

hDefn handle to the field definition to set to.

pszNameIn the new name to assign.

eTypeIn the new type (one of the OFT values like OFTInteger).

nWidthIn the preferred formatting width. Defaults to zero indicating undefined.

nPrecisionIn number of decimals places for formatting, defaults to zero indicating undefined.

eJustifyIn the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References OGR_Fld_Set().

Referenced by OGR_Fld_Set().

17.11.2.73 void OGR_Fld_SetJustify (OGRFieldDefnH *hDefn*, OGRJustification *eJustify*)

Set the justification for this field.

This function is the same as the CPP method **OGRFieldDefn::SetJustify()** (p. 125).

Parameters:

hDefn handle to the field definition to set justification to.

eJustify the new justification.

References OGR_Fld_SetJustify().

Referenced by OGR_Fld_SetJustify().

17.11.2.74 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. 125).

Parameters:

hDefn handle to the field definition to apply the new name to.

pszName the new name to apply.

References OGR_Fld_SetName().

Referenced by OGR_Fld_SetName().

17.11.2.75 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. 125).

Parameters:

hDefn handle to the field definition to set precision to.

nPrecision the new precision.

References OGR_Fld_SetPrecision().

Referenced by OGR_Fld_SetPrecision().

17.11.2.76 void OGR_Fld_SetType (OGRFieldDefnH *hDefn*, OGRFieldType *eType*)

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. 122) that is already part of an **OGRFeatureDefn** (p. 116).

This function is the same as the CPP method **OGRFieldDefn::SetType()** (p. 125).

Parameters:

hDefn handle to the field definition to set type to.

eType the new field type.

References OGR_Fld_SetType().

Referenced by OGR_Fld_SetType().

17.11.2.77 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. 126).

Parameters:

hDefn handle to the field definition to set width to.

nNewWidth the new width.

References OGR_Fld_SetWidth().

Referenced by OGR_Fld_SetWidth().

17.11.2.78 OGRErr OGR_G_AddGeometry (OGRGeometryH *hGeom*, OGRGeometryH *hNewSubGeom*)

Add a geometry to a geometry container.

Some subclasses of **OGRGeometryCollection** (p. 144) 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. 145).

Parameters:

hGeom existing geometry container.

hNewSubGeom geometry to add to the container.

Returns:

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of existing geometry.

References OGRLineString::getGeometryType(), wkbGeometryCollection, wkbLineString, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbPolygon, and OGRLinearRing::WkbSize().

17.11.2.79 OGRErr OGR_G_AddGeometryDirectly (OGRGeometryH *hGeom*, OGRGeometryH *hNewSubGeom*)

Add a geometry directly to an existing geometry container.

Some subclasses of **OGRGeometryCollection** (p. 144) 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. 145).

There is no SFCOM analog to this method.

Parameters:

hGeom existing geometry.

hNewSubGeom geometry to add to the existing geometry.

Returns:

OGRERR_NONE if successful, or OGRERR_UNSUPPORTED_GEOMETRY_TYPE if the geometry type is illegal for the type of geometry container.

References OGRLineString::getGeometryType(), wkbGeometryCollection, wkbLineString, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbPolygon, and OGRLinearRing::WkbSize().

17.11.2.80 void OGR_G_AddPoint (OGRGeometryH *hGeom*, double *dfX*, double *dfY*, double *dfZ*)

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

Parameters:

hGeom handle to the geometry to add a point to.

dfX x coordinate of point to add.

dfY y coordinate of point to add.

dfZ z coordinate of point to add.

References wkbLineString, and wkbPoint.

17.11.2.81 void OGR_G_AddPoint_2D (OGRGeometryH *hGeom*, double *dfX*, double *dfY*)

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

Parameters:

hGeom handle to the geometry to add a point to.

dfX x coordinate of point to add.

dfY y coordinate of point to add.

References wkbLineString, and wkbPoint.

17.11.2.82 void OGR_G_AssignSpatialReference (OGRGeometryH *hGeom*, OGRSpatialReferenceH *hSRS*)

Assign spatial reference to this object. Any existing spatial reference is replaced, but under no circumstances does this result in the object being reprojected. It is just changing the interpretation of the existing geometry. Note that assigning a spatial reference increments the reference count on the **OGRSpatialReference** (p. 224), but does not copy it.

This is similar to the SFCOM IGeometry::put_SpatialReference() method.

This function is the same as the CPP method **OGRGeometry::assignSpatialReference** (p. 128).

Parameters:

hGeom handle on the geometry to apply the new spatial reference system.

hSRS handle on the new spatial reference system to apply.

References OGR_G_AssignSpatialReference().

Referenced by OGR_G_AssignSpatialReference().

17.11.2.83 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. 129).

Parameters:

hGeom handle on the geometry to clone from.

Returns:

an handle on the copy of the geometry with the spatial reference system as the original.

References OGR_G_Clone().

Referenced by OGR_G_Clone().

17.11.2.84 OGRErr OGR_G_CreateFromWkb (unsigned char * *pabyData*, OGRSpatialReferenceH *hSRS*, OGRGeometryH * *phGeometry*, int *nBytes*)

Create a geometry object of the appropriate type from it's well known binary representation.

Note that if nBytes is passed as zero, no checking can be done on whether the pabyData is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **OGR_G_WkbSize()** (p. 390) on the returned geometry to establish the number of bytes it required in WKB format.

The **OGRGeometryFactory::createFromWkb()** (p. 155) CPP method is the same as this function.

Parameters:

pabyData pointer to the input BLOB data.

hSRS handle to the spatial reference to be assigned to the created geometry object. This may be NULL.

phGeometry the newly created geometry object will be assigned to the indicated handle on return. This will be NULL in case of failure.

nBytes the number of bytes of data available in pabyData, or -1 if it is not known, but assumed to be sufficient.

Returns:

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

References OGRGeometryFactory::createFromWkb(), and OGR_G_CreateFromWkb().

Referenced by OGR_G_CreateFromWkb().

17.11.2.85 OGRErr OGR_G_CreateFromWkt (char ** *ppszData*, OGRSpatialReferenceH *hSRS*, OGRGeometryH * *phGeometry*)

Create a geometry object of the appropriate type from it's well known text representation.

The **OGRGeometryFactory::createFromWkt** (p. 156) CPP method is the same as this function.

Parameters:

ppszData input zero terminated string containing well known text representation of the geometry to be created. The pointer is updated to point just beyond that last character consumed.

hSRS handle to the spatial reference to be assigned to the created geometry object. This may be NULL.

phGeometry the newly created geometry object will be assigned to the indicated handle on return. This will be NULL if the method fails.

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(), and OGR_G_CreateFromWkt().

Referenced by OGR_G_CreateFromWkt().

17.11.2.86 OGRGeometryH OGR_G_CreateGeometry (OGRwkbGeometryType *eGeometryType*)

Create an empty geometry of desired type.

This is equivelent 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. 156).

Parameters:

eGeometryType the type code of the geometry to be created.

Returns:

handle to the newly create geometry or NULL on failure.

References OGRGeometryFactory::createGeometry(), and OGR_G_CreateGeometry().

Referenced by OGR_G_CreateGeometry().

17.11.2.87 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. 157).

Parameters:

hGeom handle to the geometry to delete.

References OGRGeometryFactory::destroyGeometry(), and OGR_G_DestroyGeometry().

Referenced by OGR_G_DestroyGeometry().

17.11.2.88 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. 132).

Parameters:

hGeom handle on the geometry to dump.

fp the text file to write the geometry to.

pszPrefix the prefix to put on each line of output.

References OGR_G_DumpReadable().

Referenced by OGR_G_DumpReadable().

17.11.2.89 void OGR_G_Empty (OGRGeometryH *hGeom*)

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This function relates to the SFCOM IGeometry::Empty() method.

This function is the same as the CPP method **OGRGeometry::empty()** (p. 132).

Parameters:

hGeom handle on the geometry to empty.

References OGR_G_Empty().

Referenced by OGR_G_Empty().

17.11.2.90 int OGR_G_Equals (OGRGeometryH *hGeom*, OGRGeometryH *hOther*)

Returns two if two geometries are equivalent.

This function is the same as the CPP method **OGRGeometry::Equals()** (p. 132) method.

Parameters:

hGeom handle on the first geometry.

hOther handle on the other geometry to test against.

Returns:

TRUE if equivalent or FALSE otherwise.

References OGR_G_Equals().

Referenced by OGR_G_Equals().

17.11.2.91 OGRErr OGR_G_ExportToWkb (OGRGeometryH *hGeom*, OGRwkbByteOrder *eOrder*, unsigned char * *pabyDstBuffer*)

Convert a geometry into well known binary format.

This function relates to the SFCOM IWks::ExportToWKB() method.

This function is the same as the CPP method **OGRGeometry::exportToWkb()** (p. 133).

Parameters:

hGeom handle on the geometry to convert to a well know binary data from.

eOrder One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.

pabyDstBuffer a buffer into which the binary representation is written. This buffer must be at least **OGR_G_WkbSize()** (p. 390) byte in size.

Returns:

Currently OGRERR_NONE is always returned.

References OGR_G_ExportToWkb().

Referenced by OGR_G_ExportToWkb().

17.11.2.92 OGRErr OGR_G_ExportToWkt (OGRGeometryH *hGeom*, char ** *ppszSrcText*)

Convert a geometry into well known text format.

This function relates to the SFCOM IWks::ExportToWKT() method.

This function is the same as the CPP method **OGRGeometry::exportToWkt()** (p. 134).

Parameters:

hGeom handle on the geometry to convert to a text format from.

ppszSrcText a text buffer is allocated by the program, and assigned to the passed pointer.

Returns:

Currently OGRERR_NONE is always returned.

References OGR_G_ExportToWkt().

Referenced by OGR_G_ExportToWkt().

17.11.2.93 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. 134).

Parameters:

hGeom handle on the geometry to convert.

References OGR_G_FlattenTo2D().

Referenced by OGR_G_FlattenTo2D().

17.11.2.94 double OGR_G_GetArea (OGRGeometryH *hGeom*)

Compute geometry area.

Computes the area for an **OGRLinearRing** (p. 171), **OGRPolygon** (p. 206) or **OGRMultiPolygon** (p. 194). Undefined for all other geometry types (returns zero).

This function utilizes the C++ get_Area() methods such as **OGRPolygon::get_Area()** (p. 209).

Parameters:

hGeom the geometry to operate on.

Returns:

the area or 0.0 for unsupported geometry types.

References wkbGeometryCollection, wkbLinearRing, wkbLineString, wkbMultiPolygon, and wkbPolygon.

17.11.2.95 int OGR_G_GetCoordinateDimension (OGRGeometryH *hGeom*)

Get the dimension of the coordinates in this geometry.

This function corresponds to the SFCOM IGeometry::GetDimension() method.

This function is the same as the CPP method **OGRGeometry::getCoordinateDimension()** (p. 134).

Parameters:

hGeom handle on the geometry to get the dimension of the coordinates from.

Returns:

in practice this always returns 2 indicating that coordinates are specified within a two dimensional space.

References OGR_G_GetCoordinateDimension().

Referenced by OGR_G_GetCoordinateDimension().

17.11.2.96 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. 381) function).

This function is the same as the CPP method **OGRGeometry::getDimension()** (p. 135).

Parameters:

hGeom handle on the geometry to get the dimension from.

Returns:

0 for points, 1 for lines and 2 for surfaces.

References OGR_G_GetDimension().

Referenced by OGR_G_GetDimension().

17.11.2.97 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. 135).

Parameters:

hGeom handle of the geometry to get envelope from.

psEnvelope the structure in which to place the results.

References OGR_G_GetEnvelope().

Referenced by OGR_G_GetEnvelope().

17.11.2.98 int OGR_G_GetGeometryCount (OGRGeometryH *hGeom*)

Fetch the number of elements in a geometry or number of geometries in container.

Parameters:

hGeom single geometry or geometry container from which to get the number of elements.

Returns:

the number of elements.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

17.11.2.99 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. 135).

Parameters:

hGeom handle on the geometry to get name from.

Returns:

name used for this geometry type in well known text format.

References OGR_G_GetGeometryName().

Referenced by OGR_G_GetGeometryName().

17.11.2.100 OGRGeometryH OGR_G_GetGeometryRef (OGRGeometryH *hGeom*, int *iSubGeom*)

Fetch geometry from a geometry container.

This function returns an handle to a geometry within the container. The returned geometry remains owned by the container, and should not be modified. The handle is only valid untill the next change to the geometry container. Use **OGR_G_Clone()** (p. 377) 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. 149).

Parameters:

hGeom handle to the geometry container from which to get a geometry from.

iSubGeom the index of the geometry to fetch, between 0 and getNumGeometries() - 1.

Returns:

handle to the requested geometry.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

17.11.2.101 OGRwkbGeometryType OGR_G_GetGeometryType (OGRGeometryH *hGeom*)

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This function is the same as the CPP method **OGRGeometry::getGeometryType()** (p. 136).

Parameters:

hGeom handle on the geometry to get type from.

Returns:

the geometry type code.

References OGR_G_GetGeometryType().

Referenced by OGR_G_GetGeometryType().

17.11.2.102 void OGR_G_GetPoint (OGRGeometryH *hGeom*, int *i*, double * *pdfX*, double * *pdfY*, double * *pdfZ*)

Fetch a point in line string or a point geometry.

Parameters:

hGeom handle to the geometry from which to get the coordinates.

i the vertex to fetch, from 0 to getNumPoints()-1, zero for a point.

pdfX value of x coordinate.

pdfY value of y coordinate.

pdfZ value of z coordinate.

References wkbLineString, and wkbPoint.

17.11.2.103 int OGR_G_GetPointCount (OGRGeometryH *hGeom*)

Fetch number of points from a geometry.

Parameters:

hGeom handle to the geometry from which to get the number of points.

Returns:

the number of points.

References OGRLineString::getNumPoints(), wkbLineString, and wkbPoint.

17.11.2.104 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. 136).

Parameters:

hGeom handle on the geometry to get spatial reference from.

Returns:

a reference to the spatial reference geometry.

References OGR_G_GetSpatialReference().

Referenced by OGR_G_GetSpatialReference().

17.11.2.105 double OGR_G_GetX (OGRGeometryH *hGeom*, int *i*)

Fetch the x coordinate of a point from a geometry.

Parameters:

hGeom handle to the geometry from which to get the x coordinate.

i point to get the x coordinate.

Returns:

the X coordinate of this point.

References wkbLineString, and wkbPoint.

17.11.2.106 double OGR_G_GetY (OGRGeometryH *hGeom*, int *i*)

Fetch the x coordinate of a point from a geometry.

Parameters:

hGeom handle to the geometry from which to get the y coordinate.

i point to get the Y coordinate.

Returns:

the Y coordinate of this point.

References wkbLineString, and wkbPoint.

17.11.2.107 double OGR_G_GetZ (OGRGeometryH *hGeom*, int *i*)

Fetch the z coordinate of a point from a geometry.

Parameters:

hGeom handle to the geometry from which to get the Z coordinate.

i point to get the Z coordinate.

Returns:

the Z coordinate of this point.

References wkbLineString, and wkbPoint.

17.11.2.108 OGRErr OGR_G_ImportFromWkb (OGRGeometryH *hGeom*, unsigned char * *pabyData*, int *nSize*)

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type.

This function relates to the SFCOM IWks::ImportFromWKB() method.

This function is the same as the CPP method **OGRGeometry::importFromWkb()** (p. 136).

Parameters:

hGeom handle on the geometry to assign the well know binary data to.

pabyData the binary input data.

nSize the size of pabyData in bytes, or zero if not known.

Returns:

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_-UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGR_G_ImportFromWkb().

Referenced by OGR_G_ImportFromWkb().

17.11.2.109 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. 137).

Parameters:

hGeom handle on the geometry to assign well know text data to.

ppszSrcText pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

Returns:

OGRERR_NONE if all goes well, otherwise any of OGRERR_NOT_ENOUGH_DATA, OGRERR_-UNSUPPORTED_GEOMETRY_TYPE, or OGRERR_CORRUPT_DATA may be returned.

References OGR_G_ImportFromWkt().

Referenced by OGR_G_ImportFromWkt().

17.11.2.110 int OGR_G_Intersects (OGRGeometryH *hGeom*, OGRGeometryH *hOtherGeom*)

Do these features intersect?

Currently this is not implemented in a rigerous fashion, and generally just tests whether the envelopes of the two features intersect. Eventually this will be made rigerous.

This function is the same as the CPP method **OGRGeometry::Intersects** (p. 138).

Parameters:

hGeom handle on the first geometry.

hOtherGeom handle on the other geometry to test against.

Returns:

TRUE if the geometries intersect, otherwise FALSE.

References OGR_G_Intersects().

Referenced by OGR_G_Intersects().

17.11.2.111 int OGR_G_IsEmpty (OGRGeometryH *hGeom*)

Test if the geometry is empty

This method is the same as the CPP method **OGRGeometry::IsEmpty()** (p. 138).

Returns:

TRUE if the geometry has no points, otherwise FALSE.

References OGR_G_IsEmpty().

Referenced by OGR_G_IsEmpty().

17.11.2.112 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 method relates to the SFCOM IGeometry::IsSimple() method.

NOTE: This method is hardcoded to return TRUE at this time.

Returns:

TRUE if object is simple, otherwise FALSE.

References OGR_G_IsSimple().

Referenced by OGR_G_IsSimple().

17.11.2.113 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. 151).

Parameters:

hGeom the existing geometry to delete from.

iGeom the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.

bDelete if TRUE the geometry will be destroyed, otherwise it will not. The default is TRUE as the existing geometry is considered to own the geometries in it.

Returns:

OGRERR_NONE if successful, or OGRERR_FAILURE if the index is out of range.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

17.11.2.114 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. 140).

Parameters:

hGeom handle on the geometry to segmentize

dfMaxLength the maximum distance between 2 points after segmentization

References OGR_G_Segmentize().

Referenced by OGR_G_Segmentize().

17.11.2.115 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 accomodate the request.

Parameters:

hGeom handle to the geometry to add a vertex to.

i the index of the vertex to assign (zero based) or zero for a point.

dfX input X coordinate to assign.

dfY input Y coordinate to assign.

dfZ input Z coordinate to assign (defaults to zero).

References wkbLineString, and wkbPoint.

17.11.2.116 void OGR_G_SetPoint_2D (OGRGeometryH *hGeom*, int *i*, double *dfX*, double *dfY*)

Set the location of a vertex in a point or linestring geometry.

If iPoint is larger than the number of existing points in the linestring, the point count will be increased to accomodate the request.

Parameters:

hGeom handle to the geometry to add a vertex to.

i the index of the vertex to assign (zero based) or zero for a point.

dfX input X coordinate to assign.

dfY input Y coordinate to assign.

References wkbLineString, and wkbPoint.

17.11.2.117 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. 88) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. 224) of the **OGRCoordinateTransformation** (p. 88) will be assigned to the geometry.

This function is the same as the CPP method **OGRGeometry::transform** (p. 141).

Parameters:

hGeom handle on the geometry to apply the transform to.

hTransform handle on the transformation to apply.

Returns:

OGRERR_NONE on success or an error code.

References OGR_G_Transform().

Referenced by OGR_G_Transform().

17.11.2.118 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. 88) object it is significantly more expensive to use this function to transform many geometries than it is to create the **OGRCoordinateTransformation** (p. 88) in advance, and call transform() with that transformation. This function exists primarily for convenience when only transforming a single geometry.

This function is the same as the CPP method **OGRGeometry::transformTo** (p. 142).

Parameters:

hGeom handle on the geometry to apply the transform to.

hSRS handle on the spatial reference system to apply.

Returns:

OGRERR_NONE on success, or an error code.

References OGR_G_TransformTo().

Referenced by OGR_G_TransformTo().

17.11.2.119 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. 143).

Parameters:

hGeom handle on the geometry to get the binary size from.

Returns:

size of binary representation in bytes.

References OGR_G_WkbSize().

Referenced by OGR_G_WkbSize().

17.11.2.120 const char* OGR_GetFieldTypeName (OGRFieldType *eType*)

Fetch human readable name for a field type.

This function is the same as the CPP method **OGRFieldDefn::GetFieldTypeName()** (p. 123).

Parameters:

eType the field type to get name for.

Returns:

the name.

References OGRFieldDefn::GetFieldTypeName(), and OGR_GetFieldTypeName().

Referenced by OGR_GetFieldTypeName().

17.11.2.121 OGRErr OGR_L_CommitTransaction (OGRLayerH *hLayer*)

For datasources which support transactions, CommitTransaction commits a transaction. If no transaction is active, or the commit fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_NONE.

This function is the same as the C++ method OGRLayer::CommitTransaction().

Parameters:

hLayer handle to the layer

Returns:

OGRERR_NONE on success.

References OGR_L_CommitTransaction().

Referenced by OGR_L_CommitTransaction().

17.11.2.122 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. 160).

Parameters:

hLayer handle to the layer to write the feature to.

hFeat the handle of the feature to write to disk.

Returns:

OGRERR_NONE on success.

References OGR_L_CreateFeature().

Referenced by OGR_L_CreateFeature().

17.11.2.123 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. 116) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. 116) used by a layer directly.

This function is the same as the C++ method **OGRLayer::CreateField()** (p. 161).

Parameters:

hLayer handle to the layer to write the field definition.

hField handle of the field definition to write to disk.

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

Returns:

OGRERR_NONE on success.

References OGR_L_CreateField().

Referenced by OGR_L_CreateField().

17.11.2.124 OGRErr OGR_L_DeleteFeature (OGRLayerH *hLayer*, long *nFID*)

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The **OGR_L_TestCapability()** (p. 398) 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. 161).

Parameters:

hLayer handle to the layer

nFID the feature id to be deleted from the layer

Returns:

OGRERR_NONE on success.

References OGR_L_DeleteFeature().

Referenced by OGR_L_DeleteFeature().

17.11.2.125 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. 392) without setting a spatial filter.

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

This function is the same as the C++ method **OGRLayer::GetExtent()** (p. 162).

Parameters:

hLayer handle to the layer from which to get extent.

psExtent the structure in which the extent value will be returned.

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

Returns:

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

References OGR_L_GetExtent().

Referenced by OGR_L_GetExtent().

17.11.2.126 OGRFeatureH OGR_L_GetFeature (OGRLayerH *hLayer*, long *nFeatureId*)

Fetch a feature by it's identifier.

This function will attempt to read the identified feature. The nFID value cannot be OGRNullFID. Success or failure of this operation is unaffected by the spatial or attribute filters.

If this function returns a non-NULL feature, it is guaranteed that it's feature id (**OGR_F_GetFID()** (p. 355)) 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. 393); however, the call should always work if the feature exists as a fallback implementation just scans all the features in the layer looking for the desired feature.

Sequential reads are generally considered interrupted by a **OGR_L_GetFeature()** (p. 393) call.

This function is the same as the C++ method **OGRLayer::GetFeature( )** (p. 162).

Parameters:

hLayer handle to the layer that owned the feature.

nFeatureId the feature id of the feature to read.

Returns:

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

References OGR_L_GetFeature().

Referenced by OGR_L_GetFeature().

17.11.2.127 int OGR_L_GetFeatureCount (OGRLayerH *hLayer*, int *bForce*)

Fetch the feature count in this layer.

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

The returned count takes the spatial filter into account.

This function is the same as the CPP **OGRLayer::GetFeatureCount()** (p. 163).

Parameters:

hLayer handle to the layer that owned the features.

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

Returns:

feature count, -1 if count not known.

References OGR_L_GetFeatureCount().

Referenced by OGR_L_GetFeatureCount().

17.11.2.128 OGRFeatureDefnH OGR_L_GetLayerDefn (OGRLayerH *hLayer*)

Fetch the schema information for this layer.

The returned handle to the **OGRFeatureDefn** (p. 116) is owned by the **OGRLayer** (p. 160), 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. 164).

Parameters:

hLayer handle to the layer to get the schema information.

Returns:

an handle to the feature definition.

References OGR_L_GetLayerDefn().

Referenced by OGR_L_GetLayerDefn().

17.11.2.129 OGRFeatureH OGR_L_GetNextFeature (OGRLayerH *hLayer*)

Fetch the next available feature from this layer. The returned feature becomes the responsiblity of the caller to delete. It is critical that all features associated with an **OGRLayer** (p. 160) (more specifically an **OGRFeatureDefn** (p. 116)) 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. 395) function can be used to start at the beginning again. Random reading, writing and spatial filtering will be added to the **OGRLayer** (p. 160) in the future.

This function is the same as the C++ method **OGRLayer::GetNextFeature()** (p. 164).

Parameters:

hLayer handle to the layer from which feature are read.

Returns:

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

References OGR_L_GetNextFeature().

Referenced by OGR_L_GetNextFeature().

17.11.2.130 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. 165).

Parameters:

hLayer handle to the layer to get the spatial filter from.

Returns:

an handle to the spatial filter geometry.

References OGR_L_GetSpatialFilter().

Referenced by OGR_L_GetSpatialFilter().

17.11.2.131 OGRSpatialReferenceH OGR_L_GetSpatialRef (OGRLayerH *hLayer*)

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. 160) and should not be modified or freed by the application.

This function is the same as the C++ method **OGRLayer::GetSpatialRef()** (p. 165).

Parameters:

hLayer handle to the layer to get the spatial reference from.

Returns:

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

References OGR_L_GetSpatialRef().

Referenced by OGR_L_GetSpatialRef().

17.11.2.132 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. 166).

Parameters:

hLayer handle to the layer on which features are read.

References OGR_L_ResetReading().

Referenced by OGR_L_ResetReading().

17.11.2.133 OGRErr OGR_L_RollbackTransaction (OGRLayerH *hLayer*)

For datasources which support transactions, RollbackTransaction will roll back a datasource to its state before the start of the current transaction. If no transaction is active, or the rollback fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_NONE.

This function is the same as the C++ method OGRLayer::RollbackTransaction().

Parameters:

hLayer handle to the layer

Returns:

OGRERR_NONE on success.

References OGR_L_RollbackTransaction().

Referenced by OGR_L_RollbackTransaction().

17.11.2.134 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. 394) 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. 395)).

This function is the same as the C++ method **OGRLayer::SetAttributeFilter()** (p. 166).

Parameters:

hLayer handle to the layer on which attribute query will be executed.

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

Returns:

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

References OGR_L_SetAttributeFilter().

Referenced by OGR_L_SetAttributeFilter().

17.11.2.135 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. 101).

Use OGR_L_TestCapability(OLCRandomWrite) to establish if this layer supports random access writing via **OGR_L_SetFeature()** (p. 396).

This function is the same as the C++ method **OGRLayer::SetFeature()** (p. 166).

Parameters:

hLayer handle to the layer to write the feature.

hFeat the feature to write.

Returns:

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

References OGR_L_SetFeature().

Referenced by OGR_L_SetFeature().

17.11.2.136 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. 394) 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. 382)) overlaps the envelope of the spatial filter will be returned. This can result in more shapes being returned that should strictly be the case.

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

For the time being the passed filter geometry should be in the same SRS as the layer (as returned by **OGR_L_GetSpatialRef()** (p. 395)). In the future this may be generalized.

This function is the same as the C++ method **OGRLayer::SetSpatialFilter** (p. 167).

Parameters:

hLayer handle to the layer on which to set the spatial filter.

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

References OGR_L_SetSpatialFilter().

Referenced by OGR_L_SetSpatialFilter().

17.11.2.137 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 GetNextFeature() method. Only features that geometrically intersect the given rectangle will be returned.

The x/y values should be in the same coordinate system as the layer as a whole (as returned by **OGRLayer::GetSpatialRef()** (p. 165)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. 167). 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. 167).

Parameters:

hLayer handle to the layer on which to set the spatial filter.

dfMinX the minimum X coordinate for the rectangular region.

dfMinY the minimum Y coordinate for the rectangular region.

dfMaxX the maximum X coordinate for the rectangular region.

dfMaxY the maximum Y coordinate for the rectangular region.

References OGR_L_SetSpatialFilterRect().

Referenced by OGR_L_SetSpatialFilterRect().

17.11.2.138 OGRErr OGR_L_StartTransaction (OGRLayerH *hLayer*)

For datasources which support transactions, StartTransaction creates a transaction. If starting the transaction fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_NONE.

This function is the same as the C++ method OGRLayer::StartTransaction().

Parameters:

hLayer handle to the layer

Returns:

OGRERR_NONE on success.

References OGR_L_StartTransaction().

Referenced by OGR_L_StartTransaction().

17.11.2.139 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 **OGR_L_GetFeature()** (p. 393) function works for this layer.
- **OLCSequentialWrite /** "SequentialWrite": TRUE if the **OGR_L_CreateFeature()** (p. 391) function works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 160) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite /** "RandomWrite": TRUE if the **OGR_L_SetFeature()** (p. 396) function is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. 160) 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. 101) intersection methods should return FALSE. This can be used as a clue by the application whether it should build and maintain it's own spatial index for features in this layer.
- **OLCFastFeatureCount /** "FastFeatureCount": TRUE if this layer can return a feature count (via **OGR_L_GetFeatureCount()** (p. 393)) efficiently ... ie. without counting the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastGetExtent /** "FastGetExtent": TRUE if this layer can return its data extent (via **OGR_L_GetExtent()** (p. 392)) efficiently ... ie. without scanning all the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.

This function is the same as the C++ method **OGRLayer::TestCapability()** (p. 169).

Parameters:

hLayer handle to the layer to get the capability from.

pszCap the name of the capability to test.

Returns:

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

References OGR_L_TestCapability().

Referenced by OGR_L_TestCapability().

17.11.2.140 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().

Parameters:

hSM handle to the style manager.

hST the style tool defining the part to add.

Returns:

TRUE on success, FALSE on errors.

References OGR_SM_AddPart().

Referenced by OGR_SM_AddPart().

17.11.2.141 OGRStyleMgrH OGR_SM_Create (void * *hStyleTable*)

OGRStyleMgr factory.

This function is the same as the C++ method OGRStyleMgr::OGRStyleMgr().

Parameters:

hStyleTable (currently unused, reserved for future use), pointer to OGRStyleTable. Pass NULL for now.

Returns:

an handle to the new style manager object.

References OGR_SM_Create().

Referenced by OGR_SM_Create().

17.11.2.142 void OGR_SM_Destroy (OGRStyleMgrH *hSM*)

Destroy Style Manager

Parameters:

hSM handle to the style manager to destroy.

References OGR_SM_Destroy().

Referenced by OGR_SM_Destroy().

17.11.2.143 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().

Parameters:

hSM handle to the style manager.

nPartId the part number (0-based index)

pszStyleString (optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

Returns:

OGRStyleToolH of the requested part (style tools) or NULL on error.

References OGR_SM_GetPart().

Referenced by OGR_SM_GetPart().

17.11.2.144 int OGR_SM_GetPartCount (OGRStyleMgrH *hSM*, const char * *pszStyleString*)

Add a part (style tool) to the current style.

This function is the same as the C++ method OGRStyleMgr::GetPartCount().

Parameters:

hSM handle to the style manager.

pszStyleString (optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

Returns:

the number of parts (style tools) in the style.

References OGR_SM_GetPartCount().

Referenced by OGR_SM_GetPartCount().

17.11.2.145 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().

Parameters:

hSM handle to the style manager.

hFeature 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 OGR_SM_InitFromFeature().

Referenced by OGR_SM_InitFromFeature().

17.11.2.146 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().

Parameters:

hSM handle to the style manager.

pszStyleString the style string to use (can be NULL).

Returns:

TRUE on success, FALSE on errors.

References OGR_SM_InitStyleString().

Referenced by OGR_SM_InitStyleString().

17.11.2.147 OGRStyleToolH OGR_ST_Create (OGRSTClassId *eClassId*)

OGRStyleTool factory.

This function is a constructor for OGRStyleTool derived classes.

Parameters:

eClassId subclass of style tool to create. One of OGRSTCPen (1), OGRSTCBrush (2), OGRSTCSymbol (3) or OGRSTCLabel (4).

Returns:

an handle to the new style tool object or NULL if the creation failed.

References OGR_ST_Create().

Referenced by OGR_ST_Create().

17.11.2.148 void OGR_ST_Destroy (OGRStyleToolH *hST*)

Destroy Style Tool

Parameters:

hST handle to the style tool to destroy.

References OGR_ST_Destroy().

Referenced by OGR_ST_Destroy().

17.11.2.149 double OGR_ST_GetParamDbl (OGRStyleToolH *hST*, int *eParam*, int * *bValueIsNull*)

Get Style Tool parameter value as a double

Maps to the OGRStyleTool subclasses' GetParamDbl() methods.

Parameters:

hST handle to the style tool.

eParam the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)

bValueIsNull pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

Returns:

the parameter value as double and sets bValueIsNull.

References OGR_ST_GetParamDbl().

Referenced by OGR_ST_GetParamDbl().

17.11.2.150 int OGR_ST_GetParamNum (OGRStyleToolH *hST*, int *eParam*, int * *bValueIsNull*)

Get Style Tool parameter value as an integer

Maps to the OGRStyleTool subclasses' GetParamNum() methods.

Parameters:

hST handle to the style tool.

eParam the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)

bValueIsNull pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

Returns:

the parameter value as integer and sets bValueIsNull.

References OGR_ST_GetParamNum().

Referenced by OGR_ST_GetParamNum().

17.11.2.151 const char* OGR_ST_GetParamStr (OGRStyleToolH *hST*, int *eParam*, int * *bValueIsNull*)

Get Style Tool parameter value as string

Maps to the OGRStyleTool subclasses' GetParamStr() methods.

Parameters:

hST handle to the style tool.

eParam the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)

bValueIsNull pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

Returns:

the parameter value as string and sets bValueIsNull.

References OGR_ST_GetParamStr().

Referenced by OGR_ST_GetParamStr().

17.11.2.152 int OGR_ST_GetRGBFromString (OGRStyleToolH *hST*, const char * *pszColor*, int * *pnRed*, int * *pnGreen*, int * *pnBlue*, int * *pnAlpha*)

Return the r,g,b,a components of a color encoded in RRGGBB[AA] format

Maps to OGRStyleTool::GetRGBFromString().

Parameters:

hST handle to the style tool.

pszColor the color to parse

pnRed pointer to an int in which the red value will be returned

pnGreen pointer to an int in which the green value will be returned

pnBlue pointer to an int in which the blue value will be returned

pnAlpha pointer to an int in which the (optional) alpha value will be returned

Returns:

TRUE if the color could be succesfully parsed, or FALSE in case of errors.

References OGR_ST_GetRGBFromString().

Referenced by OGR_ST_GetRGBFromString().

17.11.2.153 const char* OGR_ST_GetStyleString (OGRStyleToolH *hST*)

Get the style string for this Style Tool

Maps to the OGRStyleTool subclasses' GetStyleString() methods.

Parameters:

hST handle to the style tool.

Returns:

the style string for this style tool or "" if the hST is invalid.

References OGR_ST_GetStyleString().

Referenced by OGR_ST_GetStyleString().

17.11.2.154 OGRSTClassId OGR_ST_GetType (OGRStyleToolH *hST*)

Determine type of Style Tool

Parameters:

hST handle to the style tool.

Returns:

the style tool type, one of OGRSTCPen (1), OGRSTCBrush (2), OGRSTCSymbol (3) or OGRSTCLabel (4). Returns OGRSTCNone (0) if the OGRStyleToolH is invalid.

References OGR_ST_GetType().

Referenced by OGR_ST_GetType().

17.11.2.155 OGRSTUnitId OGR_ST_GetUnit (OGRStyleToolH *hST*)

Get Style Tool units

Parameters:

hST handle to the style tool.

Returns:

the style tool units.

References OGR_ST_GetUnit().

Referenced by OGR_ST_GetUnit().

17.11.2.156 void OGR_ST_SetParamNum (OGRStyleToolH *hST*, int *eParam*, int *nValue*)

Set Style Tool parameter value from an integer

Maps to the OGRStyleTool subclasses' SetParamNum() methods.

Parameters:

hST handle to the style tool.

eParam the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)

nValue the new parameter value

References OGR_ST_SetParamNum().

Referenced by OGR_ST_SetParamNum().

17.11.2.157 void OGR_ST_SetParamStr (OGRStyleToolH *hST*, int *eParam*, const char * *pszValue*)

Set Style Tool parameter value from a string

Maps to the OGRStyleTool subclasses' SetParamStr() methods.

Parameters:

hST handle to the style tool.

eParam the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)

pszValue the new parameter value

References OGR_ST_SetParamStr().

Referenced by OGR_ST_SetParamStr().

17.11.2.158 void OGR_ST_SetUnit (OGRStyleToolH *hST*, OGRSTUnitId *eUnit*, double *dfGroundPaperScale*)

Set Style Tool units

This function is the same as OGRStyleTool::SetUnit()

Parameters:

hST handle to the style tool.

eUnit the new unit.

dfGroundPaperScale ground to paper scale factor.

References OGR_ST_SetUnit().

Referenced by OGR_ST_SetUnit().

17.11.2.159 OGRGeometryH OGRBuildPolygonFromEdges (OGRGeometryH *hLines*, int *bBestEffort*, int *bAutoClose*, double *dfTolerance*, OGRErr * *peErr*)

Build a ring from a bunch of arcs.

Parameters:

hLines handle to an **OGRGeometryCollection** (p. 144) (or **OGRMultiLineString** (p. 188)) containing the line string geometries to be built into rings.

bBestEffort not yet implemented???.

bAutoClose indicates if the ring should be close when first and last points of the ring are the same.

dfTolerance tolerance into which two arcs are considered close enough to be joined.

peErr OGRERR_NONE on success, or OGRERR_FAILURE on failure.

Returns:

an handle to the new geometry, a polygon.

References OGRLineString::addPoint(), OGRPolygon::addRingDirectly(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), OGRLineString::getNumPoints(), OGRLineString::getX(), OGRLineString::getY(), OGRLineString::getZ(), and OGRBuildPolygonFromEdges().

Referenced by OGRBuildPolygonFromEdges().

17.11.2.160 void OGRCleanupAll (void)

Cleanup all OGR related resources.

This function will destroy the **OGRSFDriverRegistrar** (p. 220) along with all registered drivers, and then cleanup long lived OSR (**OGRSpatialReference** (p. 224)) and CPL resources. This may be called in an application when OGR services are no longer needed. It is not normally required, but by freeing all dynamically allocated memory it can make memory leak testing easier.

In addition to destroying the OGRDriverRegistrar, this function also calls:

- OSRCleanup()
- CPLFinderClean()
- VSICleanupFileManager()
- CPLFreeConfig()
- CPLCleanupTLS()

References OGRCleanupAll().

Referenced by OGRCleanupAll().

17.11.2.161 OGRSFDriverH OGRGetDriver (int *iDriver*)

Fetch the indicated driver.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriver()** (p. 221).

Parameters:

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

Returns:

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

References OGRSFDriverRegistrar::GetDriver(), and OGRGetDriver().

Referenced by OGRGetDriver().

17.11.2.162 int OGRGetDriverCount (void)

Fetch the number of registered drivers.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriverCount()** (p. 221).

Returns:

the drivers count.

References OGRSFDriverRegistrar::GetDriverCount(), and OGRGetDriverCount().

Referenced by OGRGetDriverCount().

17.11.2.163 OGRDataSourceH OGROpen (const char * *pszName*, int *bUpdate*, OGRSFDriverH * *pahDriverList*)

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

This function loops through all the drivers registered with the driver manager trying each until one succeeds with the given data source. This function is static. Applications don't normally need to use any other **OGRSFDriverRegistrar** (p. 220) function, not do they normally need to have a pointer to an **OGRSFDriverRegistrar** (p. 220) instance.

If this function fails, **CPLGetLastErrorMsg()** (p. 299) can be used to check if there is an error message explaining why.

This function is the same as the C++ method **OGRSFDriverRegistrar::Open()** (p. 221).

Parameters:

pszName the name of the file, or data source to open.

bUpdate FALSE for read-only access (the default) or TRUE for read-write access.

pahDriverList if non-NULL, this argument will be updated with a pointer to the driver which was used to open the data source.

Returns:

NULL on error or if the pass name is not supported by this driver, otherwise an handle to an **OGRDataSource** (p. 92). This **OGRDataSource** (p. 92) should be closed by deleting the object when it is no longer needed.

Example:

```
OGRDataSourceH hDS;
OGRSFDriverH        *pahDriver;

hDS = OGROpen( "polygon.shp", 0, pahDriver );
if( hDS == NULL )
{
    return;
}

... use the data source ...

OGRReleaseDataSource( hDS );
```

References OGROpen(), and OGRSFDriverRegistrar::Open().

Referenced by OGROpen().

17.11.2.164 int OGRRegisterAll (void)

Register all drivers.

Referenced by OGRRegisterAll().

17.11.2.165 void OGRRegisterDriver (OGRSFDriverH *hDriver*)

Add a driver to the list of registered drivers.

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

This function is the same as the C++ method **OGRSFDriverRegistrar::RegisterDriver()** (p. 222).

Parameters:

hDriver handle to the driver to add.

References OGRSFDriverRegistrar::GetRegistrar(), OGRRegisterDriver(), and OGRSFDriverRegistrar::RegisterDriver().

Referenced by OGRRegisterDriver().

17.12 ogr_core.h File Reference

```
#include "cpl_port.h"
#include "gdal_version.h"
```

Classes

- class **OGREnvelope**
- union **OGRField**

Defines

- #define **GDAL_CHECK_VERSION**(pszCallingComponentName) GDALCheckVersion(GDAL_-VERSION_MAJOR, GDAL_VERSION_MINOR, pszCallingComponentName)

Typedefs

- typedef enum **ogr_style_tool_class_id OGRSTClassId**
- typedef enum **ogr_style_tool_units_id OGRSTUnitId**
- typedef enum **ogr_style_tool_param_pen_id OGRSTPenParam**
- typedef enum **ogr_style_tool_param_brush_id OGRSTBrushParam**
- typedef enum **ogr_style_tool_param_symbol_id OGRSTSymbolParam**
- typedef enum **ogr_style_tool_param_label_id OGRSTLabelParam**

Enumerations

- enum **OGRwkbGeometryType** {

  wkbUnknown = 0, **wkbPoint** = 1, **wkbLineString** = 2, **wkbPolygon** = 3,

  wkbMultiPoint = 4, **wkbMultiLineString** = 5, **wkbMultiPolygon** = 6, **wkbGeometryCollection** = 7,

  wkbNone = 100, **wkbLinearRing** = 101, **wkbPoint25D** = 0x80000001, **wkbLineString25D** = 0x80000002,

  wkbPolygon25D = 0x80000003, **wkbMultiPoint25D** = 0x80000004, **wkbMultiLineString25D** = 0x80000005, **wkbMultiPolygon25D** = 0x80000006,

  wkbGeometryCollection25D = 0x80000007 }
- enum **OGRFieldType** {

  OFTInteger = 0, **OFTIntegerList** = 1, **OFTReal** = 2, **OFTRealList** = 3,

  OFTString = 4, **OFTStringList** = 5, **OFTWideString** = 6, **OFTWideStringList** = 7,

  OFTBinary = 8, **OFTDate** = 9, **OFTTime** = 10, **OFTDateTime** = 11 }
- enum **OGRJustification**
- enum **ogr_style_tool_class_id**
- enum **ogr_style_tool_units_id**
- enum **ogr_style_tool_param_pen_id**
- enum **ogr_style_tool_param_brush_id**
- enum **ogr_style_tool_param_symbol_id**
- enum **ogr_style_tool_param_label_id**

Functions

- const char * **OGRGeometryTypeToName** (**OGRwkbGeometryType** eType)
- **OGRwkbGeometryType OGRMergeGeometryTypes** (**OGRwkbGeometryType** eMain, **OGRwkbGeometryType** eExtra)
- int **OGRParseDate** (const char *pszInput, **OGRField** *psOutput, int nOptions)
- int CPL_STDCALL **GDALCheckVersion** (int nVersionMajor, int nVersionMinor, const char *pszCallingComponentName)

17.12.1 Detailed Description

Core portability services for cross-platform OGR code.

17.12.2 Define Documentation

17.12.2.1 #define GDAL_CHECK_VERSION(pszCallingComponentName) GDALCheckVersion(GDAL_VERSION_MAJOR, GDAL_VERSION_MINOR, pszCallingComponentName)

Helper macro for GDALCheckVersion

17.12.3 Typedef Documentation

17.12.3.1 typedef enum ogr_style_tool_param_brush_id OGRSTBrushParam

List of parameters for use with OGRStyleBrush.

17.12.3.2 typedef enum ogr_style_tool_class_id OGRSTClassId

OGRStyleTool derived class types (returned by GetType()).

17.12.3.3 typedef enum ogr_style_tool_param_label_id OGRSTLabelParam

List of parameters for use with OGRStyleLabel.

17.12.3.4 typedef enum ogr_style_tool_param_pen_id OGRSTPenParam

List of parameters for use with OGRStylePen.

17.12.3.5 typedef enum ogr_style_tool_param_symbol_id OGRSTSymbolParam

List of parameters for use with OGRStyleSymbol.

17.12.3.6 typedef enum ogr_style_tool_units_id OGRSTUnitId

List of units supported by OGRStyleTools.

17.12.4 Enumeration Type Documentation

17.12.4.1 enum ogr_style_tool_class_id

OGRStyleTool derived class types (returned by GetType()).

17.12.4.2 enum ogr_style_tool_param_brush_id

List of parameters for use with OGRStyleBrush.

17.12.4.3 enum ogr_style_tool_param_label_id

List of parameters for use with OGRStyleLabel.

17.12.4.4 enum ogr_style_tool_param_pen_id

List of parameters for use with OGRStylePen.

17.12.4.5 enum ogr_style_tool_param_symbol_id

List of parameters for use with OGRStyleSymbol.

17.12.4.6 enum ogr_style_tool_units_id

List of units supported by OGRStyleTools.

17.12.4.7 enum OGRFieldType

List of feature field types. This list is likely to be extended in the future ... avoid coding applications based on the assumption that all field types can be known.

Enumerator:

OFTInteger Simple 32bit integer

OFTIntegerList List of 32bit integers

OFTReal Double Precision floating point

OFTRealList List of doubles

OFTString String of ASCII chars

OFTStringList Array of strings

OFTWideString deprecated

OFTWideStringList deprecated

OFTBinary Raw Binary data

OFTDate Date

OFTTime Time

OFTDateTime Date and Time

17.12.4.8 enum OGRJustification

Display justification for field values.

17.12.4.9 enum OGRwkbGeometryType

List of well known binary geometry types. These are used within the BLOBs but are also returned from **OGRGeometry::getGeometryType()** (p. 136) to identify the type of a geometry object.

Enumerator:

wkbUnknown unknown type, non-standard

wkbPoint 0-dimensional geometric object, standard WKB

wkbLineString 1-dimensional geometric object with linear interpolation between Points, standard WKB

wkbPolygon planar 2-dimensional geometric object defined by 1 exterior boundary and 0 or more interior boundaries, standard WKB

wkbMultiPoint GeometryCollection of Points, standard WKB

wkbMultiLineString GeometryCollection of LineStrings, standard WKB

wkbMultiPolygon GeometryCollection of Polygons, standard WKB

wkbGeometryCollection geometric object that is a collection of 1 or more geometric objects, standard WKB

wkbNone non-standard, for pure attribute records

wkbLinearRing non-standard, just for createGeometry()

wkbPoint25D 2.5D extension as per 99-402

wkbLineString25D 2.5D extension as per 99-402

wkbPolygon25D 2.5D extension as per 99-402

wkbMultiPoint25D 2.5D extension as per 99-402

wkbMultiLineString25D 2.5D extension as per 99-402

wkbMultiPolygon25D 2.5D extension as per 99-402

wkbGeometryCollection25D 2.5D extension as per 99-402

17.12.5 Function Documentation

17.12.5.1 int CPL_STDCALL GDALCheckVersion (int *nVersionMajor*, int *nVersionMinor*, const char * *pszCallingComponentName*)

Return TRUE if GDAL library version at runtime matches nVersionMajor.nVersionMinor.

The purpose of this method is to ensure that calling code will run with the GDAL version it is compiled for. It is primarly intented for external plugins.

Parameters:

nVersionMajor Major version to be tested against

nVersionMinor Minor version to be tested against

pszCallingComponentName If not NULL, in case of version mismatch, the method will issue a failure mentionning the name of the calling component.

17.12.5.2 const char* OGRGeometryTypeToName (OGRwkbGeometryType *eType*)

Fetch a human readable name corresponding to an OGRwkBGeometryType value. The returned value should not be modified, or freed by the application.

This function is C callable.

Parameters:

eType the geometry type.

Returns:

internal human readable string, or NULL on failure.

References OGRGeometryTypeToName(), wkbGeometryCollection, wkbGeometryCollection25D, wkbLineString, wkbLineString25D, wkbMultiLineString, wkbMultiLineString25D, wkbMultiPoint, wkbMultiPoint25D, wkbMultiPolygon, wkbMultiPolygon25D, wkbNone, wkbPoint, wkbPoint25D, wkbPolygon, wkbPolygon25D, and wkbUnknown.

Referenced by OGRGeometryTypeToName().

17.12.5.3 OGRwkbGeometryType OGRMergeGeometryTypes (OGRwkbGeometryType *eMain*, OGRwkbGeometryType *eExtra*)

Find common geometry type.

Given two geometry types, find the most specific common type. Normally used repeatedly with the geometries in a layer to try and establish the most specific geometry type that can be reported for the layer.

NOTE: wkbUnknown is the "worst case" indicating a mixture of geometry types with nothing in common but the base geometry type. wkbNone should be used to indicate that no geometries have been encountered yet, and means the first geometry encounted will establish the preliminary type.

Parameters:

eMain the first input geometry type.

eExtra the second input geometry type.

Returns:

the merged geometry type.

References OGRMergeGeometryTypes(), wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbNone, and wkbUnknown.

Referenced by OGRMergeGeometryTypes().

17.12.5.4 int OGRParseDate (const char * *pszInput*, OGRField * *psField*, int *nOptions*)

Parse date string.

This function attempts to parse a date string in a variety of formats into the OGRField.Date format suitable for use with OGR. Generally speaking this function is expecting values like:

YYYY-MM-DD HH:MM:SS+nn

The seconds may also have a decimal portion (which is ignored). And just dates (YYYY-MM-DD) or just times (HH:MM:SS) are also supported. The date may also be in YYYY/MM/DD format. If the year is less than 100 and greater than 30 a "1900" century value will be set. If it is less than 30 and greater than -1 then a "2000" century value will be set. In the future this function may be generalized, and additional control provided through nOptions, but an nOptions value of "0" should always do a reasonable default form of processing.

The value of psField will be indeterminate if the function fails (returns FALSE).

Parameters:

pszInput the input date string.

psField the **OGRField** (p. 121) that will be updated with the parsed result.

nOptions parsing options, for now always 0.

Returns:

TRUE if apparently successful or FALSE on failure.

References OGRField::Date, OGRField::Day, OGRField::Hour, OGRField::Minute, OGRField::Month, OGRParseDate(), OGRField::Second, OGRField::TZFlag, and OGRField::Year.

Referenced by OGRParseDate(), and OGRFeature::SetField().

17.13 ogr_feature.h File Reference

```
#include "ogr_geometry.h"
#include "ogr_featurestyle.h"
```

Classes

- class **OGRFieldDefn**
- class **OGRFeatureDefn**
- class **OGRFeature**
- class **OGRFeatureQuery**

17.13.1 Detailed Description

Simple feature classes.

17.14 ogr_geometry.h File Reference

```
#include "ogr_core.h"
#include "ogr_spatialref.h"
```

Classes

- class **OGRRawPoint**
- class **OGRGeometry**
- class **OGRPoint**
- class **OGRCurve**
- class **OGRLineString**
- class **OGRLinearRing**
- class **OGRSurface**
- class **OGRPolygon**
- class **OGRGeometryCollection**
- class **OGRMultiPolygon**
- class **OGRMultiPoint**
- class **OGRMultiLineString**
- class **OGRGeometryFactory**

17.14.1 Detailed Description

Simple feature geometry classes.

17.15 ogr_spatialref.h File Reference

```
#include "ogr_srs_api.h"
```

Classes

- class **OGR_SRSNode**
- class **OGRSpatialReference**
- class **OGRCoordinateTransformation**

Functions

- **OGRCoordinateTransformation** * **OGRCreateCoordinateTransformation** (**OGRSpatialReference** *poSource, **OGRSpatialReference** *poTarget)

17.15.1 Detailed Description

Coordinate systems services.

17.15.2 Function Documentation

17.15.2.1 OGRCoordinateTransformation* OGRCreateCoordinateTransformation (OGRSpatialReference * *poSource*, OGRSpatialReference * *poTarget*)

Create transformation object.

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

Input spatial reference system objects are assigned by copy (calling clone() method) and no ownership transfer occurs.

The delete operator, or OCTDestroyCoordinateTransformation() should be used to destroy transformation objects.

Parameters:

poSource source spatial reference system.

poTarget target spatial reference system.

Returns:

NULL on failure or a ready to use transformation object.

References OGRCreateCoordinateTransformation().

Referenced by OGRCreateCoordinateTransformation(), and OGRGeometry::transformTo().

17.16 ogr_srs_api.h File Reference

```
#include "ogr_core.h"
```

Functions

- OGRErr **OSRImportFromWkt** (OGRSpatialReferenceH, char **)
- OGRErr CPL_STDCALL **OSRExportToWkt** (OGRSpatialReferenceH, char **)
- OGRErr **OSRSetACEA** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetAE** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetBonne** (OGRSpatialReferenceH hSRS, double dfStandardParallel, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetCEA** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetCS** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEC** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEckert** (OGRSpatialReferenceH hSRS, int nVariation, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEckertIV** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEckertVI** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEquirectangular** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetEquirectangular2** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfPseudoStdParallel1, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGS** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGH** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGEOS** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfSatelliteHeight, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGaussSchreiberTMercator** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGnomonic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetHOM** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetHOM2PNO** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetIWMPolyconic** (OGRSpatialReferenceH hSRS, double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetKrovak** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **OSRSetLAEA** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCC** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCC1SP** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCCB** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMC** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMercator** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMollweide** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetNZMG** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetOS** (OGRSpatialReferenceH hSRS, double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetOrthographic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetPolyconic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetPS** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetRobinson** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetSinusoidal** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetStereographic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetSOC** (OGRSpatialReferenceH hSRS, double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTM** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMVariant** (OGRSpatialReferenceH hSRS, const char ∗pszVariantName, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMG** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMSO** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetVDG** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetWagner** (OGRSpatialReferenceH hSRS, int nVariation, double dfFalseEasting, double dfFalseNorthing)
- char ∗∗ **OPTGetProjectionMethods** ()
- char ∗∗ **OPTGetParameterList** (const char ∗pszProjectionMethod, char ∗∗ppszUserName)
- int **OPTGetParameterInfo** (const char ∗pszProjectionMethod, const char ∗pszParameterName, char ∗∗ppszUserName, char ∗∗ppszType, double ∗pdfDefaultValue)

17.16.1 Detailed Description

C spatial reference system services and defines.

See also: **ogr_spatialref.h** (p. 417)

17.16.2 Function Documentation

17.16.2.1 int OPTGetParameterInfo (const char * *pszProjectionMethod*, const char * *pszParameterName*, char ** *ppszUserName*, char ** *ppszType*, double * *pdfDefaultValue*)

Fetch information about a single parameter of a projection method.

Parameters:

pszProjectionMethod name of projection method for which the parameter applies. Not currently used, but in the future this could affect defaults. This is the internal projection method name, such as "Tranverse_Mercator".

pszParameterName name of the parameter to fetch information about. This is the internal name such as "central_meridian" (SRS_PP_CENTRAL_MERIDIAN).

ppszUserName location at which to return the user visible name for the parameter. This pointer may be NULL to skip the user name. The returned name should not be modified or freed.

ppszType location at which to return the parameter type for the parameter. This pointer may be NULL to skip. The returned type should not be modified or freed. The type values are described above.

pdfDefaultValue location at which to put the default value for this parameter. The pointer may be NULL.

Returns:

TRUE if parameter found, or FALSE otherwise.

17.16.2.2 char** OPTGetParameterList (const char * *pszProjectionMethod*, char ** *ppszUserName*)

Fetch the parameters for a given projection method.

Parameters:

pszProjectionMethod internal name of projection methods to fetch the parameters for, such as "Transverse_Mercator" (SRS_PT_TRANSVERSE_MERCATOR).

ppszUserName pointer in which to return a user visible name for the projection name. The returned string should not be modified or freed by the caller. Legal to pass in NULL if user name not required.

Returns:

returns a NULL terminated list of internal parameter names that should be freed by the caller when no longer needed. Returns NULL if projection method is unknown.

17.16.2.3 char** OPTGetProjectionMethods ()

Fetch list of possible projection methods.

Returns:

Returns NULL terminated list of projection methods. This should be freed with **CSLDestroy()** (p. 328) when no longer needed.

17.16.2.4 OGRErr CPL_STDCALL OSRExportToWkt (OGRSpatialReferenceH *hSRS*, char ** *ppszReturn*)

Convert this SRS into WKT format.

This function is the same as **OGRSpatialReference::exportToWkt()** (p. 232).

References OSRExportToWkt().

Referenced by OSRExportToWkt().

17.16.2.5 OGRErr OSRImportFromWkt (OGRSpatialReferenceH *hSRS*, char ** *ppszInput*)

Import from WKT string.

This function is the same as **OGRSpatialReference::importFromWkt()** (p. 250).

References OSRImportFromWkt().

Referenced by OSRImportFromWkt().

17.16.2.6 OGRErr OSRSetACEA (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Albers Conic Equal Area

References OSRSetACEA().

Referenced by OSRSetACEA().

17.16.2.7 OGRErr OSRSetAE (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Azimuthal Equidistant

References OSRSetAE().

Referenced by OSRSetAE().

17.16.2.8 OGRErr OSRSetBonne (OGRSpatialReferenceH *hSRS*, double *dfStandardParallel*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Bonne

References OSRSetBonne().

Referenced by OSRSetBonne().

17.16.2.9 OGRErr OSRSetCEA (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cylindrical Equal Area

References OSRSetCEA().

Referenced by OSRSetCEA().

17.16.2.10 OGRErr OSRSetCS (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Cassini-Soldner

References OSRSetCS().

Referenced by OSRSetCS().

17.16.2.11 OGRErr OSRSetEC (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equidistant Conic

References OSRSetEC().

Referenced by OSRSetEC().

17.16.2.12 OGRErr OSRSetEckert (OGRSpatialReferenceH *hSRS*, int *nVariation*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert I-VI

References OSRSetEckert().

Referenced by OSRSetEckert().

17.16.2.13 OGRErr OSRSetEckertIV (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert IV

References OSRSetEckertIV().

Referenced by OSRSetEckertIV().

17.16.2.14 OGRErr OSRSetEckertVI (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Eckert VI

References OSRSetEckertVI().

Referenced by OSRSetEckertVI().

17.16.2.15 OGRErr OSRSetEquirectangular (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equirectangular

References OSRSetEquirectangular().

Referenced by OSRSetEquirectangular().

17.16.2.16 OGRErr OSRSetEquirectangular2 (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfPseudoStdParallel1*, double *dfFalseEasting*, double *dfFalseNorthing*)

Equirectangular generalized form

References OSRSetEquirectangular2().

Referenced by OSRSetEquirectangular2().

17.16.2.17 OGRErr OSRSetGaussSchreiberTMercator (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gauss Schreiber Transverse Mercator

References OSRSetGaussSchreiberTMercator().

Referenced by OSRSetGaussSchreiberTMercator().

17.16.2.18 OGRErr OSRSetGEOS (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfSatelliteHeight*, double *dfFalseEasting*, double *dfFalseNorthing*)

GEOS - Geostationary Satellite View

References OSRSetGEOS().

Referenced by OSRSetGEOS().

17.16.2.19 OGRErr OSRSetGH (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Goode Homolosine

References OSRSetGH().

Referenced by OSRSetGH().

17.16.2.20 OGRErr OSRSetGnomonic (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gnomonic

References OSRSetGnomonic().

Referenced by OSRSetGnomonic().

17.16.2.21 OGRErr OSRSetGS (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Gall Stereograpic

References OSRSetGS().

Referenced by OSRSetGS().

17.16.2.22 OGRErr OSRSetHOM (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfAzimuth*, double *dfRectToSkew*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Hotine Oblique Mercator using azimuth angle

References OSRSetHOM().

Referenced by OSRSetHOM().

17.16.2.23 OGRErr OSRSetHOM2PNO (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfLat1*, double *dfLong1*, double *dfLat2*, double *dfLong2*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Hotine Oblique Mercator using two points on centerline

References OSRSetHOM2PNO().

Referenced by OSRSetHOM2PNO().

17.16.2.24 OGRErr OSRSetIWMPolyconic (OGRSpatialReferenceH *hSRS*, double *dfLat1*, double *dfLat2*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

International Map of the World Polyconic

References OSRSetIWMPolyconic().

Referenced by OSRSetIWMPolyconic().

17.16.2.25 OGRErr OSRSetKrovak (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfAzimuth*, double *dfPseudoStdParallelLat*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Krovak Oblique Conic Conformal

References OSRSetKrovak().

Referenced by OSRSetKrovak().

17.16.2.26 OGRErr OSRSetLAEA (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Azimuthal Equal-Area

References OSRSetLAEA().

Referenced by OSRSetLAEA().

17.16.2.27 OGRErr OSRSetLCC (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic

References OSRSetLCC().

Referenced by OSRSetLCC().

17.16.2.28 OGRErr OSRSetLCC1SP (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic 1SP

References OSRSetLCC1SP().

Referenced by OSRSetLCC1SP().

17.16.2.29 OGRErr OSRSetLCCB (OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Lambert Conformal Conic (Belgium)

References OSRSetLCCB().

Referenced by OSRSetLCCB().

17.16.2.30 OGRErr OSRSetMC (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Miller Cylindrical

References OSRSetMC().

Referenced by OSRSetMC().

17.16.2.31 OGRErr OSRSetMercator (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Mercator

References OSRSetMercator().

Referenced by OSRSetMercator().

17.16.2.32 OGRErr OSRSetMollweide (OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Mollweide

References OSRSetMollweide().

Referenced by OSRSetMollweide().

17.16.2.33 OGRErr OSRSetNZMG (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

New Zealand Map Grid

References OSRSetNZMG().

Referenced by OSRSetNZMG().

17.16.2.34 OGRErr OSRSetOrthographic (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Orthographic

References OSRSetOrthographic().

Referenced by OSRSetOrthographic().

17.16.2.35 OGRErr OSRSetOS (OGRSpatialReferenceH *hSRS*, double *dfOriginLat*, double *dfCMeridian*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Oblique Stereographic

References OSRSetOS().

Referenced by OSRSetOS().

17.16.2.36 OGRErr OSRSetPolyconic (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Polyconic

References OSRSetPolyconic().

Referenced by OSRSetPolyconic().

17.16.2.37 OGRErr OSRSetPS (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Polar Stereographic

References OSRSetPS().

Referenced by OSRSetPS().

17.16.2.38 OGRErr OSRSetRobinson (OGRSpatialReferenceH *hSRS*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Robinson

References OSRSetRobinson().

Referenced by OSRSetRobinson().

17.16.2.39 OGRErr OSRSetSinusoidal (OGRSpatialReferenceH *hSRS*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Sinusoidal

References OSRSetSinusoidal().

Referenced by OSRSetSinusoidal().

17.16.2.40 OGRErr OSRSetSOC (OGRSpatialReferenceH *hSRS*, double *dfLatitudeOfOrigin*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing*)

Swiss Oblique Cylindrical

References OSRSetSOC().

Referenced by OSRSetSOC().

17.16.2.41 OGRErr OSRSetStereographic (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Stereographic

References OSRSetStereographic().

Referenced by OSRSetStereographic().

17.16.2.42 OGRErr OSRSetTM (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator

References OSRSetTM().

Referenced by OSRSetTM().

17.16.2.43 OGRErr OSRSetTMG (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

Tunesia Mining Grid

References OSRSetTMG().

Referenced by OSRSetTMG().

17.16.2.44 OGRErr OSRSetTMSO (OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator (South Oriented)

References OSRSetTMSO().

Referenced by OSRSetTMSO().

17.16.2.45 OGRErr OSRSetTMVariant (OGRSpatialReferenceH *hSRS*, const char * *pszVariantName*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing*)

Transverse Mercator variant

References OSRSetTMVariant().

Referenced by OSRSetTMVariant().

17.16.2.46 OGRErr OSRSetVDG (OGRSpatialReferenceH *hSRS*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing*)

VanDerGrinten

References OSRSetVDG().

Referenced by OSRSetVDG().

17.16.2.47 OGRErr OSRSetWagner (OGRSpatialReferenceH *hSRS*, int *nVariation*, double *dfFalseEasting*, double *dfFalseNorthing*)

Wagner I – VII

17.17 ogrsf_frmts.h File Reference

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

Classes

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

Functions

- void **OGRRegisterAll** ()

17.17.1 Detailed Description

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

17.17.2 Function Documentation

17.17.2.1 void OGRRegisterAll (void)

Register all drivers.

References OGRSFDriverRegistrar::AutoLoadDrivers(), OGRSFDriverRegistrar::GetRegistrar(), and OGRRegisterAll().

Index