

SDTS_AL

Contents

Chapter 1

ISO8211Lib

Introduction

ISO8211Lib is intended to be a simple reader for ISO/IEC 8211 formatted files, particularly those that are part of SDTS and S-57 datasets. It consists of open source, easy to compile and integrate C++ code.

ISO 8211 Background

The [ISO 8211 FAQ](#) has some good background on ISO 8211 formatted files. I will briefly introduce it here, with reference to the library classes representing the components.

An 8211 file ([DDFModule](#)) consists of a series of logical records. The first record is special, and is called the DDR (Data Description Record). It basically contains definitions of all the data objects (fields or [DDFFieldDefn](#) objects) that can occur on the following data records.

The remainder of the records are known as DRs (data records - [DDFRecord](#)). They each contain one or more field ([DDFField](#)) instances. What fields appear on what records is not defined by ISO 8211, though more specific requirements may be implied by a particular data standard such as SDTS or S-57.

Each field instance has a name, and consists of a series of subfields. A given field always has the same subfields in each field instance, and these subfields are defined in the DDR ([DDFSubfieldDefn](#)), in association with their field definition ([DDFFieldDefn](#)). A field may appear 0, 1, or many times in a DR.

Each subfield has a name, format (from the [DDFSubfieldDefn](#)) and actual subfield data for a particular DR. Some fields contain an *array* of their group of subfields. For instance a *coordinate field* may have X and Y subfields, and they may repeat many times within one coordinate field indicating a series of points.

This would be a real good place for a UML diagram of ISO 8211, and the corresponding library classes!

Development Information

The [iso8211.h](#) contains the definitions for all public ISO8211Lib classes, enumerations and other services.

To establish access to an ISO 8211 dataset, instantiate a [DDFModule](#) object, and then use the [DDFModule::Open\(\)](#) method. This will read the DDR, and establish all the [DDFFieldDefn](#), and [DDFSubfieldDefn](#) objects which can be queried off the [DDFModule](#).

The use [DDFModule::ReadRecord\(\)](#) to fetch data records ([DDFRecord](#)). When a record is read, a list of field objects ([DDFField](#)) on that record are created. They can be queried with various [DDFRecord](#) methods.

Data pointers for individual subfields of a [DDFField](#) can be fetched with [DDFField::GetSubfieldData\(\)](#). The interpreted value can then be extracted with the appropriate one of [DDFSubfieldDefn::ExtractIntValue\(\)](#), [DDFSubfieldDefn::ExtractStringValue\(\)](#), or [DDFSubfieldDefn::ExtractFloatValue\(\)](#). Note that there is no object instantiated for individual subfields of a [DDFField](#). Instead the application extracts a pointer to the subfields raw data, and then uses the [DDFSubfieldDefn](#) for that subfield to extract a useable value from the raw data.

Once the end of the file has been encountered (`DDFModule::ReadRecord()` returns NULL), the `DDFModule` should be deleted, which will close the file, and cleanup all records, definitions and related objects.

Class APIs

- `DDFModule` class.
- `DDFFieldDefn` class.
- `DDFSubfieldDefn` class.
- `DDFRecord` class.
- `DDFField` class.

A complete `Example Reader` should clarify simple use of ISO8211Lib.

Related Information

- The ISO 8211 standard can be ordered through [ISO](#). It cost me about \$200CDN.
- The [ISO/IEC 8211/DDFS Home Page](#) contains tutorials and some code by Dr. Alfred A. Brooks, one of the originators of the 8211 standard.
- The [ISO/IEC 8211 Home Page](#) has some python code for parsing 8211 files, and some other useful background.
- The [SDTS++](#) library from the USGS includes support for ISO 8211. It doesn't include some of the 1994 additions to ISO 8211, but it is sufficient for SDTS, and quite elegantly done. Also supports writing ISO 8211 files.
- The USGS also has an older [FIPS123](#) library which supports the older profile of ISO 8211 (to some extent).

Licensing

This library is offered as [Open Source](#). In particular, it is offered under the X Consortium license which doesn't attempt to impose any copyleft, or credit requirements on users of the code.

The precise license text is:

Copyright (c) 1999, Frank Warmerdam

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Building the Source

1. First, fetch the source. The most recent source should be accessible at an url such as <http://home.gdal.org/projects/iso8211/iso8211lib-1.4.zip>.
2. Untar the source.

```
% unzip iso8211lib-1.4.zip
```
3. On unix you can now type “configure” to establish configuration options.
4. On unix you can now type make to build libiso8211.a, and the sample mainline 8211view.

Windows developers will have to create their own makefile/project but can base it on the very simple Makefile.in provided. As well, you would need to copy cpl_config.h.in to [cpl_config.h](#), and modify as needed. The default will likely work OK, but may result in some compiler warnings. Let me know if you are having difficulties, and I will prepare a VC++ makefile.

Author and Acknowledgements

The primary author of ISO8211Lib is [Frank Warmerdam](#), and I can be reached at warmerdam@pobox.com. I am eager to receive bug reports, and also open to praise or suggestions.

I would like to thank:

- [Safe Software](#) who funded development of this library, and agreed for it to be Open Source.
- Mark Colletti, a primary author of SDTS++ from which I derived most of what I know about ISO 8211 and who was very supportive, answering a variety of questions.
- Tony J Ibbs, author of the ISO/IEC 8211 home page who answered a number of questions, and collected a variety of very useful information.
- Rodney Jensen, for a detailed bug report related to repeating variable length fields (from S-57).

I would also like to dedicate this library to the memory of Sol Katz. Sol released a variety of SDTS (and hence ISO8211) translators, at substantial personal effort, to the GIS community along with the many other generous contributions he made to the community. His example has been an inspiration to me, and I hope similar efforts on my part will contribute to his memory.

Chapter 2

sdts_al_main

<title>SDTS Abstraction Library</title>

Introduction

SDTS_AL, the SDTS Abstraction Library, is intended to be an relatively easy to use library for reading vector from SDTS TVP (Topological Vector Profile) files, primary DLG data from the USGS. It also include support for reading raster data such as USGS DEMs in SDTS format. It consists of open source, easy to compile and integrate C++ code.

SDTS Background

The USGS SDTS Page at <http://mcmweb.er.usgs.gov/sdts> is the definitive source of information on the SDTS format. The SDTS format is based on the ISO 8211 encoding scheme for the underlying files, and the SDTS Abstraction Library uses ISO8211Lib library to decode them. All references to DDF* classes are from ISO8211Lib.

An SDTS Transfer is a grouping of ISO8211 encoded files (ending in the .DDF extension), normally with part of the basename in common. For instance a USGS DLG SDTS transfer might consists of many files matching the SC01?????.DDF pattern. The key file in an SDTS transfer is the catalog file, such as SC01CATD.DDF.

Development Information

The `sdts_al.h` include file contains the definitions for all public SDTS classes, enumerations and other services.

The `SDTSTransfer` class is used to access a transfer as a whole. The `SDTSTransfer::Open()` method is passed the name of the catalog file, such as SC01CATD.DDF, to open.

The `SDTSTransfer` analyses the catalog, and some other aspects of the transfer, and builds a list of feature layers. This list can be accessed using the `SDTSTransfer::GetLayerCount()`, `SDTSTransfer::GetLayerType()`, and `SDTSTransfer::GetLayerIndexedReader()` methods. A typical TVP (Topological Vector Profile) transfer might include three point layers (of type SLTPoint), a line layer (of type SLTLine), a polygon layer (of type SLTPoly) as well as some additional attribute layers (of type SLTAttr). the `SDTSTransfer::GetLayerIndexedReader()` method can be used to instantiate a reader object for reading a particular layer. (NOTE: raster layers are handled differently).

Each type of `SDTSIndexedReader` (`SDTSPointReader`, `SDTSLineReader`, `SDTSPolygonReader`, and `SDTSAttrReader`) returns specific subclasses of `SDTSIndexedFeature` from the `SDTSIndexedReader::GetNextFeature()` method. These classes are `SDTSRawPoint`, `SDTSRawLine`, `SDTSRawPolygon` and `SDTSAttrRecord`. These classes can be investigated for details on the data available for each.

See the [SDTS_AL Tutorial](#) for more information on how to use this library.

Building the Source on Unix

1. First, fetch the source. The most recent source should be accessible at an url such as ftp://gdal.velocet.ca/pub/outgoing/sdts_1_3.tar.gz.

2. Unpack the source.

```
% gzip -d sdts_1_3.tar.gz
% tar xzvf sdts_1_3.tar.gz
```

3. Type “configure” to establish configuration options.
4. Type make to build sdts_al.a, and the sample mainline sdts2shp.

See the [SDTS_AL Tutorial](#) for more information on how to use this library.

Building the Source on Windows

1. First, fetch the source. The most recent source should be accessible at an url such as ftp://gdal.velocet.ca/pub/outgoing/sdts_1_3.zip.

2. Unpack the source.

```
C:> unzip sdts_1_3.zip
```

3. Build using makefile.vc with VC++. You will need the VC++ runtime environment variables (LIB/INCLUDE) set properly. This will build the library (sdts_al.lib), and the executables sdts2shp.exe, 8211view.exe and 8211dump.exe.

```
C:> nmake /f makefile.vc
```

See the [SDTS_AL Tutorial](#) for more information on how to use this library.

The sdts2shp Sample Program

The sdts2shp program distributed with this toolkit is primary intended to serve as an example of how to use the SDTS access library. However, it can be useful to translate SDTS datasets into ESRI Shapefile format.

```
Usage: sdts2shp CATD_filename [-o shapefile_name]
      [-m module_name] [-v]
```

Modules include 'LE01', 'PC01', 'NP01' and 'ARDF'

A typical session in which we inspect the contents of a transfer, and then extract polygon and line layers might look like this:

```
warmerda[134]% sdts2shp data/SC01CATD.DDF -v
```

Layers:

```
ASCF: 'Attribute Primary      '
AHDR: 'Attribute Primary      '
NP01: 'Point-Node             '
NA01: 'Point-Node             '
NO01: 'Point-Node             '
LE01: 'Line                   '
PC01: 'Polygon                 '
```

```
warmerda[135]% sdts2shp data/SC01CATD.DDF -m PC01 -o pc01.shp
```

```
warmerda[136]% sdts2shp data/SC01CATD.DDF -m LE01 -o le01.shp
```

A [prebuilt executable](#) is available for Windows.

Licensing

This library is offered as **Open Source**. In particular, it is offered under the X Consortium license which doesn't attempt to impose any copyleft, or credit requirements on users of the code.

The precise license text is:

Copyright (c) 1999, Frank Warmerdam

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Author and Acknowledgements

The primary author of SDTS_AL is **Frank Warmerdam**, and I can be reached at warmerdam@pobox.com. I am eager to receive bug reports, and also open to praise or suggestions.

I would like to thank:

- **Safe Software** who funded development of this library, and agreed for it to be Open Source.
- Mark Colletti, a primary author of **SDTS++** from which I derived most of what I know about SDTS and ISO8211 and who was very supportive, answering a variety of questions.

I would also like to dedicate this library to the memory of Sol Katz. Sol released a variety of SDTS translators, at substantial personal effort, to the GIS community along with the many other generous contributions he made to the community. His example has been an inspiration to me, and I hope similar efforts on my part will contribute to his memory.

Chapter 3

SDTS_AL_TUT

<title>SDTS Abstraction Library Tutorial</title>

This page is a walk through of the polygon layer portion of the `sdt2shp.cpp` example application. It should give sufficient information to utilize the SDTS_AL library to read SDTS files.

Opening the Transfer

The following statements will open an SDTS transfer. The filename passed to `SDTSTransfer::Open()` should be the name of the catalog file, such as `palo_alto/SC01CATD.DDF`. The `Open()` method returns `FALSE` if it fails for any reason. In addition to the message we print out ourselves, the `SDTSTransfer::Open()` method will also emit it's own error message using `CPLERROR()`. See the `cpl_error.h` page for more information on how to capture and control `CPLERROR()` style error reporting.

```
#include "stds_al.h"

...

SDTSTransfer oTransfer;

if( !oTransfer.Open( pszCATDFilename ) )
{
    fprintf( stderr,
             "Failed to read CATD file '%s'\n",
             pszCATDFilename );
    exit( 100 );
}
```

Getting a Layer List

Once an `SDTSTransfer` has been opened, it is possible to establish what layers are available. The `sdt2shp` example problem includes a `-v` argument to dump a list of available layers. It isn't normally necessary to use the `SDTS_CATD` (catalog) from an application to access SDTS files; however, in this example we use it to fetch a module name, and description for each of the available layers.

In particular, the `SDTSTransfer::GetLayerCount()` method returns the number of feature layers in the transfer and the `SDTSTransfer::GetLayerCATDEntry()` is used to translate layer indexes into `SDTS_CATD` compatible CATD indexes.

```
printf( "Layers:\n" );
for( i = 0; i < oTransfer.GetLayerCount(); i++ )
{
    int      iCATDEntry = oTransfer.GetLayerCATDEntry(i);
```

```

    printf( "   %s: '%s'\n",
            oTransfer.GetCATD()->GetEntryModule(iCATDEntry),
            oTransfer.GetCATD()->GetEntryTypeDesc(iCATDEntry) );
}
printf( "\n" );

```

The following would be a typical layer list. Note that there are many other modules (files) registered with the catalog, but only these ones are considered to be feature layers by the [SDTSTransfer](#) object. The rest are supporting information, much of it, like data quality, is ignored by the SDTS_AL library.

```
warmerda-c[113]% sdts2shp data/SC01CATD.DDF -v
```

Layers:

```

ASCF: 'Attribute Primary      '
AHDR: 'Attribute Primary      '
NP01: 'Point-Node             '
NA01: 'Point-Node             '
NO01: 'Point-Node             '
LE01: 'Line                   '
PC01: 'Polygon                 '

```

Getting a Reader

In order to read polygon features, it is necessary to instantiate a polygon reader on the desired layer. The sdts2shp.cpp program allow the user to select a module name (such as PC01, stored in pszMODN) to write to shape format. Other application might just search for, and operate on all known layers of a desired type.

The [SDTSTransfer::GetLayerIndexedReader\(\)](#) method instantiates a reader of the desired type. In this case we know we are instantiating a [SDTSPolygonReader](#) so we can safely cast the returned [SDTSIndexedReader](#) pointer to the more specific type [SDTSPolygonReader](#).

```

SDTSPolygonReader *poPolyReader;

poPolyReader = (SDTSPolygonReader *)
    poTransfer->GetLayerIndexedReader( poTransfer->FindLayer( pszMODN ) );

if( poPolyReader == NULL )
{
    fprintf( stderr, "Failed to open %s.\n",
            poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
    return;
}

```

Note that readers returned by [SDTSTransfer::GetLayerIndexedReader\(\)](#) are managed by the [SDTSTransfer](#), and should not be deleted by the application.

Collecting Polygon Geometry

The SDTS TVP format does not directly associate a polygons geometry (the points forming it's boundary) with the polygon feature. Instead it is stored in separate line layers, and the lines contain references to the right, and left polygons that the lines border.

The SDTS_AL library provides a convenient method for forming the polygon geometry. Basically just call the [SDTSPolygonReader::AssemblePolygons\(\)](#) method. This method will scan all SLTLine layers in the transfer, indexing them and attaching their line work to the polygons. Then it assembles the line work into rings. It also ensures that the outer ring comes first, that the outer ring is counter-clockwise and that the inner ring(s) are clockwise.

```
poPolyReader->AssembleRings( poTransfer );
```

Upon completion the [SDTSPolygonReader](#) will have been "indexed". That means that all the polygon information will have been read from disk, and the polygon objects will now have information stored with them indicating the list of edges that form their border.

Identifying Attributes

In order to create the schema for the output shapefile dataset, it is necessary to identify the attributes associated with the polygons. There are two types of attributes which can occur. The first are hardcoded attributes specific to the feature type, and the second are generic user attributes stored in a separate primary attribute layer.

In the case of [SDTSRawPolygon](#), there is only one attribute of interest, and that is the record number of the polygon. This is actually stored within the `oModId` data member of the `SDTSIndexedFeature` base class, as will be seen in later examples when we write it to disk. For now we create a DBF field for the record number. This record number is a unique identifier of the polygon within this module/layer.

```
nSDTSRecordField = DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );
```

Identification of user attributes is more complicated. Any feature in a layer can have associates with 0, 1, 2 or potentially more attribute records in other primary attribute layers. In order to establish a schema for the layer it is necessary to build up a list of all attribute layers (tables) to which references appear. The [SDTSIndexedReader::ScanModuleReferences\(\)](#) method can be used to scan a whole module for references to attribute modules via the `ATID` field. The return result is a list of referenced modules in the form of a string list. In a typical case this is one or two modules, such as "ASCF".

```
char **papszModRefs = poPolyReader->ScanModuleReferences();
```

In `sdts2shp.cpp`, a subroutine (`AddPrimaryAttrToDBFSchema()`) is defined to add all the fields of all references attribute layers to the DBF file. For each module in the list the following steps are executed.

Fetch an Attribute Module Reader

The following code is similar to our code for create a polygon layer reader. It creates a reader on one of the attribute layers referenced. We explicitly rewind it since it may have been previously opened and read by another part of the application.

```
SDTSAttrReader *poAttrReader;

poAttrReader = (SDTSAttrReader *)
    poTransfer->GetLayerIndexedReader(
        poTransfer->FindLayer( papszModuleList[iModule] ) );

if( poAttrReader == NULL )
{
    printf( "Unable to open attribute module %s, skipping.\n" ,
        papszModuleList[iModule] );
    continue;
}

poAttrReader->Rewind();
```

Get a Prototype Record

In order to get access to field definitions, and in order to establish some sort of reasonable default lengths for field without fixed lengths the `sdts2shp` program fetches a prototype record from the attribute module.

```
SDTSAttrRecord *poAttrFeature;
```

```

poAttrFeature = (SDTSAttrRecord *) poAttrReader->GetNextFeature();
if( poAttrFeature == NULL )
{
    fprintf( stderr,
             "Didn't find any meaningful attribute records in %s.\n",
             papszModuleList[iModule] );

    continue;
}

```

When no longer needed, the attribute record may need to be explicitly deleted if it is not part of an indexed cached.

```

if( !poAttrReader->IsIndexed() )
    delete poAttrFeature;

```

Extract Field Definitions

The Shapefile DBF fields are defined based on the information available for each of the subfields of the attribute records ATTR [DDFField](#) (the poATTR data member). The following code loops over each of the subfields, getting a pointer to the DDSubfieldDefn containing information about that subfield.

```

DDFFieldDefn *poFDefn = poAttrFeature->poATTR->GetFieldDefn();
int iSF;
DDFField *poSR = poAttrFeature->poATTR;

for( iSF=0; iSF < poFDefn->GetSubfieldCount(); iSF++ )
{
    DDSubfieldDefn *poSFDefn = poFDefn->GetSubfield( iSF );

```

Then each of the significant ISO8211 field types is translated to an appropriate DBF field type. In cases where the nWidth field is zero, indicating that the field is variable width, we use the length of the field in the prototype record. Ideally we would scan the whole file to find the longest value for each field, but that would be a significant amount of work.

```

int nWidth = poSFDefn->GetWidth();

switch( poSFDefn->GetType() )
{
    case DDFString:
        if( nWidth == 0 )
        {
            int nMaxBytes;

            const char * pachData = poSR->GetSubfieldData(poSFDefn,
                                                         &nMaxBytes);

            nWidth = strlen(poSFDefn->ExtractStringData(pachData,
                                                         nMaxBytes, NULL ));
        }

        DBFAddField( hDBF, poSFDefn->GetName(), FTString, nWidth, 0 );
        break;

    case DDFInt:
        if( nWidth == 0 )
            nWidth = 9;

```

```

        DBFAddField( hDBF, poSFDefn->GetName(), FTInteger, nWidth, 0 );
        break;

    case DDFFloat:
        DBFAddField( hDBF, poSFDefn->GetName(), FTDouble, 18, 6 );
        break;

    default:
        fprintf( stderr,
            "Dropping attribute '%s' of module '%s'. "
            "Type unsupported\n",
            poSFDefn->GetName(),
            papszModuleList[iModule] );
        break;
    }
}

```

Reading Polygon Features

With definition of the attribute schema out of the way, we return to the main event, reading polygons from the polygon layer. We have already instantiated the [SDTSPolygonReader](#) (`poPolyReader`), and now we loop reading features from it. Note that we `Rewind()` the reader to ensure we are starting at the beginning. After we are done process the polygon we delete it, if and only if the layer does not have an index cache.

```

SDTSRawPolygon *poRawPoly;

poPolyReader->Rewind();
while( (poRawPoly = (SDTSRawPolygon *) poPolyReader->GetNextFeature())
        != NULL )
{
    ... process and write polygon ...

    if( !poPolyReader->IsIndexed() )
        delete poRawPoly;
}

```

Translate Geometry

In an earlier step we used the [SDTSPolygonReader::AssembleRings\(\)](#) method to build ring geometry on the polygons from the linework in the line layers.

Coincidentally (well, ok, maybe it isn't a coincidence) it so happens that the ring organization exactly matches what is needed for the shapefile api. The following call creates a polygon from the ring information in the [SDTSRawPolygon](#). See the [SDTSRawPolygon](#) reference help for a fuller definition of the `nRings`, `panRingStart`, `nVertices`, and vertex fields.

```

psShape = SHPCreateObject( SHPT_POLYGON, -1, poRawPoly->nRings,
                           poRawPoly->panRingStart, NULL,
                           poRawPoly->nVertices,
                           poRawPoly->padfX,
                           poRawPoly->padfY,
                           poRawPoly->padfZ,
                           NULL );

```

Write Record Number

The following call is used to write out the record number of the polygon, fetched from the `SDTSIndexedFeature::o-ModId` data member. The `szModule` value in this data field will always match the module name for the whole layer.

While not shown here, there is also an szOBRP field on oModId which have different values depending on whether the polygon is a universe or regular polygon.

```
DBFWriteIntegerAttribute( hDBF, iShape, nSDTSRecordField,
                          poRawPoly->oModId.nRecord );
```

Fetch Associated User Records

In keeping with the setting up of the schema, accessing the user records is somewhat complicated. In sdts2shp, the primary attribute records associated with any feature (including SDTSRawPolygons) can be fetched with the WriteAttrRecordToDBF() function defined as follows.

In particular, the poFeature->nAttributes member indicates how many associated attribute records there are. The poFeature->aoATID[] array contains the [SDTSModId](#)'s for each record. This [SDTSModId](#) can be passed to [SDTSTransfer::GetAttr\(\)](#) to fetch the [DDFField](#) pointer for the user attributes. The WriteAttrRecordToDBF() method is specific to sdts2shp and will be define later.

```
int      iAttrRecord;

for( iAttrRecord = 0; iAttrRecord < poFeature->nAttributes; iAttrRecord++)
{
    DDFField      *poSR;

    poSR = poTransfer->GetAttr( poFeature->aoATID+iAttrRecord );

    WriteAttrRecordToDBF( hDBF, iRecord, poTransfer, poSR );
}
```

Write User Attributes

In a manner analogous to the definition of the fields from the prototype attribute record, the following code loops over the subfields, and fetches the data for each. The data extraction via poSR->GetSubfieldData() is a bit involved, and more information can be found on the [DDFField](#) reference page.

```
/* ----- */
/*      Process each subfield in the record.      */
/* ----- */
DDFFieldDefn      *poFDefn = poSR->GetFieldDefn();

for( int iSF=0; iSF < poFDefn->GetSubfieldCount(); iSF++ )
{
    DDFFSubfieldDefn *poSFDefn = poFDefn->GetSubfield( iSF );
    int      iField;
    int      nMaxBytes;
    const char *      pachData = poSR->GetSubfieldData(poSFDefn,
                                                        &nMaxBytes);

/* ----- */
/*      Identify the related DBF field, if any.      */
/* ----- */
    for( iField = 0; iField < hDBF->nFields; iField++ )
    {
        if( EQUALN(poSFDefn->GetName(),
                  hDBF->pszHeader+iField*32,10) )
            break;
    }
}
```

```

        if( iField == hDBF->nFields )
            iField = -1;

/* ----- */
/*      Handle each of the types.                  */
/* ----- */
switch( poSFDefn->GetType() )
{
    case DDFString:
        const char *pszValue;

        pszValue = poSFDefn->ExtractStringData(pachData, nMaxBytes,
                                                NULL);

        if( iField != -1 )
            DBFWriteStringAttribute(hDBF, iRecord, iField, pszValue );
        break;

    case DDFFloat:
        double dfValue;

        dfValue = poSFDefn->ExtractFloatData(pachData, nMaxBytes,
                                              NULL);

        if( iField != -1 )
            DBFWriteDoubleAttribute( hDBF, iRecord, iField, dfValue );
        break;

    case DDFInt:
        int nValue;

        nValue = poSFDefn->ExtractIntData(pachData, nMaxBytes, NULL);

        if( iField != -1 )
            DBFWriteIntegerAttribute( hDBF, iRecord, iField, nValue );
        break;

    default:
        break;
}
} /* next subfield */

```

Cleanup

In the case of `sdt2shp`, the [SDTSTransfer](#) is created on the stack. When it falls out of scope it is destroyed, and all the indexed readers, and their indexed features caches are also cleaned up.

Chapter 4

sdts2shp.cpp

<title>SDTS To Shape Example Application</title>

```
/* *****
 * $Id: sdts2shp.cpp 19952 2010-07-02 05:44:18Z warmerdam $
 *
 * Project: SDTS Translator
 * Purpose: Mainline for converting to ArcView Shapefiles.
 * Author: Frank Warmerdam, warmerdam@pobox.com
 *
 * *****
 * Copyright (c) 1999, Frank Warmerdam
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 * *****/

#include "sdts_al.h"
#include "shapefil.h"
#include "cpl_string.h"

CPL_CVSID("$Id: sdts2shp.cpp 19952 2010-07-02 05:44:18Z warmerdam $");

static int bVerbose = FALSE;

static void WriteLineShapefile( const char *, SDTSTransfer *,
                               const char * );
static void WritePointShapefile( const char *, SDTSTransfer *,
                                 const char * );
static void WriteAttributeDBF( const char *, SDTSTransfer *,
                              const char * );
static void WritePolygonShapefile( const char *, SDTSTransfer *,
                                   const char * );

static void
AddPrimaryAttrToDBFSchema( DBFHandle hDBF, SDTSTransfer * poTransfer,
                           char ** papszModuleList );

static void
WritePrimaryAttrToDBF( DBFHandle hDBF, int nRecord,
                      SDTSTransfer *, SDTSFeature * poFeature );

static void
WriteAttrRecordToDBF( DBFHandle hDBF, int nRecord,
                     SDTSTransfer *, DDFField * poAttributes );

/* *****
 * Usage()
 * *****/

static void Usage()
```

```

{
    printf( "Usage: sdts2shp CATD_filename [-o shapefile_name]\n"
           "          [-m module_name] [-v]\n"
           "\n"
           "Modules include 'LE01', 'PC01', 'NP01' and 'ARDF'\n" );
    exit( 1 );
}

/* *****
/*                               main()
/* *****

int main( int nArgc, char ** papszArgv )
{
    {
        int          i;
        const char   *pszCATDFilename = NULL;
        const char   *pszMODN = "LE01";
        char         *pszShapefile = "sdts_out.shp";
        SDTSTransfer oTransfer;

/* ----- */
/*      Interpret commandline switches.
/* ----- */
        if( nArgc < 2 )
            Usage();

        pszCATDFilename = papszArgv[1];

        for( i = 2; i < nArgc; i++ )
        {
            if( EQUAL(papszArgv[i], "-m") && i+1 < nArgc )
                pszMODN = papszArgv[i+1];
            else if( EQUAL(papszArgv[i], "-o") && i+1 < nArgc )
                pszShapefile = papszArgv[i+1];
            else if( EQUAL(papszArgv[i], "-v") )
                bVerbose = TRUE;
            else
            {
                printf( "Incomplete, or unsupported option '%s'\n\n",
                       papszArgv[i] );
                Usage();
            }
        }

/* ----- */
/*      Message shapefile name to have no extension.
/* ----- */
        pszShapefile = strdup( pszShapefile );
        for( i = strlen(pszShapefile)-1; i >= 0; i-- )
        {
            if( pszShapefile[i] == '.' )
            {
                pszShapefile[i] = '\0';
                break;
            }
            else if( pszShapefile[i] == '/' || pszShapefile[i] == '\\' )
                break;
        }

/* ----- */
/*      Open the transfer.
/* ----- */
        if( !oTransfer.Open( pszCATDFilename ) )
        {
            fprintf( stderr,
                    "Failed to read CATD file '%s'\n",
                    pszCATDFilename );
            exit( 100 );
        }

/* ----- */
/*      Dump available layer in verbose mode.
/* ----- */
        if( bVerbose )
        {
            printf( "Layers:\n" );
            for( i = 0; i < oTransfer.GetLayerCount(); i++ )
            {
                int          iCATDEntry = oTransfer.GetLayerCATDEntry(i);

                printf( "  %s: '%s'\n",
                        oTransfer.GetCATD() ->GetEntryModule(iCATDEntry),

```

```

        oTransfer.GetCATD()->GetEntryTypeDesc(iCATDEntry) );
    }
    printf( "\n" );
}

/* ----- */
/*      Check that module exists.                      */
/* ----- */
if( oTransfer.FindLayer( pszMODN ) == -1 )
{
    fprintf( stderr, "Unable to identify module: %s\n", pszMODN );
    exit( 1 );
}

/* ----- */
/*      If the module is an LE module, write it to an Arc file. */
/* ----- */
if( pszMODN[0] == 'L' || pszMODN[0] == 'l' )
{
    WriteLineShapefile( pszShapefile, &oTransfer, pszMODN );
}

/* ----- */
/*      If the module is an attribute primary one, dump to DBF. */
/* ----- */
else if( pszMODN[0] == 'A' || pszMODN[0] == 'a'
        || pszMODN[0] == 'B' || pszMODN[0] == 'b' )
{
    WriteAttributeDBF( pszShapefile, &oTransfer, pszMODN );
}

/* ----- */
/*      If the module is a point one, dump to Shapefile. */
/* ----- */
else if( pszMODN[0] == 'N' || pszMODN[0] == 'n' )
{
    WritePointShapefile( pszShapefile, &oTransfer, pszMODN );
}

/* ----- */
/*      If the module is a polygon one, dump to Shapefile. */
/* ----- */
else if( pszMODN[0] == 'P' || pszMODN[0] == 'p' )
{
    WritePolygonShapefile( pszShapefile, &oTransfer, pszMODN );
}

else
{
    fprintf( stderr, "Unrecognised module name: %s\n", pszMODN );
}

CPLFree( pszShapefile );
}
#ifdef DBMALLOC
    malloc_dump(1);
#endif
}

/* ***** */
/*      WriteLineShapefile()                      */
/* ***** */
static void WriteLineShapefile( const char * pszShapefile,
                               SDTSTransfer * poTransfer,
                               const char * pszMODN )
{
    SDTSLineReader *poLineReader;

/* ----- */
/*      Fetch a reference to the indexed Pointgon reader. */
/* ----- */
poLineReader = (SDTSLineReader *)
    poTransfer->GetLayerIndexedReader( poTransfer->
        FindLayer( pszMODN ) );

if( poLineReader == NULL )
{
    fprintf( stderr, "Failed to open %s.\n",
        poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
    return;
}

poLineReader->Rewind();

/* ----- */

```

```

/*      Create the Shapefile.      */
/* ----- */
    SHPHandle    hSHP;

    hSHP = SHPCreate( pszShapefile, SHPT_ARC );
    if( hSHP == NULL )
    {
        fprintf( stderr, "Unable to create shapefile '%s'\n",
                  pszShapefile );
        return;
    }

/* ----- */
/*      Create the database file, and our basic set of attributes.      */
/* ----- */
    DBFHandle    hDBF;
    int          nLeftPolyField, nRightPolyField;
    int          nStartNodeField, nEndNodeField, nSDTSRecordField;
    char         szDBFFilename[1024];

    sprintf( szDBFFilename, "%s.dbf", pszShapefile );

    hDBF = DBFCreate( szDBFFilename );
    if( hDBF == NULL )
    {
        fprintf( stderr, "Unable to create shapefile .dbf for '%s'\n",
                  pszShapefile );
        return;
    }

    nSDTSRecordField = DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );
    nLeftPolyField = DBFAddField( hDBF, "LeftPoly", FTString, 12, 0 );
    nRightPolyField = DBFAddField( hDBF, "RightPoly", FTString, 12, 0 );
    nStartNodeField = DBFAddField( hDBF, "StartNode", FTString, 12, 0 );
    nEndNodeField = DBFAddField( hDBF, "EndNode", FTString, 12, 0 );

    char **papszModRefs = poLineReader->ScanModuleReferences();
    AddPrimaryAttrToDBFSchema( hDBF, poTransfer, papszModRefs );
    CSLDestroy( papszModRefs );

/* ===== */
/*      Process all the line features in the module.      */
/* ===== */
    SDTSRawLine *poRawLine;

    while( (poRawLine = poLineReader->GetNextLine()) != NULL )
    {
        int          iShape;

/* ----- */
/*      Write out a shape with the vertices.      */
/* ----- */
        SHPObject    *psShape;

        psShape = SHPCreateSimpleObject( SHPT_ARC, poRawLine->nVertices,
                                         poRawLine->padfX, poRawLine->padfY,
                                         poRawLine->padfZ );

        iShape = SHPWriteObject( hSHP, -1, psShape );

        SHPDestroyObject( psShape );

/* ----- */
/*      Write out the attributes.      */
/* ----- */
        char         szID[13];

        DBFWriteIntegerAttribute( hDBF, iShape, nSDTSRecordField,
                                poRawLine->oModId.nRecord );

        sprintf( szID, "%s:%ld",
                  poRawLine->oLeftPoly.szModule,
                  poRawLine->oLeftPoly.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nLeftPolyField, szID );

        sprintf( szID, "%s:%ld",
                  poRawLine->oRightPoly.szModule,
                  poRawLine->oRightPoly.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nRightPolyField, szID );

        sprintf( szID, "%s:%ld",
                  poRawLine->oStartNode.szModule,
                  poRawLine->oStartNode.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nStartNodeField, szID );

        sprintf( szID, "%s:%ld",
                  poRawLine->oEndNode.szModule,

```

```

        poRawLine->oEndNode.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nEndNodeField, szID );

        WritePrimaryAttrToDBF( hDBF, iShape, poTransfer, poRawLine );

        if( !poLineReader->IsIndexed() )
            delete poRawLine;
    }

    /* ----- */
    /*      Close, and cleanup.      */
    /* ----- */
    DBFClose( hDBF );
    SHPClose( hSHP );
}

/* ----- */
/*      WritePointShapefile()      */
/* ----- */
static void WritePointShapefile( const char * pszShapefile,
                                SDTSTransfer * poTransfer,
                                const char * pszMODN )

{
    SDTSPointReader *poPointReader;

    /* ----- */
    /*      Fetch a reference to the indexed Pointgon reader.      */
    /* ----- */
    poPointReader = (SDTSPointReader *)
        poTransfer->GetLayerIndexedReader( poTransfer->
            FindLayer( pszMODN ) );

    if( poPointReader == NULL )
    {
        fprintf( stderr, "Failed to open %s.\n",
            poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
        return;
    }

    poPointReader->Rewind();

    /* ----- */
    /*      Create the Shapefile.      */
    /* ----- */
    SHPHandle hSHP;

    hSHP = SHPCreate( pszShapefile, SHPT_POINT );
    if( hSHP == NULL )
    {
        fprintf( stderr, "Unable to create shapefile '%s'\n",
            pszShapefile );
        return;
    }

    /* ----- */
    /*      Create the database file, and our basic set of attributes.      */
    /* ----- */
    DBFHandle hDBF;
    int nAreaField, nSDTSRecordField;
    char szDBFFilename[1024];

    sprintf( szDBFFilename, "%s.dbf", pszShapefile );

    hDBF = DBFCreate( szDBFFilename );
    if( hDBF == NULL )
    {
        fprintf( stderr, "Unable to create shapefile .dbf for '%s'\n",
            pszShapefile );
        return;
    }

    nSDTSRecordField = DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );
    nAreaField = DBFAddField( hDBF, "AreaId", FTString, 12, 0 );

    char **papszModRefs = poPointReader->ScanModuleReferences();
    AddPrimaryAttrToDBFSchema( hDBF, poTransfer, papszModRefs );
    CSLDestroy( papszModRefs );

    /* ===== */
    /*      Process all the line features in the module.      */
    /* ===== */
    SDTSRawPoint *poRawPoint;

    while( (poRawPoint = poPointReader->GetNextPoint()) != NULL )
    {

```

```

        int                iShape;

/* ----- */
/*      Write out a shape with the vertices.      */
/* ----- */
        SHPObject          *psShape;

        psShape = SHPCreateSimpleObject( SHPT_POINT, 1,
                                          &(poRawPoint->dfX),
                                          &(poRawPoint->dfY),
                                          &(poRawPoint->dfZ) );

        iShape = SHPWriteObject( hSHP, -1, psShape );

        SHPDestroyObject( psShape );

/* ----- */
/*      Write out the attributes.                  */
/* ----- */
        char              szID[13];

        DBFWriteIntegerAttribute( hDBF, iShape, nSDTSRecordField,
                                  poRawPoint->oModId.nRecord );

        sprintf( szID, "%s:%ld",
                  poRawPoint->oAreaId.szModule,
                  poRawPoint->oAreaId.nRecord );
        DBFWriteStringAttribute( hDBF, iShape, nAreaField, szID );

        WritePrimaryAttrToDBF( hDBF, iShape, poTransfer, poRawPoint );

        if( !poPointReader->IsIndexed() )
            delete poRawPoint;
    }

/* ----- */
/*      Close, and cleanup.                        */
/* ----- */
        DBFClose( hDBF );
        SHPClose( hSHP );
    }

/* ***** */
/*      WriteAttributeDBF()                        */
/* ***** */
static void WriteAttributeDBF( const char * pszShapefile,
                              SDTSTransfer * poTransfer,
                              const char * pszMODN )
{
    SDTSAttrReader          *poAttrReader;

/* ----- */
/*      Fetch a reference to the indexed Pointgon reader.      */
/* ----- */
    poAttrReader = (SDTSAttrReader *)
        poTransfer->GetLayerIndexedReader( poTransfer->
        FindLayer( pszMODN ) );

    if( poAttrReader == NULL )
    {
        fprintf( stderr, "Failed to open %s.\n",
                  poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
        return;
    }

    poAttrReader->Rewind();

/* ----- */
/*      Create the database file, and our basic set of attributes.      */
/* ----- */
        DBFHandle          hDBF;
        char                szDBFFilename[1024];

        sprintf( szDBFFilename, "%s.dbf", pszShapefile );

        hDBF = DBFCreate( szDBFFilename );
        if( hDBF == NULL )
        {
            fprintf( stderr, "Unable to create shapefile .dbf for '%s'\n",
                      pszShapefile );
            return;
        }

        DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );

```

```

/* ----- */
/* Prepare the schema. */
/* ----- */
char **papszMODNList = CSLAddString( NULL, pszMODN );

AddPrimaryAttrToDBFSchema( hDBF, poTransfer, papszMODNList );

CSLDestroy( papszMODNList );

/* ===== */
/* Process all the records in the module. */
/* ===== */
SDTSAttrRecord *poRecord;
int iRecord = 0;

while( (poRecord = (SDTSAttrRecord*)poAttrReader->
GetNextFeature()) != NULL )
{
    DBFWriteIntegerAttribute( hDBF, iRecord, 0,
                             poRecord->oModId.nRecord );

    WriteAttrRecordToDBF( hDBF, iRecord, poTransfer, poRecord->poATTR );

    if( !poAttrReader->IsIndexed() )
        delete poRecord;

    iRecord++;
}

/* ----- */
/* Close, and cleanup. */
/* ----- */
DBFClose( hDBF );
}

/* ===== */
/* WritePolygonShapefile() */
/* ===== */
static void WritePolygonShapefile( const char * pszShapefile,
                                   SDTSTransfer * poTransfer,
                                   const char * pszMODN )
{
    SDTSPolygonReader *poPolyReader;

/* ----- */
/* Fetch a reference to the indexed polygon reader. */
/* ----- */
poPolyReader = (SDTSPolygonReader *)
poTransfer->GetLayerIndexedReader( poTransfer->
FindLayer( pszMODN ) );

if( poPolyReader == NULL )
{
    fprintf( stderr, "Failed to open %s.\n",
             poTransfer->GetCATD()->GetModuleFilePath( pszMODN ) );
    return;
}

/* ----- */
/* Assemble polygon geometries from all the line layers. */
/* ----- */
poPolyReader->AssembleRings( poTransfer, poTransfer->FindLayer( pszMODN ) );

/* ----- */
/* Create the Shapefile. */
/* ----- */
SHPHandle hSHP;

hSHP = SHPCreate( pszShapefile, SHPT_POLYGON );
if( hSHP == NULL )
{
    fprintf( stderr, "Unable to create shapefile '%s'\n",
             pszShapefile );
    return;
}

/* ----- */
/* Create the database file, and our basic set of attributes. */
/* ----- */
DBFHandle hDBF;
int nSDTSRecordField;
char szDBFFilename[1024];

sprintf( szDBFFilename, "%s.dbf", pszShapefile );

```

```

hDBF = DBFCreate( szDBFFilename );
if( hDBF == NULL )
{
    fprintf( stderr, "Unable to create shapefile .dbf for '%s'\n",
             pszShapefile );
    return;
}

nSDTSRecordField = DBFAddField( hDBF, "SDTSRecId", FTInteger, 8, 0 );

char **papszModRefs = poPolyReader->ScanModuleReferences();
AddPrimaryAttrToDBFSchema( hDBF, poTransfer, papszModRefs );
CSLDestroy( papszModRefs );

/* ===== */
/*      Process all the polygon features in the module.      */
/* ===== */
SDTSRawPolygon *poRawPoly;

poPolyReader->Rewind();
while( (poRawPoly = (SDTSRawPolygon *) poPolyReader->
    GetNextFeature())
    != NULL )
{
    int iShape;

/* ----- */
/*      Write out a shape with the vertices.      */
/* ----- */
    SHPObject *psShape;

    psShape = SHPCreateObject( SHPT_POLYGON, -1, poRawPoly->nRings,
                              poRawPoly->panRingStart, NULL,
                              poRawPoly->nVertices,
                              poRawPoly->padfX,
                              poRawPoly->padfY,
                              poRawPoly->padfZ,
                              NULL );

    iShape = SHPWriteObject( hSHP, -1, psShape );

    SHPDestroyObject( psShape );

/* ----- */
/*      Write out the attributes.      */
/* ----- */
    DBFWriteIntegerAttribute( hDBF, iShape, nSDTSRecordField,
                              poRawPoly->oModId.nRecord );
    WritePrimaryAttrToDBF( hDBF, iShape, poTransfer, poRawPoly );

    if( !poPolyReader->IsIndexed() )
        delete poRawPoly;
}

/* ----- */
/*      Close, and cleanup.      */
/* ----- */
    DBFClose( hDBF );
    SHPClose( hSHP );
}

/* ===== */
/*      AddPrimaryAttrToDBF()      */
/* ===== */
/*      Add the fields from all the given primary attribute modules      */
/*      to the schema of the passed DBF file.      */
/* ===== */

static void
AddPrimaryAttrToDBFSchema( DBFHandle hDBF, SDTSTransfer *poTransfer,
                           char ** papszModuleList )
{
    for( int iModule = 0;
        papszModuleList != NULL && papszModuleList[iModule] != NULL;
        iModule++ )
    {
        SDTSAttrReader *poAttrReader;

/* ----- */
/*      Get a reader on the desired module.      */
/* ----- */
        poAttrReader = (SDTSAttrReader *)
            poTransfer->GetLayerIndexedReader(
                poTransfer->FindLayer( papszModuleList[iModule] ) );

        if( poAttrReader == NULL )

```

```

{
    printf( "Unable to open attribute module %s, skipping.\n" ,
            papszModuleList[iModule] );
    continue;
}

poAttrReader->Rewind();

/* ----- */
/*      Read the first record so we can clone schema information off      */
/*      of it.                                                              */
/* ----- */
SDTSAttrRecord *poAttrFeature;

poAttrFeature = (SDTSAttrRecord *) poAttrReader->
GetNextFeature();
if( poAttrFeature == NULL )
{
    fprintf( stderr,
            "Didn't find any meaningful attribute records in %s.\n",
            papszModuleList[iModule] );

    continue;
}

/* ----- */
/*      Clone schema off the first record.  Eventually we need to        */
/*      get the information out of the DDR record, but it isn't          */
/*      clear to me how to accomplish that with the SDTS++ API.          */
/* ----- */
/*      The following approach may fail (dramatically) if some            */
/*      records do not include all subfields.  Furthermore, no            */
/*      effort is made to make DBF field names unique.  The SDTS          */
/*      attributes often have names much beyond the 14 character dbf      */
/*      limit which may result in non-unique attributes.                  */
/* ----- */
DDFFieldDefn *poFDefn = poAttrFeature->poATTR->
GetFieldDefn();
int iSF;
DDFField *poSR = poAttrFeature->poATTR;

for( iSF=0; iSF < poFDefn->GetSubfieldCount(); iSF++ )
{
    DDFSFieldDefn *poSFDefn = poFDefn->GetSubfield( iSF );
    int nWidth = poSFDefn->GetWidth();

    switch( poSFDefn->GetType() )
    {
        case DDFString:
            if( nWidth == 0 )
            {
                int nMaxBytes;

                const char * pachData = poSR->GetSubfieldData(poSFDefn,
                                                                &nMaxBytes);

                nWidth = strlen(poSFDefn->ExtractStringData(pachData,
                                                                nMaxBytes, NULL ));
            }

            DBFAddField( hDBF, poSFDefn->GetName(), FTString, nWidth, 0 );
            break;

        case DDFInt:
            if( nWidth == 0 )
                nWidth = 9;

            DBFAddField( hDBF, poSFDefn->GetName(), FTInteger, nWidth, 0 );
            break;

        case DDFFloat:
            DBFAddField( hDBF, poSFDefn->GetName(), FTDouble, 18, 6 );
            break;

        default:
            fprintf( stderr,
                    "Dropping attribute '%s' of module '%s'.  "
                    "Type unsupported\n",
                    poSFDefn->GetName(),
                    papszModuleList[iModule] );

            break;
    }
}

if( !poAttrReader->IsIndexed() )
    delete poAttrFeature;

```

```

    } /* next module */
}

/* *****
/*      WritePrimaryAttrToDBF()
/* *****

static void
WritePrimaryAttrToDBF( DBFHandle hDBF, int iRecord,
                      SDTSTransfer * poTransfer, SDTSFeature * poFeature )

{
/* =====
/*      Loop over all the attribute records linked to this feature.
/* =====
    int          iAttrRecord;

    for( iAttrRecord = 0; iAttrRecord < poFeature->nAttributes; iAttrRecord++)
    {
        DDFField      *poSR;

        poSR = poTransfer->GetAttr( poFeature->paoATID+iAttrRecord );

        WriteAttrRecordToDBF( hDBF, iRecord, poTransfer, poSR );
    }
}

/* *****
/*      WriteAttrRecordToDBF()
/* *****

static void
WriteAttrRecordToDBF( DBFHandle hDBF, int iRecord,
                      SDTSTransfer * poTransfer, DDFField * poSR )

{
/* -----
/*      Process each subfield in the record.
/* -----
    DDFFieldDefn      *poFDefn = poSR->GetFieldDefn();

    for( int iSF=0; iSF < poFDefn->GetSubfieldCount(); iSF++ )
    {
        DDFFSubfieldDefn *poSFDefn = poFDefn->GetSubfield( iSF );
        int              iField;
        int              nMaxBytes;
        const char *      pachData = poSR->GetSubfieldData( poSFDefn,
                                                            &nMaxBytes);

/* -----
/*      Identify the related DBF field, if any.
/* -----
        for( iField = 0; iField < hDBF->nFields; iField++ )
        {
            if( EQUALN(poSFDefn->GetName(),
                      hDBF->pszHeader+iField*32,10) )
                break;
        }

        if( iField == hDBF->nFields )
            iField = -1;

/* -----
/*      Handle each of the types.
/* -----
        switch( poSFDefn->GetType() )
        {
            case DDFSString:
                const char *pszValue;

                pszValue = poSFDefn->ExtractStringData( pachData, nMaxBytes,
                                                         NULL);

                if( iField != -1 )
                    DBFWriteStringAttribute( hDBF, iRecord, iField, pszValue );
                break;

            case DDFFloat:
                double      dfValue;

                dfValue = poSFDefn->ExtractFloatData( pachData, nMaxBytes,
                                                         NULL);

                if( iField != -1 )
                    DBFWriteDoubleAttribute( hDBF, iRecord, iField, dfValue );
                break;

```

```
case DDFInt:
    int        nValue;

    nValue = posFDefn->ExtractIntData(pachData, nMaxBytes, NULL);

    if( iField != -1 )
        DBFWriteIntegerAttribute( hDBF, iRecord, iField, nValue );
    break;

default:
    break;
}
} /* next subfield */
}
```


Chapter 5

ISO8211_Example

```
/* *****
 * $Id: 8211view.cpp 10645 2007-01-18 02:22:39Z warmerdam $
 *
 * Project:   SDTS Translator
 * Purpose:   Example program dumping data in 8211 data to stdout.
 * Author:    Frank Warmerdam, warmerdam@pobox.com
 *
 * *****
 * Copyright (c) 1999, Frank Warmerdam
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 * *****/

#include <stdio.h>
#include "iso8211.h"

CPL_CVSID("$Id: 8211view.cpp 10645 2007-01-18 02:22:39Z warmerdam $");

static void ViewRecordField( DDFField * poField );
static int ViewSubfield( DDFSFieldDefn *poSFDefn,
                        const char * pachFieldData,
                        int nBytesRemaining );

/* *****
 *                               main()
 * *****
 */

int main( int nArgc, char ** papszArgv )
{
    DDFModule    oModule;
    const char   *pszFilename = NULL;
    int          bFSPTHack = FALSE;

    for( int iArg = 1; iArg < nArgc; iArg++ )
    {
        if( EQUAL(papszArgv[iArg], "-fspt_repeating") )
            bFSPTHack = TRUE;
        else
            pszFilename = papszArgv[iArg];
    }

    if( pszFilename == NULL )
    {
        printf( "Usage: 8211view filename\n" );
        exit( 1 );
    }
}
```

```

/* ----- */
/*      Open the file. Note that by default errors are reported to      */
/*      stderr, so we don't bother doing it ourselves.                  */
/* ----- */
if( !oModule.Open( pszFilename ) )
{
    exit( 1 );
}

if( bFSPTHack )
{
    DDFFieldDefn *poFSPT = oModule.FindFieldDefn( "FSPT" );

    if( poFSPT == NULL )
        fprintf( stderr,
            "unable to find FSPT field to set repeating flag.\n" );
    else
        poFSPT->SetRepeatingFlag( TRUE );
}

/* ----- */
/*      Loop reading records till there are none left.                  */
/* ----- */
DDFRecord    *poRecord;
int          iRecord = 0;

while( (poRecord = oModule.ReadRecord()) != NULL )
{
    printf( "Record %d (%d bytes)\n",
        ++iRecord, poRecord->GetDataSize() );

    /* ----- */
    /*      Loop over each field in this particular record.              */
    /* ----- */
    for( int iField = 0; iField < poRecord->GetFieldCount(); iField++ )
    {
        DDFField    *poField = poRecord->GetField( iField );

        ViewRecordField( poField );
    }
}

/* *****/
/*      ViewRecordField()                                              */
/* *****/
/*      Dump the contents of a field instance in a record.          */
/* *****/
static void ViewRecordField( DDFField * poField )
{
    int          nBytesRemaining;
    const char  *pachFieldData;
    DDFFieldDefn *poFieldDefn = poField->GetFieldDefn();

    // Report general information about the field.
    printf( "      Field %s: %s\n",
        poFieldDefn->GetName(), poFieldDefn->GetDescription() );

    // Get pointer to this fields raw data. We will move through
    // it consuming data as we report subfield values.

    pachFieldData = poField->GetData();
    nBytesRemaining = poField->GetDataSize();

    /* ----- */
    /*      Loop over the repeat count for this fields                  */
    /*      subfields. The repeat count will almost                    */
    /*      always be one.                                              */
    /* ----- */
    int          iRepeat;

    for( iRepeat = 0; iRepeat < poField->GetRepeatCount(); iRepeat++ )
    {
        /* ----- */
        /*      Loop over all the subfields of this field, advancing    */
        /*      the data pointer as we consume data.                    */
        /* ----- */
        int          iSF;

        for( iSF = 0; iSF < poFieldDefn->GetSubfieldCount(); iSF++ )
        {
            DDFSubfieldDefn *poSFDefn = poFieldDefn->GetSubfield( iSF );
            int          nBytesConsumed;

```

```

        nBytesConsumed = ViewSubfield( poSFDefn, pachFieldData,
                                         nBytesRemaining );

        nBytesRemaining -= nBytesConsumed;
        pachFieldData += nBytesConsumed;
    }
}

/* *****
/*                               ViewSubfield()                               */
/* *****

static int ViewSubfield( DDFSubfieldDefn *poSFDefn,
                        const char * pachFieldData,
                        int nBytesRemaining )

{
    int          nBytesConsumed = 0;

    switch( poSFDefn->GetType() )
    {
        case DDFInt:
            if( poSFDefn->GetBinaryFormat() == DDFSubfieldDefn::UInt )
                printf( "      %s = %u\n",
                        poSFDefn->GetName(),
                        poSFDefn->ExtractIntData( pachFieldData, nBytesRemaining,
                                                  &nBytesConsumed ) );
            else
                printf( "      %s = %d\n",
                        poSFDefn->GetName(),
                        poSFDefn->ExtractIntData( pachFieldData, nBytesRemaining,
                                                  &nBytesConsumed ) );
            break;

        case DDFFloat:
            printf( "      %s = %f\n",
                    poSFDefn->GetName(),
                    poSFDefn->ExtractFloatData( pachFieldData, nBytesRemaining,
                                                  &nBytesConsumed ) );
            break;

        case DDFString:
            printf( "      %s = '%s'\n",
                    poSFDefn->GetName(),
                    poSFDefn->ExtractStringData( pachFieldData, nBytesRemaining,
                                                  &nBytesConsumed ) );
            break;

        case DDFBinaryString:
        {
            int i;
            //rjensen 19-Feb-2002 5 integer variables to decode NAME and LNAM
            int vrid_rcnm=0;
            int vrid_rcid=0;
            int foid_agen=0;
            int foid_find=0;
            int foid_fids=0;

            GByte *pabyBString = (GByte *)
                poSFDefn->ExtractStringData( pachFieldData, nBytesRemaining,
                                             &nBytesConsumed );

            printf( "      %s = 0x", poSFDefn->GetName() );
            for( i = 0; i < MIN(nBytesConsumed,24); i++ )
                printf( "%02X", pabyBString[i] );

            if( nBytesConsumed > 24 )
                printf( "%s", "... " );

            // rjensen 19-Feb-2002 S57 quick hack. decode NAME and LNAM bitfields
            if ( EQUAL(poSFDefn->GetName(),"NAME") )
            {
                vrid_rcnm=pabyBString[0];
                vrid_rcid=pabyBString[1] + (pabyBString[2]*256)+
                    (pabyBString[3]*65536)+ (pabyBString[4]*16777216);
                printf("\tVRID RCNM = %d,RCID = %u,vrid_rcnm,vrid_rcid);
            }
            else if ( EQUAL(poSFDefn->GetName(),"LNAM") )
            {
                foid_agen=pabyBString[0] + (pabyBString[1]*256);
                foid_find=pabyBString[2] + (pabyBString[3]*256)+
                    (pabyBString[4]*65536)+ (pabyBString[5]*16777216);
                foid_fids=pabyBString[6] + (pabyBString[7]*256);
                printf("\tFOID AGEN = %u,FIDN = %u,FIDS = %u",
                    foid_agen,foid_find,foid_fids);
            }
        }
    }
}

```

```
        }  
        printf( "\n" );  
    }  
    break;  
}  
  
return nBytesConsumed;  
}
```

Chapter 6

Hierarchical Index

6.1 Class Hierarchy

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

_CPLHashSet	??
_CPLList	??
_CPLQuadTree	??
_CPLSpawnedProcess	??
_MutexLinkedElt	??
_QuadTreeNode	??
CPLErrorContext	??
CPLHTTPResult	??
CPLKeywordParser	??
CPLMimePart	??
CPLODBCDriverInstaller	??
CPLODBCSession	??
CPLODBCStatement	??
CPLRectObj	??
CPLSharedFileInfo	??
CPLSharedFileInfoExtra	??
CPLStdCallThreadInfo	??
CPLXMLNode	??
CPLZip	??
ctb	??
curfile_info	??
DDFField	??
DDFFieldDefn	??
DDFModule	??
DDFRecord	??
DDFSubfieldDefn	??
DefaultCSVFileNameTLS	??
errHandler	??
file_in_zip_read_info_s	??
FindFileTLS	??
GDALPamDataset	
SDTSDataset	??
GDALPamRasterBand	
SDTSRasterBand	??
GDALScaledProgressInfo	??
GZipSnapshot	??
linkedlist_data_s	??
linkedlist_datablock_internal_s	??

VSI SparseFileHandle	??
VSI StdinHandle	??
VSI StdoutHandle	??
VSI StdoutRedirectHandle	??
VSI SubFileHandle	??
VSI UnixStdioHandle	??
VSI ZipWriteHandle	??
zip_fileinfo	??
zip_internal	??
zlib_filefunc_def_s	??

Chapter 7

Class Index

7.1 Class List

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

_CPLHashSet	??
_CPLList	??
_CPLQuadTree	??
_CPLSpawnedProcess	??
_MutexLinkedElt	??
_QuadTreeNode	??
CPLErrorContext	??
CPLHTTPResult	??
CPLKeywordParser	??
CPLMimePart	??
CPLODBCDriverInstaller	??
CPLODBCSession	??
CPLODBCStatement	??
CPLRectObj	??
CPLSharedFileInfo	??
CPLSharedFileInfoExtra	??
CPLStdCallThreadInfo	??
CPLXMLNode	??
CPLZip	??
ctb	??
curfile_info	??
DDFField	??
DDFFieldDefn	??
DDFModule	??
DDFRecord	??
DDFSubfieldDefn	??
DefaultCSVFileNameTLS	??
errHandler	??
file_in_zip_read_info_s	??
FindFileTLS	??
GDALScaledProgressInfo	??
GZipSnapshot	??
linkedList_data_s	??
linkedList_datablock_internal_s	??
ParseContext	??
SDTS_CATD	??
SDTS_CATDentry	??
SDTS_IREF	??

SDTS_XREF	??
SDTSAttrReader	??
SDTSAttrRecord	??
SDTSDataset	??
SDTSFeature	??
SDTSIndexedReader	??
SDTSLineReader	??
SDTSModId	??
SDTSPointReader	??
SDTSPolygonReader	??
SDTSRasterBand	??
SDTSRasterReader	??
SDTSRawLine	??
SDTSRawPoint	??
SDTSRawPolygon	??
SDSTTransfer	??
SFRegion	??
StackContext	??
tm_unz_s	??
tm_zip_s	??
unz_file_info_internal_s	??
unz_file_info_s	??
unz_file_pos_s	??
unz_global_info_s	??
unz_s	??
VSIArchiveContent	??
VSIArchiveEntry	??
VSIArchiveEntryFileOffset	??
VSIArchiveFilesystemHandler	??
VSIArchiveReader	??
VSIBufferedReaderHandle	??
VSIChunk	??
VSIFileManager	??
VSIFilesystemHandler	??
VSIZipFilesystemHandler	??
VSIZipHandle	??
VSIZipWriteHandle	??
VSIMemFile	??
VSIMemFilesystemHandler	??
VSIMemHandle	??
VSIReadDirRecursiveTask	??
VSIReadOnlyFilesystemHandler	??
VSIReadOnlyHandle	??
VSIStdinFilesystemHandler	??
VSIStdinHandle	??
VSIStdoutFilesystemHandler	??
VSIStdoutHandle	??
VSIStdoutRedirectFilesystemHandler	??
VSIStdoutRedirectHandle	??
VSISubFileFilesystemHandler	??
VSISubFileHandle	??
VSIEntryFileOffset	??
VSIFilesystemHandler	??
VSIReader	??
VSIUnixStdioFilesystemHandler	??
VSIUnixStdioHandle	??
VSVirtualHandle	??

VSZipEntryFileOffset	??
VSZipFilesystemHandler	??
VSZipReader	??
VSZipWriteHandle	??
zip_fileinfo	??
zip_internal	??
zlib_filefunc_def_s	??

Chapter 8

File Index

8.1 File List

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

cpl_atomic_ops.h	??
cpl_config.h	??
cpl_config_extras.h	??
cpl_conv.h	??
cpl_csv.h	??
cpl_error.h	??
cpl_hash_set.h	??
cpl_http.h	??
cpl_list.h	??
cpl_minixml.h	??
cpl_minizip_ioapi.h	??
cpl_minizip_unzip.h	??
cpl_minizip_zip.h	??
cpl_multiproc.h	??
cpl_odbc.h	??
cpl_port.h	??
cpl_progress.h	??
cpl_quad_tree.h	??
cpl_spawn.h	??
cpl_string.h	??
cpl_time.h	??
cpl_vsi.h	??
cpl_vsi_virtual.h	??
cpl_vsil_curl_priv.h	??
cpl_win32ce_api.h	??
cpl_wince.h	??
cplkeywordparser.h	??
gdal_csv.h	??
iso8211.h	??
sdts_al.h	??
sdtsdataset.cpp	??

Chapter 9

Class Documentation

9.1 _CPLHashSet Struct Reference

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

- [cpl_hash_set.cpp](#)

9.2 _CPLList Struct Reference

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

Public Attributes

- void * [pData](#)
- struct _CPLList * [psNext](#)

9.2.1 Detailed Description

List element structure.

9.2.2 Member Data Documentation

9.2.2.1 void* _CPLList::pData

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

9.2.2.2 struct _CPLList* _CPLList::psNext

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

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

- [cpl_list.h](#)

9.3 `_CPLQuadTree` Struct Reference

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

- `cpl_quad_tree.cpp`

9.4 `_CPLSpawnedProcess` Struct Reference

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

- `cpl_spawn.cpp`

9.5 `_MutexLinkedElt` Struct Reference

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

- `cpl_multiproc.cpp`

9.6 `_QuadTreeNode` Struct Reference

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

- `cpl_quad_tree.cpp`

9.7 `CPLErrorContext` Struct Reference

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

- `cpl_error.cpp`

9.8 `CPLHTTPResult` Struct Reference

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

Public Attributes

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

9.8.1 Detailed Description

Describe the result of a `CPLHTTPFetch()` call

9.8.2 Member Data Documentation

9.8.2.1 int CPLHTTPResult::nDataLen

Length of the pabyData buffer

9.8.2.2 int CPLHTTPResult::nMimePartCount

Number of parts in a multipart message

9.8.2.3 int CPLHTTPResult::nStatus

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

9.8.2.4 GByte* CPLHTTPResult::pabyData

Buffer with downloaded data

9.8.2.5 char** CPLHTTPResult::papszHeaders

Headers returned

9.8.2.6 CPLMimePart* CPLHTTPResult::pasMimePart

Array of parts (resolved by [CPLHTTPParseMultipartMime\(\)](#))

9.8.2.7 char* CPLHTTPResult::pszContentType

Content-Type of the response

9.8.2.8 char* CPLHTTPResult::pszErrBuf

Error message from curl, or NULL

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

- [cpl_http.h](#)

9.9 CPLKeywordParser Class Reference

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

- cplkeywordparser.h
- cplkeywordparser.cpp

9.10 CPLMimePart Struct Reference

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

Public Attributes

- char ** [papszHeaders](#)
- GByte * [pabyData](#)
- int [nDataLen](#)

9.10.1 Detailed Description

Describe a part of a multipart message

9.10.2 Member Data Documentation

9.10.2.1 int CPLMimePart::nDataLen

Buffer length

9.10.2.2 GByte* CPLMimePart::pabyData

Buffer with data of the part

9.10.2.3 char** CPLMimePart::papszHeaders

NULL terminated array of headers

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

- [cpl_http.h](#)

9.11 CPODBCDriverInstaller Class Reference

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

Public Member Functions

- int [InstallDriver](#) (const char *pszDriver, const char *pszPathIn, WORD fRequest=ODBC_INSTALL_COMPLETE)
- int [RemoveDriver](#) (const char *pszDriverName, int fRemovedDSN=FALSE)

9.11.1 Detailed Description

A class providing functions to install or remove ODBC driver.

9.11.2 Member Function Documentation

9.11.2.1 int CPODBCDriverInstaller::InstallDriver (const char * *pszDriver*, const char * *pszPathIn*, WORD *fRequest* = ODBC_INSTALLCOMPLETE)

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

Parameters

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

Returns

TRUE indicates success, FALSE if it fails.

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

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

Parameters

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

Returns

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

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

- [cpl_odbc.h](#)
- [cpl_odbc.cpp](#)

9.12 CPLODBCSession Class Reference

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

Public Member Functions

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

9.12.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

9.12.2 Member Function Documentation

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

Connect to database and logon.

Parameters

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

Returns

TRUE on success or FALSE on failure. Call [GetLastError\(\)](#) to get details on failure.

9.12.2.2 `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.

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

- [cpl_odbc.h](#)
- [cpl_odbc.cpp](#)

9.13 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 *,...) CPL_PRINT_FUNC_FORMAT(2)
- int [ExecuteSQL](#) (const char *=NULL)
- 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 *=NULL)
- const char * [GetColData](#) (const char *, const char *=NULL)
- int [GetColumns](#) (const char *pszTable, const char *pszCatalog=NULL, const char *pszSchema=NULL)
- int [GetPrimaryKeys](#) (const char *pszTable, const char *pszCatalog=NULL, const char *pszSchema=NULL)
- int [GetTables](#) (const char *pszCatalog=NULL, const char *pszSchema=NULL)
- void [DumpResult](#) (FILE *fp, int bShowSchema=FALSE)

Static Public Member Functions

- static CPLString [GetTypeName](#) (int)
- static SQLSMALLINT [GetTypeMapping](#) (SQLSMALLINT)

9.13.1 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.

9.13.2 Member Function Documentation

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

Append text to internal command.

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

Parameters

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

9.13.2.2 void CPODBCStatement::Append (int *nValue*)

Append to internal command.

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

Parameters

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

9.13.2.3 void CPODBCStatement::Append (double *dfValue*)

Append to internal command.

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

Parameters

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

9.13.2.4 void CPODBCStatement::AppendEscaped (const char * *pszText*)

Append text to internal command.

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

Parameters

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

9.13.2.5 int CPODBCStatement::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\(\)](#) methods.

Parameters

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

Returns

FALSE if formatting fails due to result being too large.

9.13.2.6 void CPODBCStatement::Clear ()

Clear internal command text and result set definitions.

9.13.2.7 void CPODBCStatement::DumpResult (FILE * *fp*, int *bShowSchema* = FALSE)

Dump resultset to file.

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

Parameters

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

9.13.2.8 int CPODBCStatement::ExecuteSQL (const char * *pszStatement* = NULL)

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\(\)](#) or [Appendf\(\)](#) calls will be used.

Parameters

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

Returns

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

9.13.2.9 int CPODBCStatement::Fetch (int *nOrientation* = SQL_FETCH_NEXT, int *nOffset* = 0)

Fetch a new record.

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

Currently it isn't clear how to determine whether an error or a normal out of data condition has occurred if [Fetch\(\)](#) fails.

Parameters

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

Returns

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

9.13.2.10 int CPLODBCStatement::GetColCount ()

Fetch the resultset column count.

Returns

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

9.13.2.11 const char * CPLODBCStatement::GetColData (int iCol, const char * pszDefault = NULL)

Fetch column data.

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

Parameters

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

Returns

pointer to internal column data or NULL on failure.

9.13.2.12 const char * CPLODBCStatement::GetColData (const char * pszColName, const char * pszDefault = NULL)

Fetch column data.

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

Parameters

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

Returns

pointer to internal column data or NULL on failure.

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

Fetch column index.

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

Parameters

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

Returns

the column index, or -1 if not found.

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

Fetch a column name.

Parameters

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

Returns

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

9.13.2.15 `short CPODBCStatement::GetColNullable ( int iCol )`

Fetch the column nullability.

Parameters

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

Returns

TRUE if the column may contains or FALSE otherwise.

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

Fetch the column precision.

Parameters

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

Returns

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

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

Fetch the column width.

Parameters

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

Returns

column width, zero for unknown width columns.

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

Fetch a column data type.

The return type code is a an ODBC SQL_ code, one of SQL_UNKNOWN_TYPE, SQL_CHAR, SQL_NUMERIC, SQL_DECIMAL, SQL_INTEGER, SQL_SMALLINT, SQL_FLOAT, SQL_REAL, SQL_DOUBLE, SQL_DATETIME, SQL_VARCHAR, SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP.

Parameters

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

Returns

type code or -1 if the column is illegal.

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

Fetch a column data type name.

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

Parameters

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

Returns

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

9.13.2.20 int CPODBCStatement::GetColumns (const char * *pszTable*, const char * *pszCatalog* = NULL, const char * *pszSchema* = NULL)

Fetch column definitions for a table.

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

Parameters

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

Returns

TRUE on success or FALSE on failure.

9.13.2.21 `int CPODBCStatement::GetPrimaryKeys ( const char * pszTable, const char * pszCatalog = NULL, const char * pszSchema = NULL )`

Fetch primary keys for a table.

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

Parameters

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

Returns

TRUE on success or FALSE on failure.

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

Fetch tables in database.

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

Parameters

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

Returns

TRUE on success or FALSE on failure.

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

Get appropriate C data type for SQL column type.

Returns a C data type code, corresponding to the indicated SQL data type code (as returned from [CPODBCStatement::GetColType\(\)](#)).

Parameters

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

Returns

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

9.13.2.24 CPLString CPODBCStatement::GetTypeName (int *nTypeCode*) [static]

Get name for SQL column type.

Returns a string name for the indicated type code (as returned from [CPODBCStatement::GetColType\(\)](#)).

Parameters

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

Returns

internal string, "UNKNOWN" if code not recognised.

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

- [cpl_odbc.h](#)
- [cpl_odbc.cpp](#)

9.14 CPLRectObj Struct Reference

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

- [cpl_quad_tree.h](#)

9.15 CPLSharedFileInfo Struct Reference

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

- [cpl_conv.h](#)

9.16 CPLSharedFileInfoExtra Struct Reference

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

- [cpl_conv.cpp](#)

9.17 CPLStdCallThreadInfo Struct Reference

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

- [cpl_multiproc.cpp](#)

9.18 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.

9.18.1 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\(\)](#). 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](#) structures.

9.18.2 Member Data Documentation

9.18.2.1 [CPLXMLNodeType CPLXMLNode::eType](#)

Node type.

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

9.18.2.2 struct [CPLXMLNode](#)* [CPLXMLNode::psChild](#)

Child node.

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

9.18.2.3 struct [CPLXMLNode](#)* [CPLXMLNode::psNext](#)

Next sibling.

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

9.18.2.4 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 "‐>").

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

- [cpl_minixml.h](#)

9.19 CPLZip Struct Reference

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

- [cpl_minizip_zip.cpp](#)

9.20 ctb Struct Reference

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

- [cpl_csv.cpp](#)

9.21 curfile_info Struct Reference

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

- [cpl_minizip_zip.cpp](#)

9.22 DDFField Class Reference

```
#include <iso8211.h>
```

Public Member Functions

- void [Dump](#) (FILE *fp)
- const char * [GetSubfieldData](#) (DDFSubfieldDefn *, int *p=NULL, int=0)
- const char * [GetInstanceData](#) (int nIndex, int *pnSize)
- const char * [GetData](#) ()
- int [GetDataSize](#) ()
- int [GetRepeatCount](#) ()
- DDFFieldDefn * [GetFieldDefn](#) ()

9.22.1 Detailed Description

This object represents one field in a [DDFRecord](#). This models an instance of the fields data, rather than it's data definition which is handled by the [DDFFieldDefn](#) class. Note that a [DDFField](#) doesn't have DDFSubfield children as you would expect. To extract subfield values use [GetSubfieldData\(\)](#) to find the right data pointer and then use [ExtractIntData\(\)](#), [ExtractFloatData\(\)](#) or [ExtractStringData\(\)](#).

9.22.2 Member Function Documentation

9.22.2.1 void DDFField::Dump (FILE * *fp*)

Write out field contents to debugging file.

A variety of information about this field, and all it's subfields is written to the given debugging file handle. Note that field definition information (ala [DDFFieldDefn](#)) isn't written.

Parameters

<i>fp</i>	The standard io file handle to write to. ie. stderr
-----------	---

9.22.2.2 const char* DDFField::GetData () [inline]

Return the pointer to the entire data block for this record. This is an internal copy, and shouldn't be freed by the application.

9.22.2.3 int DDFField::GetDataSize () [inline]

Return the number of bytes in the data block returned by [GetData\(\)](#).

9.22.2.4 DDFFieldDefn* DDFField::GetFieldDefn () [inline]

Fetch the corresponding [DDFFieldDefn](#).

9.22.2.5 const char * DDFField::GetInstanceData (int *nInstance*, int * *pnInstanceSize*)

Get field instance data and size.

The returned data pointer and size values are suitable for use with [DDFRecord::SetFieldRaw\(\)](#).

Parameters

<i>nInstance</i>	a value from 0 to GetRepeatCount() -1.
<i>pnInstanceSize</i>	a location to put the size (in bytes) of the field instance data returned. This size will include the unit terminator (if any), but not the field terminator. This size pointer may be NULL if not needed.

Returns

the data pointer, or NULL on error.

9.22.2.6 int DDFField::GetRepeatCount ()

How many times do the subfields of this record repeat? This will always be one for non-repeating fields.

Returns

The number of times that the subfields of this record occur in this record. This will be one for non-repeating fields.

See Also

[8211view example program](#) for demonstation of handling repeated fields properly.

9.22.2.7 `const char * DDFField::GetSubfieldData ( DDFFieldDefn * poSFDefn, int * pnMaxBytes = NULL, int iSubfieldIndex = 0 )`

Fetch raw data pointer for a particular subfield of this field.

The passed [DDFFieldDefn](#) (poSFDefn) should be acquired from the [DDFFieldDefn](#) corresponding with this field. This is normally done once before reading any records. This method involves a series of calls to [DDFFieldDefn::GetSubfieldDataLength\(\)](#) in order to track through the [DDFField](#) data to that belonging to the requested subfield. This can be relatively expensive.

Parameters

<i>poSFDefn</i>	The definition of the subfield for which the raw data pointer is desired.
<i>pnMaxBytes</i>	The maximum number of bytes that can be accessed from the returned data pointer is placed in this int, unless it is NULL.
<i>iSubfieldIndex</i>	The instance of this subfield to fetch. Use zero (the default) for the first instance.

Returns

A pointer into the [DDFField](#)'s data that belongs to the subfield. This returned pointer is invalidated by the next record read ([DDFRecord::ReadRecord\(\)](#)) and the returned pointer should not be freed by the application.

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

- iso8211.h
- ddffield.cpp

9.23 DDFFieldDefn Class Reference

```
#include <iso8211.h>
```

Public Member Functions

- void [Dump](#) (FILE *fp)
- const char * [GetName](#) ()
- const char * [GetDescription](#) ()
- int [GetSubfieldCount](#) ()
- [DDFFieldDefn](#) * [GetSubfield](#) (int i)
- [DDFFieldDefn](#) * [FindSubfieldDefn](#) (const char *)
- int [GetFixedWidth](#) ()
- int [IsRepeating](#) ()
- void [SetRepeatingFlag](#) (int n)
- char * [GetDefaultValue](#) (int *pnSize)

9.23.1 Detailed Description

Information from the DDR defining one field. Note that just because a field is defined for a [DDFModule](#) doesn't mean that it actually occurs on any records in the module. [DDFFieldDefns](#) are normally just significant as containers of the [DDFFieldDefns](#).

9.23.2 Member Function Documentation

9.23.2.1 void DDFFieldDefn::Dump (FILE * *fp*)

Write out field definition info to debugging file.

A variety of information about this field definition, and all it's subfields is written to the give debugging file handle.

Parameters

<i>fp</i>	The standard io file handle to write to. ie. stderr
-----------	---

9.23.2.2 DDFFSubfieldDefn * DDFFieldDefn::FindSubfieldDefn (const char * *pszMnemonic*)

Find a subfield definition by it's mnemonic tag.

Parameters

<i>pszMnemonic</i>	The name of the field.
--------------------	------------------------

Returns

The subfield pointer, or NULL if there isn't any such subfield.

9.23.2.3 char * DDFFieldDefn::GetDefaultValue (int * *pnSize*)

Return default data for field instance.

9.23.2.4 const char* DDFFieldDefn::GetDescription () [inline]

Fetch a longer descriptio of this field.

Returns

this is an internal copy and shouldn't be freed.

9.23.2.5 int DDFFieldDefn::GetFixedWidth () [inline]

Get the width of this field. This function isn't normally used by applications.

Returns

The width of the field in bytes, or zero if the field is not apparently of a fixed width.

9.23.2.6 const char* DDFFieldDefn::GetName () [inline]

Fetch a pointer to the field name (tag).

Returns

this is an internal copy and shouldn't be freed.

9.23.2.7 DDFSubfieldDefn * DDFFieldDefn::GetSubfield (int *i*)

Fetch a subfield by index.

Parameters

<i>i</i>	The index subfield index. (Between 0 and GetSubfieldCount()-1)
----------	---

Returns

The subfield pointer, or NULL if the index is out of range.

9.23.2.8 int DDFFieldDefn::GetSubfieldCount () [inline]

Get the number of subfields.

9.23.2.9 int DDFFieldDefn::IsRepeating () [inline]

Fetch repeating flag.

See Also

[DDFField::GetRepeatCount\(\)](#)

Returns

TRUE if the field is marked as repeating.

9.23.2.10 void DDFFieldDefn::SetRepeatingFlag (int *n*) [inline]

this is just for an S-57 hack for swedish data

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

- iso8211.h
- ddffielddefn.cpp

9.24 DDFModule Class Reference

```
#include <iso8211.h>
```

Public Member Functions

- [DDFModule](#) ()
- [~DDFModule](#) ()
- int [Open](#) (const char *pszFilename, int bFailQuietly=FALSE)
- void [Close](#) ()
- void [Dump](#) (FILE *fp)
- [DDFRecord](#) * [ReadRecord](#) (void)
- void [Rewind](#) (long nOffset=-1)
- [DDFFieldDefn](#) * [FindFieldDefn](#) (const char *)
- int [GetFieldCount](#) ()
- [DDFFieldDefn](#) * [GetField](#) (int)
- void [AddField](#) ([DDFFieldDefn](#) *poNewFDefn)

9.24.1 Detailed Description

The primary class for reading ISO 8211 files. This class contains all the information read from the DDR record, and is used to read records from the file.

9.24.2 Constructor & Destructor Documentation

9.24.2.1 DDFModule::DDFModule ()

The constructor.

9.24.2.2 DDFModule::~~DDFModule ()

The destructor.

9.24.3 Member Function Documentation

9.24.3.1 void DDFModule::AddField (DDFFieldDefn * *poNewFDefn*)

Add new field definition.

Field definitions may only be added to DDFModules being used for writing, not those being used for reading. Ownership of the [DDFFieldDefn](#) object is taken by the [DDFModule](#).

Parameters

<i>poNewFDefn</i>	definition to be added to the module.
-------------------	---------------------------------------

9.24.3.2 void DDFModule::Close ()

Close an ISO 8211 file.

9.24.3.3 void DDFModule::Dump (FILE * *fp*)

Write out module info to debugging file.

A variety of information about the module is written to the debugging file. This includes all the field and subfield definitions read from the header.

Parameters

<i>fp</i>	The standard io file handle to write to. ie. stderr.
-----------	--

9.24.3.4 DDFFieldDefn * DDFModule::FindFieldDefn (const char * *pszFieldName*)

Fetch the definition of the named field.

This function will scan the [DDFFieldDefn](#)'s on this module, to find one with the indicated field name.

Parameters

<i>pszFieldName</i>	The name of the field to search for. The comparison is case insensitive.
---------------------	--

Returns

A pointer to the request [DDFFieldDefn](#) object is returned, or NULL if none matching the name are found. The return object remains owned by the [DDFModule](#), and should not be deleted by application code.

9.24.3.5 [DDFFieldDefn](#) * [DDFModule::GetField](#) (int *i*)

Fetch a field definition by index.

Parameters

<i>i</i>	(from 0 to GetFieldCount () - 1.
----------	--

Returns

the returned field pointer or NULL if the index is out of range.

9.24.3.6 int [DDFModule::GetFieldCount](#) () [[inline](#)]

Fetch the number of defined fields.

9.24.3.7 int [DDFModule::Open](#) (const char * *pszFilename*, int *bFailQuietly* = FALSE)

Open a ISO 8211 (DDF) file for reading.

If the open succeeds the data descriptive record (DDR) will have been read, and all the field and subfield definitions will be available.

Parameters

<i>pszFilename</i>	The name of the file to open.
<i>bFailQuietly</i>	If FALSE a CPL Error is issued for non-8211 files, otherwise quietly return NULL.

Returns

FALSE if the open fails or TRUE if it succeeds. Errors messages are issued internally with [CPL_Error\(\)](#).

9.24.3.8 [DDFRecord](#) * [DDFModule::ReadRecord](#) (void)

Read one record from the file.

Returns

A pointer to a [DDFRecord](#) object is returned, or NULL if a read error, or end of file occurs. The returned record is owned by the module, and should not be deleted by the application. The record is only valid until the next [ReadRecord\(\)](#) at which point it is overwritten.

9.24.3.9 void [DDFModule::Rewind](#) (long *nOffset* = -1)

Return to first record.

The next call to [ReadRecord\(\)](#) will read the first data record in the file.

Parameters

<i>nOffset</i>	the offset in the file to return to. By default this is -1, a special value indicating that reading should return to the first data record. Otherwise it is an absolute byte offset in the file.
----------------	--

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

- iso8211.h
- ddfmodule.cpp

9.25 DDFRecord Class Reference

```
#include <iso8211.h>
```

Public Member Functions

- [DDFRecord](#) * [Clone](#) ()
- [DDFRecord](#) * [CloneOn](#) ([DDFModule](#) *)
- void [Dump](#) (FILE *)
- int [GetFieldCount](#) ()
- [DDFField](#) * [FindField](#) (const char *, int=0)
- [DDFField](#) * [GetField](#) (int)
- int [GetIntSubfield](#) (const char *, int, const char *, int, int *=NULL)
- double [GetFloatSubfield](#) (const char *, int, const char *, int, int *=NULL)
- const char * [GetStringSubfield](#) (const char *, int, const char *, int, int *=NULL)
- int [SetIntSubfield](#) (const char *pszField, int iFieldIndex, const char *pszSubfield, int iSubfieldIndex, int nValue)
- int [SetStringSubfield](#) (const char *pszField, int iFieldIndex, const char *pszSubfield, int iSubfieldIndex, const char *pszValue, int nValueLength=-1)
- int [SetFloatSubfield](#) (const char *pszField, int iFieldIndex, const char *pszSubfield, int iSubfieldIndex, double dfNewValue)
- int [GetDataSize](#) ()
- const char * [GetData](#) ()
- [DDFModule](#) * [GetModule](#) ()
- int [ResizeField](#) ([DDFField](#) *poField, int nNewDataSize)
- int [DeleteField](#) ([DDFField](#) *poField)
- [DDFField](#) * [AddField](#) ([DDFFieldDefn](#) *)
- int [CreateDefaultFieldInstance](#) ([DDFField](#) *poField, int iIndexWithinField)
- int [SetFieldRaw](#) ([DDFField](#) *poField, int iIndexWithinField, const char *pachRawData, int nRawDataSize)
- int [Write](#) ()

9.25.1 Detailed Description

Contains instance data from one data record (DR). The data is contained as a list of [DDFField](#) instances partitioning the raw data into fields.

9.25.2 Member Function Documentation

9.25.2.1 [DDFField](#) * [DDFRecord::AddField](#) ([DDFFieldDefn](#) * *poDefn*)

Add a new field to record.

Add a new zero sized field to the record. The new field is always added at the end of the record.

NOTE: This method doesn't currently update the header information for the record to include the field information for this field, so the resulting record image isn't suitable for writing to disk. However, everything else about the record state should be updated properly to reflect the new field.

Parameters

<i>poDefn</i>	the definition of the field to be added.
---------------	--

Returns

the field object on success, or NULL on failure.

9.25.2.2 DDFRecord * DDFRecord::Clone ()

Make a copy of a record.

This method is used to make a copy of a record that will become (mostly) the property of application. However, it is automatically destroyed if the [DDFModule](#) it was created relative to is destroyed, as it's field and subfield definitions relate to that [DDFModule](#). However, it does persist even when the record returned by [DDFModule::ReadRecord\(\)](#) is invalidated, such as when reading a new record. This allows an application to cache whole DDFRecords.

Returns

A new copy of the [DDFRecord](#). This can be delete'd by the application when no longer needed, otherwise it will be cleaned up when the [DDFModule](#) it relates to is destroyed or closed.

9.25.2.3 DDFRecord * DDFRecord::CloneOn (DDFModule * poTargetModule)

Recreate a record referencing another module.

Works similarly to the [DDFRecord::Clone\(\)](#) method, but creates the new record with reference to a different [DDFModule](#). All [DDFFieldDefn](#) references are transcribed onto the new module based on field names. If any fields don't have a similarly named field on the target module the operation will fail. No validation of field types and properties is done, but this operation is intended only to be used between modules with matching definitions of all affected fields.

The new record will be managed as a clone by the target module in a manner similar to regular clones.

Parameters

<i>poTargetModule</i>	the module on which the record copy should be created.
-----------------------	--

Returns

NULL on failure or a pointer to the cloned record.

9.25.2.4 int DDFRecord::CreateDefaultFieldInstance (DDFField * poField, int iIndexWithinField)

Initialize default instance.

This method is normally only used internally by the [AddField\(\)](#) method to initialize the new field instance with default subfield values. It installs default data for one instance of the field in the record using the [DDFFieldDefn::GetDefaultValue\(\)](#) method and [DDFRecord::SetFieldRaw\(\)](#).

Parameters

<i>poField</i>	the field within the record to be assign a default instance.
<i>iIndexWithinField</i>	the instance to set (may not have been tested with values other than 0).

Returns

TRUE on success or FALSE on failure.

9.25.2.5 int DDFRecord::DeleteField (DDFField * *poTarget*)

Delete a field instance from a record.

Remove a field from this record, cleaning up the data portion and repacking the fields list. We don't try to reallocate the data area of the record to be smaller.

NOTE: This method doesn't actually remove the header information for this field from the record tag list yet. This should be added if the resulting record is even to be written back to disk!

Parameters

<i>poTarget</i>	the field instance on this record to delete.
-----------------	--

Returns

TRUE on success, or FALSE on failure. Failure can occur if *poTarget* isn't really a field on this record.

9.25.2.6 void DDFRecord::Dump (FILE * *fp*)

Write out record contents to debugging file.

A variety of information about this record, and all it's fields and subfields is written to the given debugging file handle. Note that field definition information (ala [DDFFieldDefn](#)) isn't written.

Parameters

<i>fp</i>	The standard io file handle to write to. ie. stderr
-----------	---

9.25.2.7 DDFField * DDFRecord::FindField (const char * *pszName*, int *iFieldIndex* = 0)

Find the named field within this record.

Parameters

<i>pszName</i>	The name of the field to fetch. The comparison is case insensitive.
<i>iFieldIndex</i>	The instance of this field to fetch. Use zero (the default) for the first instance.

Returns

Pointer to the requested [DDFField](#). This pointer is to an internal object, and should not be freed. It remains valid until the next record read.

9.25.2.8 const char* DDFRecord::GetData () [inline]

Fetch the raw data for this record. The returned pointer is effectively to the data for the first field of the record, and is of size [GetDataSize\(\)](#).

9.25.2.9 int DDFRecord::GetDataSize () [inline]

Fetch size of records raw data ([GetData\(\)](#)) in bytes.

9.25.2.10 DDFField * DDFRecord::GetField (int *i*)

Fetch field object based on index.

Parameters

<i>i</i>	The index of the field to fetch. Between 0 and GetFieldCount() -1.
----------	--

Returns

A [DDFField](#) pointer, or NULL if the index is out of range.

9.25.2.11 int DDFRecord::GetFieldCount () [inline]

Get the number of DDFFields on this record.

9.25.2.12 double DDFRecord::GetFloatSubfield (const char * *pszField*, int *iFieldIndex*, const char * *pszSubfield*, int *iSubfieldIndex*, int * *pnSuccess* = NULL)

Fetch value of a subfield as a float (double). This is a convenience function for fetching a subfield of a field within this record.

Parameters

<i>pszField</i>	The name of the field containing the subfield.
<i>iFieldIndex</i>	The instance of this field within the record. Use zero for the first instance of this field.
<i>pszSubfield</i>	The name of the subfield within the selected field.
<i>iSubfieldIndex</i>	The instance of this subfield within the record. Use zero for the first instance.
<i>pnSuccess</i>	Pointer to an int which will be set to TRUE if the fetch succeeds, or FALSE if it fails. Use NULL if you don't want to check success.

Returns

The value of the subfield, or zero if it failed for some reason.

9.25.2.13 int DDFRecord::GetIntSubfield (const char * *pszField*, int *iFieldIndex*, const char * *pszSubfield*, int *iSubfieldIndex*, int * *pnSuccess* = NULL)

Fetch value of a subfield as an integer. This is a convenience function for fetching a subfield of a field within this record.

Parameters

<i>pszField</i>	The name of the field containing the subfield.
<i>iFieldIndex</i>	The instance of this field within the record. Use zero for the first instance of this field.
<i>pszSubfield</i>	The name of the subfield within the selected field.
<i>iSubfieldIndex</i>	The instance of this subfield within the record. Use zero for the first instance.
<i>pnSuccess</i>	Pointer to an int which will be set to TRUE if the fetch succeeds, or FALSE if it fails. Use NULL if you don't want to check success.

Returns

The value of the subfield, or zero if it failed for some reason.

9.25.2.14 DDFModule* DDFRecord::GetModule () [inline]

Fetch the [DDFModule](#) with which this record is associated.

9.25.2.15 const char * DDFRecord::GetStringSubfield (const char * *pszField*, int *iFieldIndex*, const char * *pszSubfield*, int *iSubfieldIndex*, int * *pnSuccess* = NULL)

Fetch value of a subfield as a string. This is a convenience function for fetching a subfield of a field within this record.

Parameters

<i>pszField</i>	The name of the field containing the subfield.
<i>iFieldIndex</i>	The instance of this field within the record. Use zero for the first instance of this field.
<i>pszSubfield</i>	The name of the subfield within the selected field.
<i>iSubfieldIndex</i>	The instance of this subfield within the record. Use zero for the first instance.
<i>pnSuccess</i>	Pointer to an int which will be set to TRUE if the fetch succeeds, or FALSE if it fails. Use NULL if you don't want to check success.

Returns

The value of the subfield, or NULL if it failed for some reason. The returned pointer is to internal data and should not be modified or freed by the application.

9.25.2.16 int DDFRecord::ResizeField (DDFField * *poField*, int *nNewDataSize*)

Alter field data size within record.

This method will rearrange a [DDFRecord](#) altering the amount of space reserved for one of the existing fields. All following fields will be shifted accordingly. This includes updating the [DDFField](#) infos, and actually moving stuff within the data array after reallocating to the desired size.

Parameters

<i>poField</i>	the field to alter.
<i>nNewDataSize</i>	the number of data bytes to be reserved for the field.

Returns

TRUE on success or FALSE on failure.

9.25.2.17 int DDFRecord::SetFieldRaw (DDFField * *poField*, int *iIndexWithinField*, const char * *pachRawData*, int *nRawDataSize*)

Set the raw contents of a field instance.

Parameters

<i>poField</i>	the field to set data within.
<i>iIndexWithinField</i>	The instance of this field to replace. Must be a value between 0 and GetRepeatCount(). If GetRepeatCount() is used, a new instance of the field is appended.
<i>pachRawData</i>	the raw data to replace this field instance with.
<i>nRawDataSize</i>	the number of bytes pointed to by pachRawData.

Returns

TRUE on success or FALSE on failure.

9.25.2.18 `int DDFRecord::SetFloatSubfield ( const char * pszField, int iFieldIndex, const char * pszSubfield, int iSubfieldIndex, double dfNewValue )`

Set a float subfield in record.

The value of a given subfield is replaced with a new float value formatted appropriately.

Parameters

<i>pszField</i>	the field name to operate on.
<i>iFieldIndex</i>	the field index to operate on (zero based).
<i>pszSubfield</i>	the subfield name to operate on.
<i>iSubfieldIndex</i>	the subfield index to operate on (zero based).
<i>dfNewValue</i>	the new value to place in the subfield.

Returns

TRUE if successful, and FALSE if not.

9.25.2.19 `int DDFRecord::SetIntSubfield ( const char * pszField, int iFieldIndex, const char * pszSubfield, int iSubfieldIndex, int nNewValue )`

Set an integer subfield in record.

The value of a given subfield is replaced with a new integer value formatted appropriately.

Parameters

<i>pszField</i>	the field name to operate on.
<i>iFieldIndex</i>	the field index to operate on (zero based).
<i>pszSubfield</i>	the subfield name to operate on.
<i>iSubfieldIndex</i>	the subfield index to operate on (zero based).
<i>nNewValue</i>	the new value to place in the subfield.

Returns

TRUE if successful, and FALSE if not.

9.25.2.20 `int DDFRecord::SetStringSubfield ( const char * pszField, int iFieldIndex, const char * pszSubfield, int iSubfieldIndex, const char * pszValue, int nValueLength = -1 )`

Set a string subfield in record.

The value of a given subfield is replaced with a new string value formatted appropriately.

Parameters

<i>pszField</i>	the field name to operate on.
<i>iFieldIndex</i>	the field index to operate on (zero based).
<i>pszSubfield</i>	the subfield name to operate on.
<i>iSubfieldIndex</i>	the subfield index to operate on (zero based).
<i>pszValue</i>	the new string to place in the subfield. This may be arbitrary binary bytes if <i>nValueLength</i> is specified.
<i>nValueLength</i>	the number of valid bytes in <i>pszValue</i> , may be -1 to internally fetch with <code>strlen()</code> .

Returns

TRUE if successful, and FALSE if not.

9.25.2.21 int DDFRecord::Write ()

Write record out to module.

This method writes the current record to the module to which it is attached. Normally this would be at the end of the file, and only used for modules newly created with `DDFModule::Create()`. Rewriting existing records is not supported at this time. Calling `Write()` multiple times on a `DDFRecord` will result it multiple copies being written at the end of the module.

Returns

TRUE on success or FALSE on failure.

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

- iso8211.h
- ddfrecord.cpp

9.26 DDSubfieldDefn Class Reference

```
#include <iso8211.h>
```

Public Types

- enum `DDFBinaryFormat`

Public Member Functions

- const char * `GetName` ()
- const char * `GetFormat` ()
- DDFDataType `GetType` ()
- double `ExtractFloatData` (const char *pachData, int nMaxBytes, int *pnConsumedBytes)
- int `ExtractIntData` (const char *pachData, int nMaxBytes, int *pnConsumedBytes)
- const char * `ExtractStringData` (const char *pachData, int nMaxBytes, int *pnConsumedBytes)
- int `GetDataLength` (const char *, int, int *)
- void `DumpData` (const char *pachData, int nMaxBytes, FILE *fp)
- int `FormatStringValue` (char *pachData, int nBytesAvailable, int *pnBytesUsed, const char *pszValue, int nValueLength=-1)
- int `FormatIntValue` (char *pachData, int nBytesAvailable, int *pnBytesUsed, int nNewValue)
- int `FormatFloatValue` (char *pachData, int nBytesAvailable, int *pnBytesUsed, double dfNewValue)
- int `GetWidth` ()
- int `GetDefaultValue` (char *pachData, int nBytesAvailable, int *pnBytesUsed)
- void `Dump` (FILE *fp)

9.26.1 Detailed Description

Information from the DDR record describing one subfield of a `DDFFieldDefn`. All subfields of a field will occur in each occurrence of that field (as a `DDFField`) in a `DDFRecord`. Subfield's actually contain formatted data (as instances within a record).

9.26.2 Member Enumeration Documentation

9.26.2.1 enum DDFSubfieldDefn::DDFBinaryFormat

Binary format: this is the digit immediately following the B or b for binary formats.

9.26.3 Member Function Documentation

9.26.3.1 void DDFSubfieldDefn::Dump (FILE * *fp*)

Write out subfield definition info to debugging file.

A variety of information about this field definition is written to the give debugging file handle.

Parameters

<i>fp</i>	The standard io file handle to write to. ie. stderr
-----------	---

9.26.3.2 void DDFSubfieldDefn::DumpData (const char * *pachData*, int *nMaxBytes*, FILE * *fp*)

Dump subfield value to debugging file.

Parameters

<i>pachData</i>	Pointer to data for this subfield.
<i>nMaxBytes</i>	Maximum number of bytes available in pachData.
<i>fp</i>	File to write report to.

9.26.3.3 double DDFSubfieldDefn::ExtractFloatData (const char * *pachSourceData*, int *nMaxBytes*, int * *pnConsumedBytes*)

Extract a subfield value as a float. Given a pointer to the data for this subfield (from within a [DDFRecord](#)) this method will return the floating point data for this subfield. The number of bytes consumed as part of this field can also be fetched. This method may be called for any type of subfield, and will return zero if the subfield is not numeric.

Parameters

<i>pachSourceData</i>	The pointer to the raw data for this field. This may have come from DDFRecord::GetData() , taking into account skip factors over previous subfields data.
<i>nMaxBytes</i>	The maximum number of bytes that are accessible after pachSourceData.
<i>pnConsumed-Bytes</i>	Pointer to an integer into which the number of bytes consumed by this field should be written. May be NULL to ignore. This is used as a skip factor to increment pachSourceData to point to the next subfields data.

Returns

The subfield's numeric value (or zero if it isn't numeric).

See Also

[ExtractIntData\(\)](#), [ExtractStringData\(\)](#)

9.26.3.4 int DDFSubfieldDefn::ExtractIntData (const char * *pachSourceData*, int *nMaxBytes*, int * *pnConsumedBytes*)

Extract a subfield value as an integer. Given a pointer to the data for this subfield (from within a [DDFRecord](#)) this method will return the int data for this subfield. The number of bytes consumed as part of this field can also be

fetches. This method may be called for any type of subfield, and will return zero if the subfield is not numeric.

Parameters

<i>pachSourceData</i>	The pointer to the raw data for this field. This may have come from DDFRecord::GetData() , taking into account skip factors over previous subfields data.
<i>nMaxBytes</i>	The maximum number of bytes that are accessible after pachSourceData.
<i>pnConsumed-Bytes</i>	Pointer to an integer into which the number of bytes consumed by this field should be written. May be NULL to ignore. This is used as a skip factor to increment pachSourceData to point to the next subfields data.

Returns

The subfield's numeric value (or zero if it isn't numeric).

See Also

[ExtractFloatData\(\)](#), [ExtractStringData\(\)](#)

9.26.3.5 `const char * DDFSubfieldDefn::ExtractStringData ( const char * pachSourceData, int nMaxBytes, int * pnConsumedBytes )`

Extract a zero terminated string containing the data for this subfield. Given a pointer to the data for this subfield (from within a [DDFRecord](#)) this method will return the data for this subfield. The number of bytes consumed as part of this field can also be fetched. This number may be one longer than the string length if there is a terminator character used.

This function will return the raw binary data of a subfield for types other than DDFString, including data past zero chars. This is the standard way of extracting DDFBinaryString subfields for instance.

Parameters

<i>pachSourceData</i>	The pointer to the raw data for this field. This may have come from DDFRecord::GetData() , taking into account skip factors over previous subfields data.
<i>nMaxBytes</i>	The maximum number of bytes that are accessible after pachSourceData.
<i>pnConsumed-Bytes</i>	Pointer to an integer into which the number of bytes consumed by this field should be written. May be NULL to ignore. This is used as a skip factor to increment pachSourceData to point to the next subfields data.

Returns

A pointer to a buffer containing the data for this field. The returned pointer is to an internal buffer which is invalidated on the next [ExtractStringData\(\)](#) call on this DDFSubfieldDefn(). It should not be freed by the application.

See Also

[ExtractIntData\(\)](#), [ExtractFloatData\(\)](#)

9.26.3.6 `int DDFSubfieldDefn::FormatFloatValue ( char * pachData, int nBytesAvailable, int * pnBytesUsed, double dfNewValue )`

Format float subfield value.

Returns a buffer with the passed in float value reformatted in a way suitable for storage in a [DDFField](#) for this subfield.

9.26.3.7 `int DDSubfieldDefn::FormatIntValue ( char * pachData, int nBytesAvailable, int * pnBytesUsed, int nNewValue )`

Format int subfield value.

Returns a buffer with the passed in int value reformatted in a way suitable for storage in a [DDField](#) for this subfield.

9.26.3.8 `int DDSubfieldDefn::FormatStringValue ( char * pachData, int nBytesAvailable, int * pnBytesUsed, const char * pszValue, int nValueLength = -1 )`

Format string subfield value.

Returns a buffer with the passed in string value reformatted in a way suitable for storage in a [DDField](#) for this subfield.

9.26.3.9 `int DDSubfieldDefn::GetDataLength ( const char * pachSourceData, int nMaxBytes, int * pnConsumedBytes )`

Scan for the end of variable length data. Given a pointer to the data for this subfield (from within a [DDRecord](#)) this method will return the number of bytes which are data for this subfield. The number of bytes consumed as part of this field can also be fetched. This number may be one longer than the length if there is a terminator character used.

This method is mainly for internal use, or for applications which want the raw binary data to interpret themselves. Otherwise use one of [ExtractStringData\(\)](#), [ExtractIntData\(\)](#) or [ExtractFloatData\(\)](#).

Parameters

<i>pachSourceData</i>	The pointer to the raw data for this field. This may have come from DDRecord::GetData() , taking into account skip factors over previous subfields data.
<i>nMaxBytes</i>	The maximum number of bytes that are accessible after <i>pachSourceData</i> .
<i>pnConsumed-Bytes</i>	Pointer to an integer into which the number of bytes consumed by this field should be written. May be NULL to ignore.

Returns

The number of bytes at *pachSourceData* which are actual data for this record (not including unit, or field terminator).

9.26.3.10 `int DDSubfieldDefn::GetDefaultValue ( char * pachData, int nBytesAvailable, int * pnBytesUsed )`

Get default data.

Returns the default subfield data contents for this subfield definition. For variable length numbers this will normally be "0<unit-terminator>". For variable length strings it will be "<unit-terminator>". For fixed length numbers it is zero filled. For fixed length strings it is space filled. For binary numbers it is binary zero filled.

Parameters

<i>pachData</i>	the buffer into which the returned default will be placed. May be NULL if just querying default size.
<i>nBytesAvailable</i>	the size of <i>pachData</i> in bytes.
<i>pnBytesUsed</i>	will receive the size of the subfield default data in bytes.

Returns

TRUE on success or FALSE on failure or if the passed buffer is too small to hold the default.

9.26.3.11 `const char* DDFSubfieldDefn::GetFormat ( ) [inline]`

Get pointer to subfield format string

9.26.3.12 `const char* DDFSubfieldDefn::GetName ( ) [inline]`

Get pointer to subfield name.

9.26.3.13 `DDFDataType DDFSubfieldDefn::GetType ( ) [inline]`

Get the general type of the subfield. This can be used to determine which of [ExtractFloatData\(\)](#), [ExtractIntData\(\)](#) or [ExtractStringData\(\)](#) should be used.

Returns

The subfield type. One of DDFInt, DDFFloat, DDFString or DDFBinaryString.

9.26.3.14 `int DDFSubfieldDefn::GetWidth ( ) [inline]`

Get the subfield width (zero for variable).

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

- iso8211.h
- ddsubfielddefn.cpp

9.27 DefaultCSVFileNameTLS Struct Reference

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

- cpl_csv.cpp

9.28 errHandler Struct Reference

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

- cpl_error.cpp

9.29 file_in_zip_read_info_s Struct Reference

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

- cpl_minizip_unzip.cpp

9.30 FindFileTLS Struct Reference

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

- cpl_findfile.cpp

9.31 GDALScaledProgressInfo Struct Reference

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

- `cpl_progress.cpp`

9.32 GZipSnapshot Struct Reference

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

- `cpl_vsil_gzip.cpp`

9.33 linkedlist_data_s Struct Reference

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

- `cpl_minizip_zip.cpp`

9.34 linkedlist_datablock_internal_s Struct Reference

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

- `cpl_minizip_zip.cpp`

9.35 ParseContext Struct Reference

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

- `cpl_minixml.cpp`

9.36 SDTS_CATD Class Reference

```
#include <sdts_al.h>
```

Public Member Functions

- `const char *` [GetEntryTypeDesc](#) (int)
- `const char *` [GetEntryFilePath](#) (int)
- `SDTSLayerType` [GetEntryType](#) (int)

9.36.1 Detailed Description

Class for accessing the CATD (Catalog Directory) file containing a list of all other files (modules) in the transfer.

9.36.2 Member Function Documentation

9.36.2.1 `const char * SDTS_CATD::GetEntryFilePath ( int iEntry )`

Fetch the full filename of the requested module.

Parameters

<i>iEntry</i>	The module index within the CATD catalog. A number from zero to <code>GetEntryCount()-1</code> .
---------------	--

Returns

A pointer to an internal string containing the filename. This string should not be altered, or freed by the application.

9.36.2.2 `SDTSLayerType SDTS_CATD::GetEntryType ( int iEntry )`

Fetch the enumerated type of a module in the catalog.

Parameters

<i>iEntry</i>	The module index within the CATD catalog. A number from zero to <code>GetEntryCount()-1</code> .
---------------	--

Returns

A value from the `SDTSLayerType` enumeration indicating the type of the module, and indicating the corresponding type of reader.

- `SLTPoint`: Read with [SDTSPointReader](#), underlying type of `Point-Node`.
- `SLTLine`: Read with [SDTSLineReader](#), underlying type of `Line`.
- `SLTAttr`: Read with [SDTSAttrReader](#), underlying type of `Attribute Primary` or `Attribute Secondary`.
- `SLTPolygon`: Read with [SDTSPolygonReader](#), underlying type of `Polygon`.

9.36.2.3 `const char * SDTS_CATD::GetEntryTypeDesc ( int iEntry )`

Fetch the type description of a module in the catalog.

Parameters

<i>iEntry</i>	The module index within the CATD catalog. A number from zero to <code>GetEntryCount()-1</code> .
---------------	--

Returns

A pointer to an internal string with the type description for this module. This is from the CATD file (subfield `TYPE` of field `CATD`), and will be something like "Attribute Primary".

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

- `sdtc_al.h`
- `sdtscatd.cpp`

9.37 SDTS_CATDEntry Class Reference

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

- `sdts_catd.cpp`

9.38 SDTS_IREF Class Reference

```
#include <sdts_al.h>
```

9.38.1 Detailed Description

Class holding SDTS IREF (internal reference) information, internal coordinate system format, scaling and resolution. This object isn't normally needed by applications.

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

- `sdts_al.h`
- `sdtsiref.cpp`

9.39 SDTS_XREF Class Reference

```
#include <sdts_al.h>
```

Public Attributes

- `char * pszSystemName`
- `char * pszDatum`
- `int nZone`

9.39.1 Detailed Description

Class for reading the XREF (external reference) module containing the data projection definition.

9.39.2 Member Data Documentation

9.39.2.1 `int SDTS_XREF::nZone`

Zone number for UTM and SPCS projections, from the ZONE field.

9.39.2.2 `char* SDTS_XREF::pszDatum`

Horizontal datum name, from the HDAT field. One of NAS, NAX, WGA, WGB, WGC, WGE.

9.39.2.3 `char* SDTS_XREF::pszSystemName`

Projection system name, from the RSNM field. One of GEO, SPCS, UTM, UPS, OTHR, UNSP.

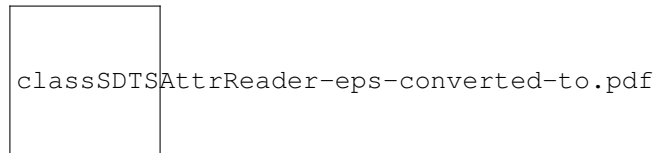
The documentation for this class was generated from the following files:

- [sdts_al.h](#)
- [sdtsxref.cpp](#)

9.40 SDTSAttrReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSAttrReader:



Public Member Functions

- [int IsSecondary \(\)](#)

9.40.1 Detailed Description

Class for reading [SDTSAttrRecord](#) features from a primary or secondary attribute module.

9.40.2 Member Function Documentation

9.40.2.1 [int SDTSAttrReader::IsSecondary \(\)](#) [[inline](#)]

Returns TRUE if this is a Attribute Secondary layer rather than an Attribute Primary layer.

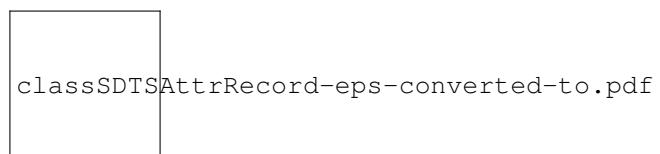
The documentation for this class was generated from the following files:

- [sdts_al.h](#)
- [sdtsattrreader.cpp](#)

9.41 SDTSAttrRecord Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSAttrRecord:



Public Member Functions

- [virtual void Dump \(FILE *\)](#)

Public Attributes

- [DDFRecord](#) * [poWholeRecord](#)
- [DDFField](#) * [poATTR](#)

9.41.1 Detailed Description

SDTS attribute record feature, as read from A* modules by [SDTSAttrReader](#).

Note that even though [SDTSAttrRecord](#) is derived from [SDTSFeature](#), there are never any attribute records associated with attribute records using the `aoATID[]` mechanism. [SDTSFeature::nAttributes](#) will always be zero.

9.41.2 Member Function Documentation

9.41.2.1 `void SDTSAttrRecord::Dump ( FILE * ) [virtual]`

Dump readable description of feature to indicated stream.

Implements [SDTSFeature](#).

9.41.3 Member Data Documentation

9.41.3.1 `DDFField*` [SDTSAttrRecord::poATTR](#)

The ATTR [DDFField](#) with the user attribute. Each subfield is a attribute value.

9.41.3.2 `DDFRecord*` [SDTSAttrRecord::poWholeRecord](#)

The entire [DDFRecord](#) read from the file.

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

- `sdts_al.h`
- `sdtsattrreader.cpp`

9.42 SDTSDataset Class Reference

Inheritance diagram for SDTSDataset:



Friends

- class [SDTSRasterBand](#)

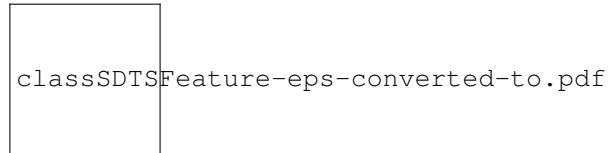
The documentation for this class was generated from the following file:

- [sdtsdataset.cpp](#)

9.43 SDTSFeature Class Reference

```
#include <sdt_s_al.h>
```

Inheritance diagram for SDTSFeature:



Public Member Functions

- virtual void [Dump](#) (FILE *)=0

Public Attributes

- [SDTSModId](#) [oModId](#)
- int [nAttributes](#)
- [SDTSModId](#) * [paoATID](#)

9.43.1 Detailed Description

Base class for various SDTS features classes, providing a generic module identifier, and list of attribute references.

9.43.2 Member Function Documentation

9.43.2.1 virtual void SDTSFeature::Dump (FILE *) [pure virtual]

Dump readable description of feature to indicated stream.

Implemented in [SDTSRawPolygon](#), [SDTSRawPoint](#), [SDTSAttrRecord](#), and [SDTSRawLine](#).

9.43.3 Member Data Documentation

9.43.3.1 int SDTSFeature::nAttributes

Number of attribute links (aoATID[]) on this feature.

9.43.3.2 SDTSModId SDTSFeature::oModId

Unique identifier for this record/feature within transfer.

9.43.3.3 SDTSModId* SDTSFeature::paoATID

List of nAttributes attribute record identifiers related to this feature.

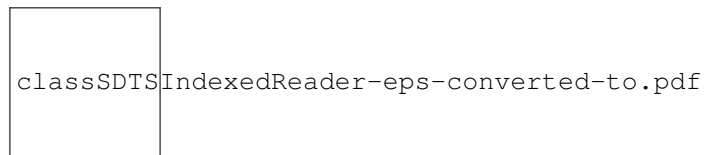
The documentation for this class was generated from the following files:

- [sdt_s_al.h](#)
- [sdtlib.cpp](#)

9.44 SDTSIndexedReader Class Reference

```
#include <sdt_s_al.h>
```

Inheritance diagram for SDTSIndexedReader:



Public Member Functions

- [SDTSFeature](#) * [GetNextFeature](#) ()
- virtual void [Rewind](#) ()
- void [FillIndex](#) ()
- void [ClearIndex](#) ()
- int [IsIndexed](#) ()
- [SDTSFeature](#) * [GetIndexedFeatureRef](#) (int)
- char ** [ScanModuleReferences](#) (const char *="ATID")

9.44.1 Detailed Description

Base class for all the [SDTSFeature](#) type readers. Provides feature caching semantics and fetching based on a record number.

9.44.2 Member Function Documentation

9.44.2.1 void SDTSIndexedReader::ClearIndex ()

Free all features in the index (if filled).

After this the reader is considered to not be indexed, and [IsIndexed\(\)](#) will return FALSE until the index is forcibly filled again.

9.44.2.2 void SDTSIndexedReader::FillIndex ()

Read all features into a memory indexed cached.

The [ClearIndex\(\)](#) method can be used to free all indexed features. [FillIndex\(\)](#) does nothing, if an index has already been built.

9.44.2.3 SDTSFeature * SDTSIndexedReader::GetIndexedFeatureRef (int *iRecordId*)

Fetch a feature based on it's record number.

This method will forcibly fill the feature cache, reading all the features in the file into memory, if they haven't already been loaded. The [ClearIndex\(\)](#) method can be used to flush this cache when no longer needed.

Parameters

<i>iRecordId</i>	the record to fetch, normally based on the nRecord field of an SDTSModId .
------------------	--

Returns

a pointer to an internal feature (not to be deleted) or NULL if there is no matching feature.

9.44.2.4 SDTSFeature * SDTSIndexedReader::GetNextFeature ()

Fetch the next available feature from this reader.

The returned [SDTSFeature](#) * is to an internal indexed object if the [IsIndexed\(\)](#) method returns TRUE, otherwise the returned feature becomes the responsibility of the caller to destroy with delete.

Note that the [Rewind\(\)](#) method can be used to start over at the beginning of the modules feature list.

Returns

next feature, or NULL if no more are left. Please review above ownership/delete semantics.

9.44.2.5 int SDTSIndexedReader::IsIndexed ()

Returns TRUE if the module is indexed, otherwise it returns FALSE.

If the module is indexed all the feature have already been read into memory, and searches based on the record number can be performed efficiently.

9.44.2.6 void SDTSIndexedReader::Rewind () [virtual]

Rewind so that the next feature returned by [GetNextFeature\(\)](#) will be the first in the module.

9.44.2.7 char ** SDTSIndexedReader::ScanModuleReferences (const char * pszFName = "ATID")

Scan an entire SDTS module for record references with the given field name.

The fields are required to have a MODN subfield from which the module is extracted.

This method is normally used to find all the attribute modules referred to by a point, line or polygon module to build a unified schema.

This method will have the side effect of rewinding unindexed readers because the scanning operation requires reading all records in the module from disk.

Parameters

<i>pszFName</i>	the field name to search for. By default "ATID" is used.
-----------------	--

Returns

a NULL terminated list of module names. Free with [CSLDestroy\(\)](#).

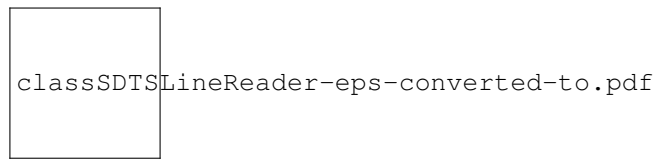
The documentation for this class was generated from the following files:

- [sdts_al.h](#)
- [sdtsindexedreader.cpp](#)

9.45 SDTSLineReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSLineReader:



Public Member Functions

- void [AttachToPolygons](#) ([SDTSTransfer](#) *, int iPolyLayer)

9.45.1 Detailed Description

Reader for SDTS line modules.

Returns [SDTSRawLine](#) features. Normally readers are instantiated with the [SDTSTransfer::GetIndexedReader\(\)](#) method.

9.45.2 Member Function Documentation

9.45.2.1 void [SDTSLineReader::AttachToPolygons](#) ([SDTSTransfer](#) * *poTransfer*, int *iTargetPolyLayer*)

Attach lines in this module to their polygons as the first step in polygon formation.

See also the [SDTSRawPolygon::AssembleRings\(\)](#) method.

Parameters

<i>poTransfer</i>	the SDTSTransfer of this SDTSLineReader , and from which the related SDTSPolygonReader will be instantiated.
<i>iTargetPolyLayer</i>	the polygon reader instance number, used to avoid processing lines for other layers.

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

- [sdts_al.h](#)
- [sdtslinereader.cpp](#)

9.46 SDTSModId Class Reference

```
#include <sdts_al.h>
```

Public Attributes

- char [szModule](#) [8]
- long [nRecord](#)
- char [szOB RP](#) [8]
- char [szName](#) [20]

9.46.1 Detailed Description

Object representing a unique module/record identifier within an SDTS transfer.

9.46.2 Member Data Documentation

9.46.2.1 long SDTSModId::nRecord

The record within the module referred to. This is -1 for unused SDTSModIds.

9.46.2.2 char SDTSModId::szModule[8]

The module name, such as PC01, containing the indicated record.

9.46.2.3 char SDTSModId::szName[20]

String "szModule:nRecord"

9.46.2.4 char SDTSModId::szOB RP[8]

The "role" of this record within the module. This is normally empty for references, but set in the oModId member of a feature.

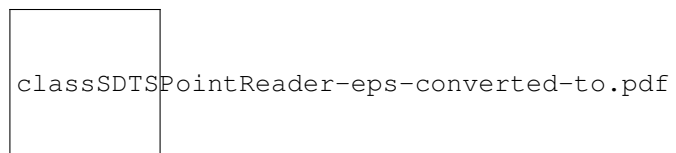
The documentation for this class was generated from the following files:

- sdts_al.h
- sdtslib.cpp

9.47 SDTSPointReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSPointReader:



Additional Inherited Members

9.47.1 Detailed Description

Class for reading [SDTSRawPoint](#) features from a point module (type NA, NO or NP).

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

- sdts_al.h
- sdtspointreader.cpp

9.48 SDTSPolygonReader Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSPolygonReader:



Public Member Functions

- void [AssembleRings](#) ([SDTSTransfer](#) *, int *iPolyLayer*)

9.48.1 Detailed Description

Class for reading [SDTSRawPolygon](#) features from a polygon (PC*) module.

9.48.2 Member Function Documentation

9.48.2.1 void SDTSPolygonReader::AssembleRings ([SDTSTransfer](#) * *poTransfer*, int *iPolyLayer*)

Assemble geometry for a polygon transfer.

This method takes care of attaching lines from all the line layers in this transfer to this polygon layer, assembling the lines into rings on the polygons, and then cleaning up unnecessary intermediate results.

Currently this method will leave the line layers rewound to the beginning but indexed, and the polygon layer rewound but indexed. In the future it may restore reading positions, and possibly flush line indexes if they were not previously indexed.

This method does nothing if the rings have already been assembled on this layer using this method.

See [SDTSRawPolygon::AssembleRings\(\)](#) for more information on how the lines are assembled into rings.

Parameters

<i>poTransfer</i>	the SDTSTransfer that this reader is a part of. Used to get a list of line layers that might be needed.
<i>iPolyLayer</i>	the polygon reader instance number, used to avoid processing lines for other layers.

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

- `sdtc_al.h`
- `sdtspolygonreader.cpp`

9.49 SDTSRasterBand Class Reference

Inheritance diagram for SDTSRasterBand:



Friends

- class **SDTSDataset**

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

- [sdtsdataset.cpp](#)

9.50 SDTSRasterReader Class Reference

```
#include <sdts_al.h>
```

Public Member Functions

- int [GetRasterType](#) ()
- int [GetTransform](#) (double *)
- int [GetMinMax](#) (double *pdfMin, double *pdfMax, double dfNoData)
- int [GetXSize](#) ()
- int [GetYSize](#) ()
- int [GetBlockXSize](#) ()
- int [GetBlockYSize](#) ()
- int [GetBlock](#) (int nXOffset, int nYOffset, void *pData)

9.50.1 Detailed Description

Class for reading raster data from a raster layer.

This class is somewhat unique among the reader classes in that it isn't derived from SDTSIndexedFeature, and it doesn't return "features". Instead it is used to read raster blocks, in the natural block size of the dataset.

9.50.2 Member Function Documentation

9.50.2.1 int SDTSRasterReader::GetBlock (int *nXOffset*, int *nYOffset*, void * *pData*)

Read a block of raster data from the file.

Parameters

<i>nXOffset</i>	X block offset into the file. Normally zero for scanline organized raster files.
<i>nYOffset</i>	Y block offset into the file. Normally the scanline offset from top of raster for scanline organized raster files.
<i>pData</i>	pointer to GInt16 (signed short) buffer of data into which to read the raster.

Returns

TRUE on success and FALSE on error.

9.50.2.2 int SDTSRasterReader::GetBlockXSize () [inline]

Fetch the width of a source block (usually same as raster width).

9.50.2.3 int SDTSRasterReader::GetBlockYSize () [inline]

Fetch the height of a source block (usually one).

9.50.2.4 int SDTSRasterReader::GetMinMax (double * *pdfMin*, double * *pdfMax*, double *dfNoData*)

Fetch the minimum and maximum raster values that occur in the file.

Note this operation current results in a scan of the entire file.

Parameters

<i>pdfMin</i>	variable in which the minimum value encountered is returned.
<i>pdfMax</i>	variable in which the maximum value encountered is returned.
<i>dfNoData</i>	a value to ignore when computing min/max, defaults to -32766.

Returns

TRUE on success, or FALSE if an error occurs.

9.50.2.5 int SDTSRasterReader::GetRasterType ()

Fetch the pixel data type.

Returns one of SDTS_RT_INT16 (1) or SDTS_RT_FLOAT32 (6) indicating the type of buffer that should be passed to [GetBlock\(\)](#).

9.50.2.6 int SDTSRasterReader::GetTransform (double * *padfTransformOut*)

Fetch the transformation between pixel/line coordinates and georeferenced coordinates.

Parameters

<i>padfTransformOut</i>	pointer to an array of six doubles which will be filled with the georeferencing transform.
-------------------------	--

Returns

TRUE is returned, indicating success.

The *padfTransformOut* array consists of six values. The pixel/line coordinate (Xp,Yp) can be related to a georeferenced coordinate (Xg,Yg) or (Easting, Northing).

```
Xg = padfTransformOut[0] + Xp * padfTransform[1] + Yp * padfTransform[2]
Yg = padfTransformOut[3] + Xp * padfTransform[4] + Yp * padfTransform[5]
```

In other words, for a north up image the top left corner of the top left pixel is at georeferenced coordinate (*padfTransform*[0],*padfTransform*[3]) the pixel width is *padfTransform*[1], the pixel height is *padfTransform*[5] and *padfTransform*[2] and *padfTransform*[4] will be zero.

9.50.2.7 int SDTSRasterReader::GetXSize () [inline]

Fetch the raster width.

Returns

the width in pixels.

9.50.2.8 `int SDTSRasterReader::GetYSize ( ) [inline]`

Fetch the raster height.

Returns

the height in pixels.

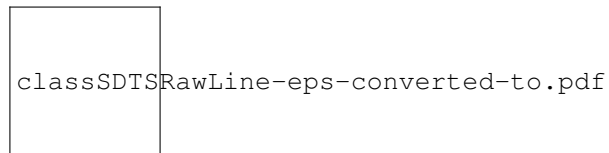
The documentation for this class was generated from the following files:

- `sdt_s_al.h`
- `sdt_rasterreader.cpp`

9.51 SDTSRawLine Class Reference

```
#include <sdt_s_al.h>
```

Inheritance diagram for SDTSRawLine:



Public Member Functions

- void [Dump](#) (FILE *)

Public Attributes

- int [nVertices](#)
- double * [padfX](#)
- double * [padfY](#)
- double * [padfZ](#)
- [SDTSModId](#) [oLeftPoly](#)
- [SDTSModId](#) [oRightPoly](#)
- [SDTSModId](#) [oStartNode](#)
- [SDTSModId](#) [oEndNode](#)

9.51.1 Detailed Description

SDTS line feature, as read from LE* modules by [SDTSLineReader](#).

9.51.2 Member Function Documentation

9.51.2.1 `void SDTSRawLine::Dump ( FILE * ) [virtual]`

Dump readable description of feature to indicated stream.

Implements [SDTSFeature](#).

9.51.3 Member Data Documentation

9.51.3.1 `int SDTSRawLine::nVertices`

Number of vertices in the `padfX`, `padfY` and `padfZ` arrays.

9.51.3.2 `SDTSMoId SDTSRawLine::oEndNode`

Identifier for the end node of this line. This is the SDTS ENID subfield.

9.51.3.3 `SDTSMoId SDTSRawLine::oLeftPoly`

Identifier of polygon to left of this line. This is the SDTS PIDL subfield.

9.51.3.4 `SDTSMoId SDTSRawLine::oRightPoly`

Identifier of polygon to right of this line. This is the SDTS PIDR subfield.

9.51.3.5 `SDTSMoId SDTSRawLine::oStartNode`

Identifier for the start node of this line. This is the SDTS SNID subfield.

9.51.3.6 `double* SDTSRawLine::padfX`

List of `nVertices` X coordinates.

9.51.3.7 `double* SDTSRawLine::padfY`

List of `nVertices` Y coordinates.

9.51.3.8 `double* SDTSRawLine::padfZ`

List of `nVertices` Z coordinates - currently always zero.

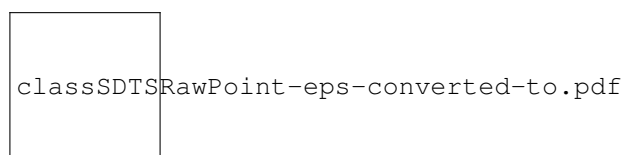
The documentation for this class was generated from the following files:

- `sdtc_al.h`
- `sdtc_linerreader.cpp`

9.52 SDTSRawPoint Class Reference

```
#include <sdtc_al.h>
```

Inheritance diagram for `SDTSRawPoint`:



Public Member Functions

- virtual void [Dump](#) (FILE *)

Public Attributes

- double [dfX](#)
- double [dfY](#)
- double [dfZ](#)
- [SDTSModId](#) [oAreald](#)

9.52.1 Detailed Description

Object containing a point feature (type NA, NO or NP).

9.52.2 Member Function Documentation

9.52.2.1 void SDTSRawPoint::Dump (FILE *) [virtual]

Dump readable description of feature to indicated stream.

Implements [SDTSFeature](#).

9.52.3 Member Data Documentation

9.52.3.1 double SDTSRawPoint::dfX

X coordinate of point.

9.52.3.2 double SDTSRawPoint::dfY

Y coordinate of point.

9.52.3.3 double SDTSRawPoint::dfZ

Z coordinate of point.

9.52.3.4 SDTSModId SDTSRawPoint::oAreald

Optional identifier of area marked by this point (ie. PC01:27).

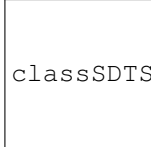
The documentation for this class was generated from the following files:

- [sdts_al.h](#)
- [sdtspointreader.cpp](#)

9.53 SDTSRawPolygon Class Reference

```
#include <sdts_al.h>
```

Inheritance diagram for SDTSRawPolygon:


 classSDTSRawPolygon-eps-converted-to.pdf

Public Member Functions

- int [AssembleRings](#) ()
- virtual void [Dump](#) (FILE *)

Public Attributes

- int [nRings](#)
- int [nVertices](#)
- int * [panRingStart](#)
- double * [padfX](#)
- double * [padfY](#)
- double * [padfZ](#)

9.53.1 Detailed Description

Class for holding information about a polygon feature.

When directly read from a polygon module, the polygon has no concept of it's geometry. Just it's ID, and references to attribute records. However, if the [SDTSLineReader::AttachToPolygons\(\)](#) method is called on the module containing the lines forming the polygon boundaries, then the nEdges/papoEdges information on the [SDTSRawPolygon](#) will be filled in.

Once this is complete the [AssembleRings\(\)](#) method can be used to fill in the nRings/nVertices/panRingStart/padfX/padfY/padfZ information defining the ring geometry.

Note that the rings may not appear in any particular order, nor with any meaningful direction (clockwise or counter-clockwise).

9.53.2 Member Function Documentation

9.53.2.1 int SDTSRawPolygon::AssembleRings ()

This method will assemble the edges associated with a polygon into rings, returning FALSE if problems are encountered during assembly.

Form border lines (arcs) into outer and inner rings.

See [SDTSPolygonReader::AssemblePolygons\(\)](#) for a simple one step process to assembling geometry for all polygons in a transfer.

This method will assemble the lines attached to a polygon into an outer ring, and zero or more inner rings. Before calling it is necessary that all the lines associated with this polygon have already been attached. Normally this is accomplished by calling [SDTSLineReader::AttachToPolygons\(\)](#) on all line layers that might contain edges related to this layer.

This method then forms the lines into rings. Rings are formed by:

1. Take a previously unconsumed line, and start a ring with it. Mark it as consumed, and keep track of it's start and end node ids as being the start and end node ids of the ring.
2. If the rings start id is the same as the end node id then this ring is completely formed, return to step 1.

3. Search all unconsumed lines for a line with the same start or end node id as the rings current node id. If none are found then the assembly has failed. Return to step 1 but report failure on completion.
4. Once found, add the line to the current ring, dropping the duplicated vertex and reverse order if necessary. Mark the line as consumed, and update the rings end node id accordingly.
5. go to step 2.

Once ring assembly from lines is complete, another pass is made to order the rings such that the exterior ring is first, the first ring has counter-clockwise vertex ordering and the inner rings have clockwise vertex ordering. This is accomplished based on the assumption that the outer ring has the largest area, and using the +/- sign of area to establish direction of rings.

Returns

TRUE if all rings assembled without problems or FALSE if a problem occurred. If a problem occurs rings are still formed from all lines, but some of the rings will not be closed, and rings will have no particular order or direction.

9.53.2.2 void SDTSRawPolygon::Dump (FILE *) [virtual]

Dump readable description of feature to indicated stream.

Implements [SDTSFeature](#).

9.53.3 Member Data Documentation

9.53.3.1 int SDTSRawPolygon::nRings

Number of rings in assembled polygon.

9.53.3.2 int SDTSRawPolygon::nVertices

Total number of vertices in all rings of assembled polygon.

9.53.3.3 double* SDTSRawPolygon::padfX

List of nVertices X coordinates for the polygon (split over multiple rings via panRingStart).

9.53.3.4 double* SDTSRawPolygon::padfY

List of nVertices Y coordinates for the polygon (split over multiple rings via panRingStart).

9.53.3.5 double* SDTSRawPolygon::padfZ

List of nVertices Z coordinates for the polygon (split over multiple rings via panRingStart). The values are almost always zero.

9.53.3.6 int* SDTSRawPolygon::panRingStart

Offsets into padfX/padfY/padfZ for the beginning of each ring in the polygon. This array is nRings long.

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

- `sdt_al.h`
- `sdtspolygonreader.cpp`

9.54 SDTSTransfer Class Reference

```
#include <sdts_al.h>
```

Public Member Functions

- int [Open](#) (const char *)
- int [FindLayer](#) (const char *)
- SDTSLayerType [GetLayerType](#) (int)
- int [GetLayerCATDEntry](#) (int)
- SDTSRasterReader * [GetLayerRasterReader](#) (int)
- SDTSIndexedReader * [GetLayerIndexedReader](#) (int)
- SDTS_CATD * [GetCATD](#) ()
- SDTS_XREF * [GetXREF](#) ()
- DDFField * [GetAttr](#) (SDTSModId *)
- int [GetBounds](#) (double *pdfMinX, double *pdfMinY, double *pdfMaxX, double *pdfMaxY)

9.54.1 Detailed Description

Master class representing an entire SDTS transfer.

This class is used to open the transfer, to get a list of available feature layers, and to instantiate readers for those layers.

9.54.2 Member Function Documentation

9.54.2.1 int SDTSTransfer::FindLayer (const char * *pszModule*)

Fetch the [SDTSTransfer](#) layer number corresponding to a module name.

Parameters

<i>pszModule</i>	the name of the module to search for, such as "PC01".
------------------	---

Returns

the layer number (between 0 and GetLayerCount()-1 corresponding to the module, or -1 if it doesn't correspond to a layer.

9.54.2.2 DDFField * SDTSTransfer::GetAttr (SDTSModId * *poModId*)

Fetch the attribute fields given a particular module/record id.

Parameters

<i>poModId</i>	an attribute record identifier, normally taken from the aoATID[] array of an SDTSIndexed-Feature.
----------------	---

Returns

a pointer to the [DDFField](#) containing the user attribute values as subfields.

9.54.2.3 `int SDTSTransfer::GetBounds ( double * pdfMinX, double * pdfMinY, double * pdfMaxX, double * pdfMaxY )`

Fetch approximate bounds for a transfer by scanning all point layers and raster layers.

For TVP datasets (where point layers are scanned) the results can, in theory miss some lines that go outside the bounds of the point layers. However, this isn't common since most TVP sets contain a bounding rectangle whose corners will define the most extreme extents.

Parameters

<i>pdfMinX</i>	western edge of dataset
<i>pdfMinY</i>	southern edge of dataset
<i>pdfMaxX</i>	eastern edge of dataset
<i>pdfMaxY</i>	northern edge of dataset

Returns

TRUE if success, or FALSE on a failure.

9.54.2.4 `SDTS_CATD* SDTSTransfer::GetCATD ( ) [inline]`

Fetch the catalog object for this transfer.

Returns

pointer to the internally managed [SDTS_CATD](#) for the transfer.

9.54.2.5 `int SDTSTransfer::GetLayerCATDEntry ( int iEntry )`

Fetch the CATD module index for a layer. This can be used to fetch details about the layer/module from the [SDTS_CATD](#) object, such as it's filename, and description.

Parameters

<i>iEntry</i>	the layer index from 0 to GetLayerCount()-1.
---------------	--

Returns

the module index suitable for use with the various [SDTS_CATD](#) methods.

9.54.2.6 `SDTSIndexedReader * SDTSTransfer::GetLayerIndexedReader ( int iEntry )`

Returns a pointer to a reader of the appropriate type to the requested layer.

Notes:

- The returned reader remains owned by the [SDTSTransfer](#), and will be destroyed when the [SDTSTransfer](#) is destroyed. It should not be destroyed by the application.
- If an indexed reader was already created for this layer using [GetLayerIndexedReader\(\)](#), it will be returned instead of creating a new reader. Amongst other things this means that the returned reader may not be positioned to read from the beginning of the module, and may already have it's index filled.
- The returned reader will be of a type appropriate to the layer. See [SDTSTransfer::GetLayerType\(\)](#) to see what reader classes correspond to what layer types, so it can be cast accordingly (if necessary).

Parameters

<i>iEntry</i>	the index of the layer to instantiate a reader for. A value between 0 and GetLayerCount()-1.
---------------	--

Returns

a pointer to an appropriate reader or NULL if the method fails.

9.54.2.7 SDTSRasterReader * SDTSTransfer::GetLayerRasterReader (int *iEntry*)

Instantiate an [SDTSRasterReader](#) for the indicated layer.

Parameters

<i>iEntry</i>	the index of the layer to instantiate a reader for. A value between 0 and GetLayerCount()-1.
---------------	--

Returns

a pointer to a new [SDTSRasterReader](#) object, or NULL if the method fails.

NOTE: The reader returned from [GetLayerRasterReader\(\)](#) becomes the responsibility of the caller to delete, and isn't automatically deleted when the [SDTSTransfer](#) is destroyed. This method is different from the [GetLayerIndexedReader\(\)](#) method in this regard.

9.54.2.8 SDTSLayerType SDTSTransfer::GetLayerType (int *iEntry*)

Fetch type of requested feature layer.

Parameters

<i>iEntry</i>	the index of the layer to fetch information on. A value from zero to GetLayerCount()-1.
---------------	---

Returns

the layer type.

- SLTPoint: A point layer. An [SDTSPointReader](#) is returned by [SDTSTransfer::GetLayerIndexedReader\(\)](#).
- SLTLine: A line layer. An [SDTSLineReader](#) is returned by [SDTSTransfer::GetLayerIndexedReader\(\)](#).
- SLTAttr: An attribute primary or secondary layer. An [SDTSAttrReader](#) is returned by [SDTSTransfer::GetLayerIndexedReader\(\)](#).
- SLTPoly: A polygon layer. An [SDTSPolygonReader](#) is returned by [SDTSTransfer::GetLayerIndexedReader\(\)](#).
- SLTRaster: A raster layer. [SDTSTransfer::GetLayerIndexedReader\(\)](#) is not implemented. Use [SDTSTransfer::GetLayerRasterReader\(\)](#) instead.

9.54.2.9 SDTS_XREF* SDTSTransfer::GetXREF () [inline]

Fetch the external reference object for this transfer.

Returns

pointer to the internally managed [SDTS_XREF](#) for the transfer.

9.54.2.10 int SDTSTransfer::Open (const char * *pszFilename*)

Open an SDTS transfer, and establish a list of data layers in the transfer.

Parameters

<i>pszFilename</i>	The name of the CATD file within the transfer.
--------------------	--

Returns

TRUE if the open success, or FALSE if it fails.

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

- sdt_s_al.h
- sdtstransfer.cpp

9.55 SFRegion Class Reference

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

- cpl_vsil_sparsefile.cpp

9.56 StackContext Struct Reference

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

- cpl_minixml.cpp

9.57 tm_unz_s Struct Reference

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

- cpl_minizip_unzip.h

9.58 tm_zip_s Struct Reference

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

- cpl_minizip_zip.h

9.59 unz_file_info_internal_s Struct Reference

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

- cpl_minizip_unzip.cpp

9.60 unz_file_info_s Struct Reference

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

- cpl_minizip_unzip.h

9.61 unz_file_pos_s Struct Reference

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

- cpl_minizip_unzip.h

9.62 unz_global_info_s Struct Reference

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

- cpl_minizip_unzip.h

9.63 unz_s Struct Reference

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

- cpl_minizip_unzip.cpp

9.64 VSIArchiveContent Struct Reference

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

- cpl_vsi_virtual.h

9.65 VSIArchiveEntry Struct Reference

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

- cpl_vsi_virtual.h

9.66 VSIArchiveEntryFileOffset Class Reference

Inheritance diagram for VSIArchiveEntryFileOffset:



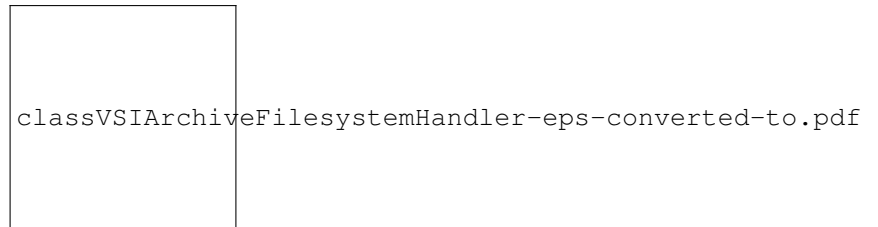
classVSIArchiveEntryFileOffset-eps-converted-to.pdf

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

- `cpl_vsi_virtual.h`
- `cpl_vsil_abstract_archive.cpp`

9.67 VSIArchiveFilesystemHandler Class Reference

Inheritance diagram for VSIArchiveFilesystemHandler:

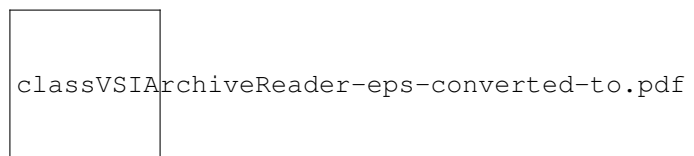


The documentation for this class was generated from the following files:

- `cpl_vsi_virtual.h`
- `cpl_vsil_abstract_archive.cpp`

9.68 VSIArchiveReader Class Reference

Inheritance diagram for VSIArchiveReader:

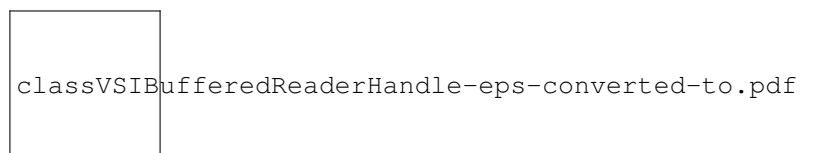


The documentation for this class was generated from the following files:

- `cpl_vsi_virtual.h`
- `cpl_vsil_abstract_archive.cpp`

9.69 VSIBufferedReaderHandle Class Reference

Inheritance diagram for VSIBufferedReaderHandle:



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

- `cpl_vsil_buffered_reader.cpp`

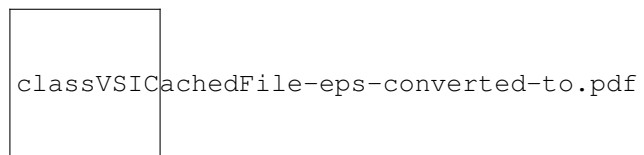
9.70 VSICacheChunk Class Reference

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

- `cpl_vsil_cache.cpp`

9.71 VSICachedFile Class Reference

Inheritance diagram for VSICachedFile:



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

- `cpl_vsil_cache.cpp`

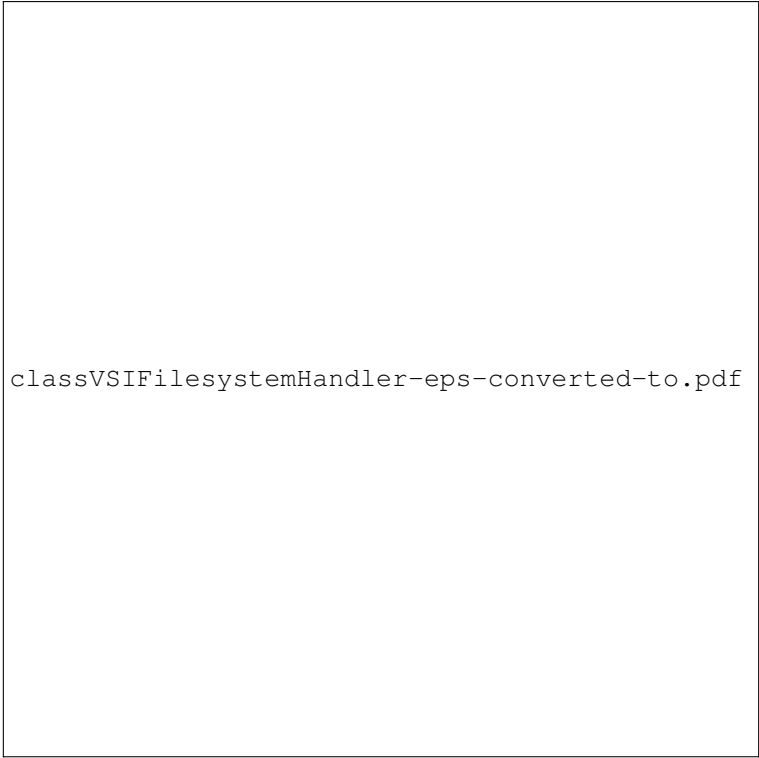
9.72 VSIFileManager Class Reference

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

- `cpl_vsi_virtual.h`
- `cpl_vsil.cpp`

9.73 VSIFilesystemHandler Class Reference

Inheritance diagram for VSIFilesystemHandler:



classVSIFilesystemHandler-eps-converted-to.pdf

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

- cpl_vsi_virtual.h

9.74 VSIGZipFilesystemHandler Class Reference

Inheritance diagram for VSIGZipFilesystemHandler:



classVSIGZipFilesystemHandler-eps-converted-to.pdf

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

- cpl_vsil_gzip.cpp

9.75 VSIGZipHandle Class Reference

Inheritance diagram for VSIGZipHandle:



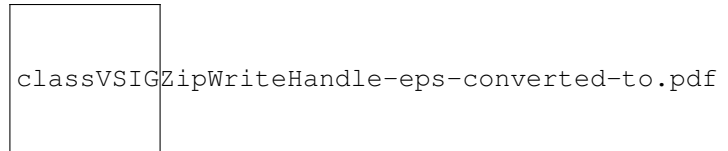
classVSIGZipHandle-eps-converted-to.pdf

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

- `cpl_vsil_gzip.cpp`

9.76 VSIGZipWriteHandle Class Reference

Inheritance diagram for VSIGZipWriteHandle:



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

- `cpl_vsil_gzip.cpp`

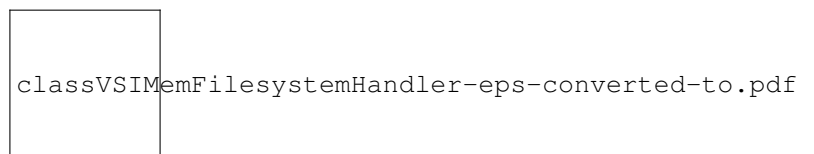
9.77 VSIMemFile Class Reference

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

- `cpl_vsi_mem.cpp`

9.78 VSIMemFilesystemHandler Class Reference

Inheritance diagram for VSIMemFilesystemHandler:

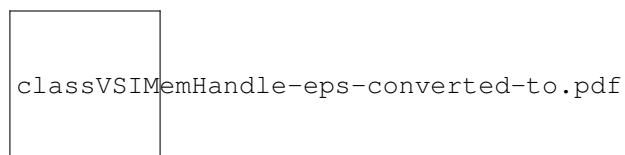


The documentation for this class was generated from the following file:

- `cpl_vsi_mem.cpp`

9.79 VSIMemHandle Class Reference

Inheritance diagram for VSIMemHandle:



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

- `cpl_vsi_mem.cpp`

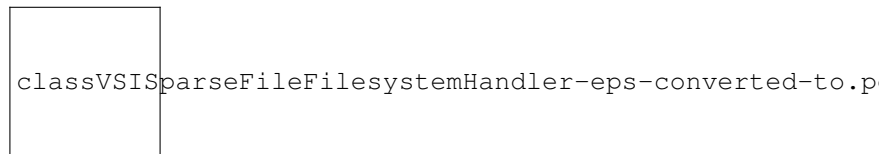
9.80 VSIReadDirRecursiveTask Struct Reference

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

- `cpl_vsil.cpp`

9.81 VSISparseFileFilesystemHandler Class Reference

Inheritance diagram for VSISparseFileFilesystemHandler:

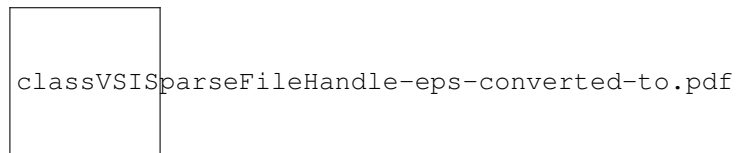


The documentation for this class was generated from the following file:

- `cpl_vsil_sparsefile.cpp`

9.82 VSISparseFileHandle Class Reference

Inheritance diagram for VSISparseFileHandle:

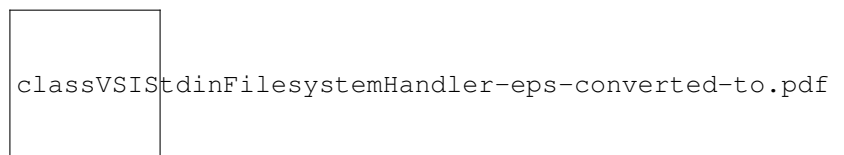


The documentation for this class was generated from the following file:

- `cpl_vsil_sparsefile.cpp`

9.83 VSIStdinFilesystemHandler Class Reference

Inheritance diagram for VSIStdinFilesystemHandler:

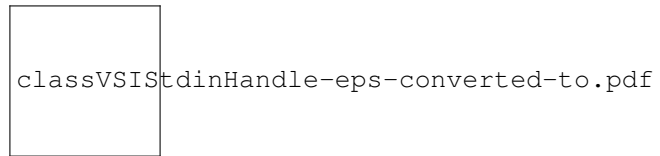


The documentation for this class was generated from the following file:

- `cpl_vsil_stdin.cpp`

9.84 VSIStdinHandle Class Reference

Inheritance diagram for VSIStdinHandle:

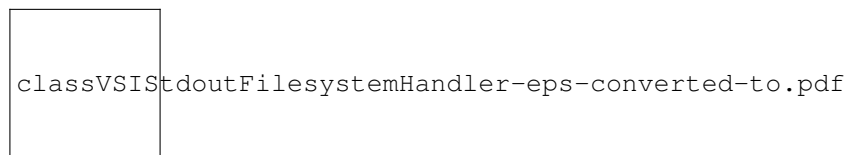


The documentation for this class was generated from the following file:

- cpl_vsil_stdin.cpp

9.85 VSIStdoutFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutFilesystemHandler:

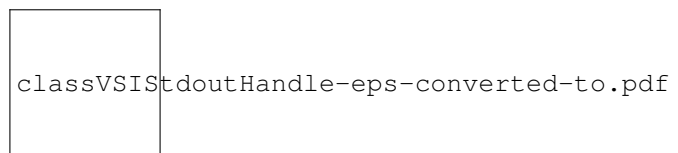


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

9.86 VSIStdoutHandle Class Reference

Inheritance diagram for VSIStdoutHandle:

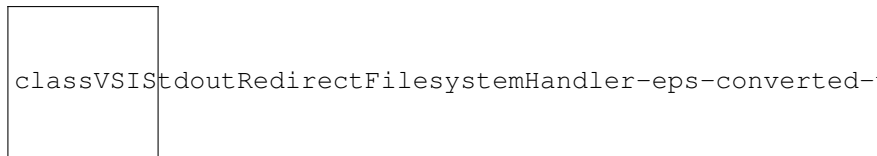


The documentation for this class was generated from the following file:

- cpl_vsil_stdout.cpp

9.87 VSIStdoutRedirectFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutRedirectFilesystemHandler:

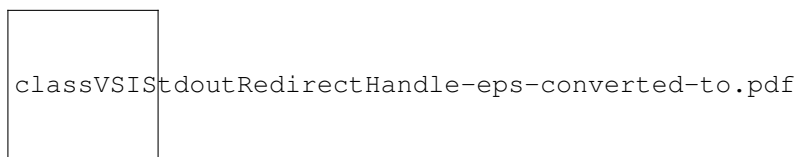


The documentation for this class was generated from the following file:

- `cpl_vsil_stdout.cpp`

9.88 VSIStdoutRedirectHandle Class Reference

Inheritance diagram for VSIStdoutRedirectHandle:

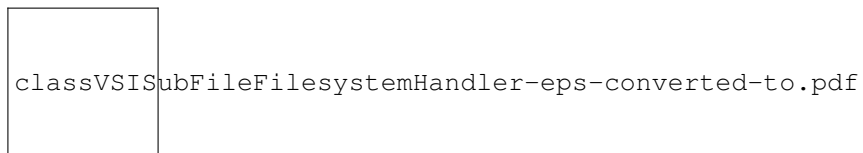


The documentation for this class was generated from the following file:

- `cpl_vsil_stdout.cpp`

9.89 VSISubFileFilesystemHandler Class Reference

Inheritance diagram for VSISubFileFilesystemHandler:

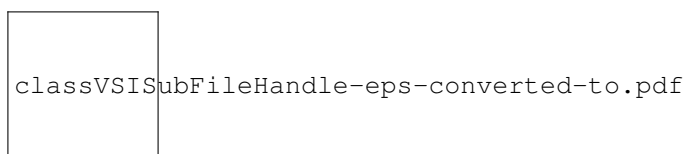


The documentation for this class was generated from the following file:

- `cpl_vsil_subfile.cpp`

9.90 VSISubFileHandle Class Reference

Inheritance diagram for VSISubFileHandle:

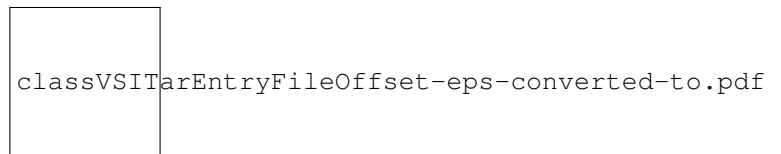


The documentation for this class was generated from the following file:

- `cpl_vsil_subfile.cpp`

9.91 VSITarEntryFileOffset Class Reference

Inheritance diagram for VSITarEntryFileOffset:

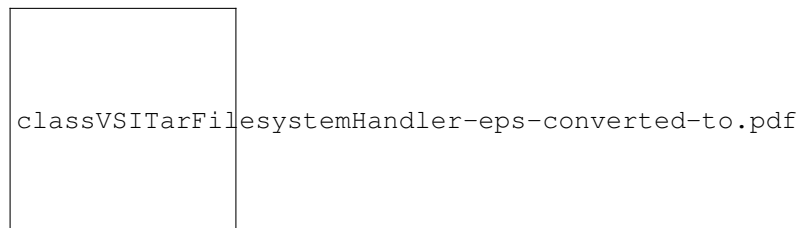


The documentation for this class was generated from the following file:

- `cpl_vsil_tar.cpp`

9.92 VSITarFilesystemHandler Class Reference

Inheritance diagram for VSITarFilesystemHandler:

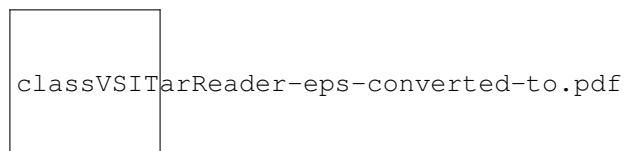


The documentation for this class was generated from the following file:

- `cpl_vsil_tar.cpp`

9.93 VSITarReader Class Reference

Inheritance diagram for VSITarReader:

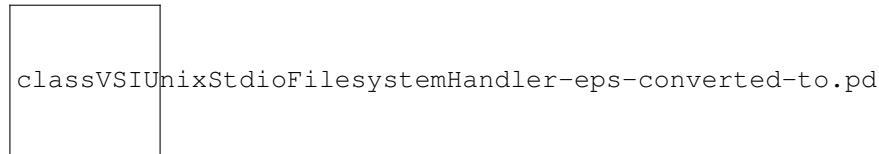


The documentation for this class was generated from the following file:

- `cpl_vsil_tar.cpp`

9.94 VSIUnixStdioFilesystemHandler Class Reference

Inheritance diagram for VSIUnixStdioFilesystemHandler:

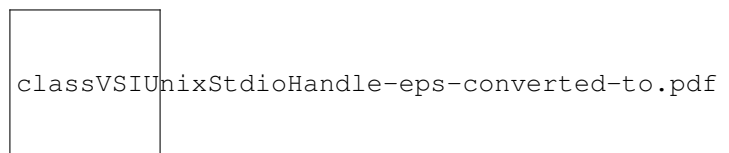


The documentation for this class was generated from the following file:

- cpl_vsil_unix_stdio_64.cpp

9.95 VSIUnixStdioHandle Class Reference

Inheritance diagram for VSIUnixStdioHandle:



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

- cpl_vsil_unix_stdio_64.cpp

9.96 VSIVirtualHandle Class Reference

Inheritance diagram for VSIVirtualHandle:

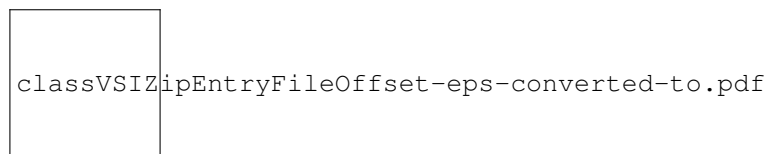


The documentation for this class was generated from the following files:

- cpl_vsi_virtual.h
- cpl_vsil.cpp

9.97 VSZipEntryFileOffset Class Reference

Inheritance diagram for VSZipEntryFileOffset:

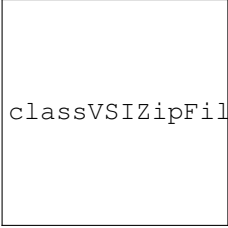


The documentation for this class was generated from the following file:

- cpl_vsil_gzip.cpp

9.98 VSZipFilesystemHandler Class Reference

Inheritance diagram for VSZipFilesystemHandler:



classVSZipFilesystemHandler-eps-converted-to.pdf

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

- cpl_vsil_gzip.cpp

9.99 VSZipReader Class Reference

Inheritance diagram for VSZipReader:



classVSZipReader-eps-converted-to.pdf

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

- cpl_vsil_gzip.cpp

9.100 VSZipWriteHandle Class Reference

Inheritance diagram for VSZipWriteHandle:



classVSZipWriteHandle-eps-converted-to.pdf

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

- cpl_vsil_gzip.cpp

9.101 zip_fileinfo Struct Reference

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

- cpl_minizip_zip.h

9.102 zip_internal Struct Reference

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

- cpl_minizip_zip.cpp

9.103 zlib_filefunc_def_s Struct Reference

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

- cpl_minizip_ioapi.h

Chapter 10

File Documentation

10.1 cpl_conv.h File Reference

```
#include "cpl_port.h"
#include "cpl_vsi.h"
#include "cpl_error.h"
```

Classes

- struct [CPLSharedFileInfo](#)

Functions

- const char CPL_DLL *CPL_STDCALL [CPLGetConfigOption](#) (const char *, const char *) CPL_WARN_UNUSED_RESULT
- void CPL_DLL CPL_STDCALL [CPLSetConfigOption](#) (const char *, const char *)
- void CPL_DLL CPL_STDCALL [CPLSetThreadLocalConfigOption](#) (const char *pszKey, const char *pszValue)
- void CPL_DLL * [CPLMalloc](#) (size_t) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * [CPLCalloc](#) (size_t, size_t) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * [CPLRealloc](#) (void *, size_t) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * [CPLStrdup](#) (const char *) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * [CPLStrlwr](#) (char *)
- char CPL_DLL * [CPLFGets](#) (char *, int, FILE *)
- const char CPL_DLL * [CPLReadLine](#) (FILE *)
- const char CPL_DLL * [CPLReadLineL](#) (VSILFILE *)
- const char CPL_DLL * [CPLReadLine2L](#) (VSILFILE *, int nMaxCols, char **papszOptions)
- double CPL_DLL [CPLAtof](#) (const char *)
- double CPL_DLL [CPLAtofDelim](#) (const char *, char)
- double CPL_DLL [CPLStrtod](#) (const char *, char **)
- double CPL_DLL [CPLStrtodDelim](#) (const char *, char **, char)
- float CPL_DLL [CPLStrtof](#) (const char *, char **)
- float CPL_DLL [CPLStrtofDelim](#) (const char *, char **, char)
- double CPL_DLL [CPLAtofM](#) (const char *)
- char CPL_DLL * [CPLScanString](#) (const char *, int, int, int)
- double CPL_DLL [CPLScanDouble](#) (const char *, int)
- long CPL_DLL [CPLScanLong](#) (const char *, int)
- unsigned long CPL_DLL [CPLScanULong](#) (const char *, int)
- GUIntBig CPL_DLL [CPLScanUIntBig](#) (const char *, int)

- void CPL_DLL * [CPLScanPointer](#) (const char *, int)
- int CPL_DLL [CPLPrintString](#) (char *, const char *, int)
- int CPL_DLL [CPLPrintStringFill](#) (char *, const char *, int)
- int CPL_DLL [CPLPrintInt32](#) (char *, GLint32, int)
- int CPL_DLL [CPLPrintUIntBig](#) (char *, GLuintBig, int)
- int CPL_DLL [CPLPrintDouble](#) (char *, const char *, double, const char *)
- int CPL_DLL [CPLPrintTime](#) (char *, int, const char *, const struct tm *, const char *)
- int CPL_DLL [CPLPrintPointer](#) (char *, void *, int)
- void CPL_DLL * [CPLGetSymbol](#) (const char *, const char *)
- int CPL_DLL [CPLGetExecPath](#) (char *pszPathBuf, int nMaxLength)
- const char CPL_DLL * [CPLGetPath](#) (const char *)
- const char CPL_DLL * [CPLGetDirname](#) (const char *)
- const char CPL_DLL * [CPLGetFilename](#) (const char *)
- const char CPL_DLL * [CPLGetBasename](#) (const char *)
- const char CPL_DLL * [CPLGetExtension](#) (const char *)
- char CPL_DLL * [CPLGetCurrentDir](#) (void)
- const char CPL_DLL * [CPLFormFilename](#) (const char *pszPath, const char *pszBasename, const char *pszExtension)
- const char CPL_DLL * [CPLFormCIFilename](#) (const char *pszPath, const char *pszBasename, const char *pszExtension)
- const char CPL_DLL * [CPLResetExtension](#) (const char *, const char *)
- const char CPL_DLL * [CPLProjectRelativeFilename](#) (const char *pszProjectDir, const char *pszSecondaryFilename)
- int CPL_DLL [CPLIsFilenameRelative](#) (const char *pszFilename)
- const char CPL_DLL * [CPLExtractRelativePath](#) (const char *, const char *, int *)
- const char CPL_DLL * [CPLCleanTrailingSlash](#) (const char *)
- char CPL_DLL ** [CPLCorrespondingPaths](#) (const char *pszOldFilename, const char *pszNewFilename, char **papszFileList)
- int CPL_DLL [CPLCheckForFile](#) (char *pszFilename, char **papszSiblingList)
- const char CPL_DLL * [CPLGenerateTempFilename](#) (const char *pszStem)
- FILE CPL_DLL * [CPLOpenShared](#) (const char *, const char *, int)
- void CPL_DLL [CPLCloseShared](#) (FILE *)
- [CPLSharedFileInfo](#) CPL_DLL * [CPLGetSharedList](#) (int *)
- void CPL_DLL [CPLDumpSharedList](#) (FILE *)
- double CPL_DLL [CPLPackedDMSToDec](#) (double)
- double CPL_DLL [CPLDecToPackedDMS](#) (double dfDec)
- int CPL_DLL [CPLUnlinkTree](#) (const char *)
- void CPL_DLL * [CPLZLibDeflate](#) (const void *ptr, size_t nBytes, int nLevel, void *outptr, size_t nOutAvailableBytes, size_t *pnOutBytes)
- Compress a buffer with ZLib DEFLATE compression.*
- void CPL_DLL * [CPLZLibInflate](#) (const void *ptr, size_t nBytes, void *outptr, size_t nOutAvailableBytes, size_t *pnOutBytes)
- Uncompress a buffer compressed with ZLib DEFLATE compression.*

10.1.1 Detailed Description

Various convenience functions for CPL.

10.1.2 Function Documentation

10.1.2.1 double CPL_DLL 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\(\)](#) function if you want to specify custom delimiter.

IMPORTANT NOTE. Existence of this function does not mean you should always use it. Sometimes you should use standard locale aware `atof(3)` and its family. When you need to process the user's input (for example, command line parameters) use `atof(3)`, because user works in localized environment and her input will be done accordingly the locale set. In particular that means we should not make assumptions about character used as decimal delimiter, it can be either "." or ",". But when you are parsing some ASCII file in predefined format, you most likely need [CPLAtof\(\)](#), because such files distributed across the systems with different locales and floating point representation should be considered as a part of file format. If the format uses "." as a delimiter the same character must be used when parsing number regardless of actual locale setting.

Parameters

<i>nptr</i>	Pointer to string to convert.
-------------	-------------------------------

Returns

Converted value, if any.

10.1.2.2 double CPL_DLL 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\(\)](#) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>point</i>	Decimal delimiter.

Returns

Converted value, if any.

10.1.2.3 double CPL_DLL CPLAtofM (const char * *nptr*)

Converts ASCII string to floating point number using any numeric locale.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. This function does the same as standard `atof()`, but it allows a variety of locale representations. That is it supports numeric values with either a comma or a period for the decimal delimiter.

PS. The M stands for Multi-lingual.

Parameters

<i>nptr</i>	The string to convert.
-------------	------------------------

Returns

Converted value, if any. Zero on failure.

10.1.2.4 void CPL_DLL* 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() 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\(\)](#) as well. CPLFree() or VSIFree() can be used free memory allocated by [CPLCalloc\(\)](#).

Parameters

<i>nCount</i>	number of objects to allocate.
<i>nSize</i>	size (in bytes) of object to allocate.

Returns

pointer to newly allocated memory, only NULL if nSize * nCount is NULL.

10.1.2.5 int CPL_DLL CPLCheckForFile (char * pszFilename, char ** papszSiblingFiles)

Check for file existence.

The function checks if a named file exists in the filesystem, hopefully in an efficient fashion if a sibling file list is available. It exists primarily to do faster file checking for functions like GDAL open methods that get a list of files from the target directory.

If the sibling file list exists (is not NULL) it is assumed to be a list of files in the same directory as the target file, and it will be checked (case insensitively) for a match. If a match is found, pszFilename is updated with the correct case and TRUE is returned.

If papszSiblingFiles is NULL, a [VSISatL\(\)](#) is used to test for the files existence, and no case insensitive testing is done.

Parameters

<i>pszFilename</i>	name of file to check for - filename case updated in some cases.
<i>papszSiblingFiles</i>	a list of files in the same directory as pszFilename if available, or NULL. This list should have no path components.

Returns

TRUE if a match is found, or FALSE if not.

10.1.2.6 const char CPL_DLL* CPLCleanTrailingSlash (const char * pszPath)

Remove trailing forward/backward slash from the path for unix/windows resp.

Returns a string containing the portion of the passed path string with trailing slash removed. If there is no path in the passed filename an empty string will be returned (not NULL).

```

CPLCleanTrailingSlash( "abc/def/" ) == "abc/def"
CPLCleanTrailingSlash( "abc/def" ) == "abc/def"
CPLCleanTrailingSlash( "c:\abc\def\" ) == "c:\abc\def"
CPLCleanTrailingSlash( "c:\abc\def" ) == "c:\abc\def"
CPLCleanTrailingSlash( "abc" ) == "abc"

```

Parameters

<i>pszPath</i>	the path to be cleaned up
----------------	---------------------------

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

10.1.2.7 void CPL_DLL CPLCloseShared (FILE * *fp*)

Close shared file.

Dereferences the indicated file handle, and closes it if the reference count has dropped to zero. A CPLError() is issued if the file is not in the shared file list.

Parameters

<i>fp</i>	file handle from CPLOpenShared() to deaccess.
-----------	---

10.1.2.8 char CPL_DLL 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\(\)](#).

Parameters

<i>pszOldFilename</i>	path to old prototype file.
<i>pszNew-Filename</i>	path to new prototype file.
<i>papszFileList</i>	list of other files associated with <i>pszOldFilename</i> to rename similarly.

Returns

a list of files corresponding to *papszFileList* but renamed to correspond to *pszNewFilename*.

10.1.2.9 double CPL_DLL 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\(\)](#).

Parameters

<i>dfDec</i>	Angle in decimal degrees.
--------------	---------------------------

Returns

Angle in packed DMS format.

10.1.2.10 void CPL_DLL 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() call.

Parameters

<i>fp</i>	File handle to write to.
-----------	--------------------------

10.1.2.11 const char CPL_DLL* CPLExtractRelativePath (const char * *pszBaseDir*, const char * *pszTarget*, int * *pbGotRelative*)

Get relative path from directory to target file.

Computes a relative path for *pszTarget* relative to *pszBaseDir*. Currently this only works if they share a common base path. The returned path is normally into the *pszTarget* string. It should only be considered valid as long as *pszTarget* is valid or till the next call to this function, whichever comes first.

Parameters

<i>pszBaseDir</i>	the name of the directory relative to which the path should be computed. <i>pszBaseDir</i> may be NULL in which case the original target is returned without relitizing.
<i>pszTarget</i>	the filename to be changed to be relative to <i>pszBaseDir</i> .
<i>pbGotRelative</i>	Pointer to location in which a flag is placed indicating that the returned path is relative to the basename (TRUE) or not (FALSE). This pointer may be NULL if flag is not desired.

Returns

an adjusted path or the original if it could not be made relative to the *pszBaseFile*'s path.

10.1.2.12 char CPL_DLL* 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\(\)](#): single '\r' and '\n' and '\r\n' combination.

Parameters

<i>pszBuffer</i>	pointer to the targeting character buffer.
<i>nBufferSize</i>	maximum size of the string to read (not including terminating '\0').
<i>fp</i>	file pointer to read from.

Returns

pointer to the pszBuffer containing a string read from the file or NULL if the error or end of file was encountered.

10.1.2.13 `const char CPL_DLL* CPLFormCIFilename ( const char * pszPath, const char * pszBasename, const char * pszExtension )`

Case insensitive file searching, returning full path.

This function tries to return the path to a file regardless of whether the file exactly matches the basename, and extension case, or is all upper case, or all lower case. The path is treated as case sensitive. This function is equivalent to `CPLFormFilename()` on case insensitive file systems (like Windows).

Parameters

<i>pszPath</i>	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
<i>pszBasename</i>	file basename. May optionally have path and/or extension. May not be NULL.
<i>pszExtension</i>	file extension, optionally including the period. May be NULL.

Returns

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

10.1.2.14 `const char CPL_DLL* CPLFormFilename ( const char * pszPath, const char * pszBasename, const char * pszExtension )`

Build a full file path from a passed path, file basename and extension.

The path, and extension are optional. The basename may in fact contain an extension if desired.

```
CPLFormFilename("abc/xyz", "def", ".dat" ) == "abc/xyz/def.dat"
CPLFormFilename(NULL, "def", NULL ) == "def"
CPLFormFilename(NULL, "abc/def.dat", NULL ) == "abc/def.dat"
CPLFormFilename("/abc/xyz/", "def.dat", NULL ) == "/abc/xyz/def.dat"
```

Parameters

<i>pszPath</i>	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
<i>pszBasename</i>	file basename. May optionally have path and/or extension. Must <i>NOT</i> be NULL.
<i>pszExtension</i>	file extension, optionally including the period. May be NULL.

Returns

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

10.1.2.15 `const char CPL_DLL* CPLGenerateTempFilename ( const char * pszStem )`

Generate temporary file name.

Returns a filename that may be used for a temporary file. The location of the file tries to follow operating system semantics but may be forced via the CPL_TMPDIR configuration option.

Parameters

<i>pszStem</i>	if non-NULL this will be part of the filename.
----------------	--

Returns

a filename which is valid till the next CPL call in this thread.

10.1.2.16 const char CPL_DLL* CPLGetBasename (const char * *pszFullFilename*)

Extract basename (non-directory, non-extension) portion of filename.

Returns a string containing the file basename portion of the passed name. If there is no basename (passed value ends in trailing directory separator, or filename starts with a dot) an empty string is returned.

```
CPLGetBasename( "abc/def.xyz" ) == "def"
CPLGetBasename( "abc/def" ) == "def"
CPLGetBasename( "abc/def/" ) == ""
```

Parameters

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

Returns

just the non-directory, non-extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

10.1.2.17 const char CPL_DLL* CPL_STD_CALL CPLGetConfigOption (const char * *pszKey*, const char * *pszDefault*)

Get the value of a configuration option.

The value is the value of a (key, value) option set with [CPLSetConfigOption\(\)](#). If the given option was not defined with [CPLSetConfigOption\(\)](#), it tries to find it in environment variables.

Note: the string returned by [CPLGetConfigOption\(\)](#) might be short-lived, and in particular it will become invalid after a call to [CPLSetConfigOption\(\)](#) with the same key.

To override temporary a potentially existing option with a new value, you can use the following snippet :

```
// backup old value
const char* pszOldValTmp = CPLGetConfigOption(pszKey, NULL);
char* pszOldVal = pszOldValTmp ? CPLStrdup(pszOldValTmp) : NULL;
// override with new value
CPLSetConfigOption(pszKey, pszNewVal);
// do something usefull
// restore old value
CPLSetConfigOption(pszKey, pszOldVal);
CPLFree(pszOldVal);
```

Parameters

<i>pszKey</i>	the key of the option to retrieve
<i>pszDefault</i>	a default value if the key does not match existing defined options (may be NULL)

Returns

the value associated to the key, or the default value if not found

See Also

[CPLSetConfigOption\(\)](#), <http://trac.osgeo.org/gdal/wiki/ConfigOptions>

10.1.2.18 char CPL_DLL* 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.

10.1.2.19 const char CPL_DLL* 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\(\)](#).

```
CPLGetDirname( "abc/def.xyz" ) == "abc"
CPLGetDirname( "/abc/def/" ) == "/abc/def"
CPLGetDirname( "/" ) == "/"
CPLGetDirname( "/abc/def" ) == "/abc"
CPLGetDirname( "abc" ) == "."
```

Parameters

<i>pszFilename</i>	the filename potentially including a path.
--------------------	--

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

10.1.2.20 int CPL_DLL CPLGetExecPath (char * *pszPathBuf*, int *nMaxLength*)

Fetch path of executable.

The path to the executable currently running is returned. This path includes the name of the executable. Currently this only works on win32 and linux platforms. The returned path is UTF-8 encoded.

Parameters

<i>pszPathBuf</i>	the buffer into which the path is placed.
<i>nMaxLength</i>	the buffer size, MAX_PATH+1 is suggested.

Returns

FALSE on failure or TRUE on success.

10.1.2.21 `const char CPL_DLL* CPLGetExtension ( const char * pszFullFilename )`

Extract filename extension from full filename.

Returns a string containing the extension portion of the passed name. If there is no extension (the filename has no dot) an empty string is returned. The returned extension will not include the period.

```
CPLGetExtension( "abc/def.xyz" ) == "xyz"  
CPLGetExtension( "abc/def" ) == ""
```

Parameters

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

Returns

just the extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

10.1.2.22 `const char CPL_DLL* CPLGetFilename ( const char * pszFullFilename )`

Extract non-directory portion of filename.

Returns a string containing the bare filename portion of the passed filename. If there is no filename (passed value ends in trailing directory separator) an empty string is returned.

```
CPLGetFilename( "abc/def.xyz" ) == "def.xyz"  
CPLGetFilename( "/abc/def/" ) == ""  
CPLGetFilename( "abc/def" ) == "def"
```

Parameters

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

Returns

just the non-directory portion of the path (points back into original string).

10.1.2.23 `const char CPL_DLL* CPLGetPath ( const char * pszFilename )`

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLGetPath( "abc/def.xyz" ) == "abc"  
CPLGetPath( "/abc/def/" ) == "/abc/def"  
CPLGetPath( "/" ) == "/"  
CPLGetPath( "/abc/def" ) == "/abc"  
CPLGetPath( "abc" ) == ""
```

Parameters

<i>pszFilename</i>	the filename potentially including a path.
--------------------	--

Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

10.1.2.24 CPLSharedFileInfo CPL_DLL* CPLGetSharedList (int * *pnCount*)

Fetch list of open shared files.

Parameters

<i>pnCount</i>	place to put the count of entries.
----------------	------------------------------------

Returns

the pointer to the first in the array of shared file info structures.

10.1.2.25 void CPL_DLL* 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 `CPL_Error()` or not are not currently well defined, and will have to be resolved in the future.

Currently [CPLGetSymbol\(\)](#) doesn't try to:

- prevent the reference count on the library from going up for every request, or given any opportunity to unload the library.
- Attempt to look for the library in non-standard locations.
- Attempt to try variations on the symbol name, like pre-pending or post-pending an underscore.

Some of these issues may be worked on in the future.

Parameters

<i>pszLibrary</i>	the name of the shared library or DLL containing the function. May contain path to file. If not system supplies search paths will be used.
<i>pszSymbolName</i>	the name of the function to fetch a pointer to.

Returns

A pointer to the function if found, or NULL if the function isn't found, or the shared library can't be loaded.

10.1.2.26 int CPL_DLL CPLIsFilenameRelative (const char * *pszFilename*)

Is filename relative or absolute?

The test is filesystem convention agnostic. That is it will test for Unix style and windows style path conventions regardless of the actual system in use.

Parameters

<i>pszFilename</i>	the filename with path to test.
--------------------	---------------------------------

Returns

TRUE if the filename is relative or FALSE if it is absolute.

10.1.2.27 void CPL_DLL* CPLMalloc (size_t nSize)

Safe version of malloc().

This function is like the C library malloc(), but raises a CE_Fatal error with CPLError() 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() as well. CPLFree() or VSIFree() can be used free memory allocated by CPLMalloc().

Parameters

<i>nSize</i>	size (in bytes) of memory block to allocate.
--------------	--

Returns

pointer to newly allocated memory, only NULL if nSize is zero.

10.1.2.28 FILE CPL_DLL* 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() function is used to actually open the file, when an existing file handle can't be shared.

Parameters

<i>pszFilename</i>	the name of the file to open.
<i>pszAccess</i>	the normal fopen()/VSIFOpen() style access string.
<i>bLarge</i>	If TRUE VSIFOpenL() (for large files) will be used instead of VSIFOpen().

Returns

a file handle or NULL if opening fails.

10.1.2.29 double CPL_DLL CPLPackedDMSToDec (double dfPacked)

Convert a packed DMS value (DDDDMMSS.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.
1. The degrees are separated out: $\text{deg} = \text{ang} / 1000000$ (fractional portion truncated)
1. The minutes are separated out: $\text{min} = (\text{ang} - \text{deg} * 1000000) / 1000$ (fractional portion truncated)
1. The seconds are then computed: $\text{sec} = \text{ang} - \text{deg} * 1000000 - \text{min} * 1000$
1. The total angle in seconds is computed: $\text{sec} = \text{deg} * 3600.0 + \text{min} * 60.0 + \text{sec}$
1. The sign of sec is set to that of the input angle.

Packed DMS values used by the USGS GCTP package and probably by other software.

NOTE: This code does not validate input value. If you give the wrong value, you will get the wrong result.

Parameters

<i>dfPacked</i>	Angle in packed DMS format.
-----------------	-----------------------------

Returns

Angle in decimal degrees.

10.1.2.30 `int CPL_DLL CPLPrintDouble ( char * pszBuffer, const char * pszFormat, double dfValue, const char * pszLocale )`

Print double value into specified string buffer. Exponential character flag 'E' (or 'e') will be replaced with 'D', as in Fortran. Resulting string will not to be NULL-terminated.

Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>pszFormat</i>	Format specifier (for example, "%16.9E").
<i>dfValue</i>	Numerical value to print.
<i>pszLocale</i>	Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. With the pszLocale option we can control what exact locale will be used for printing a numeric value to the string (in most cases it should be C/POSIX).

Returns

Number of characters printed.

10.1.2.31 `int CPL_DLL CPLPrintInt32 ( char * pszBuffer, GInt32 iValue, int nMaxLen )`

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
------------------	--

<i>iValue</i>	Numerical value to print.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

10.1.2.32 int CPL_DLL CPLPrintPointer (char * *pszBuffer*, void * *pValue*, int *nMaxLen*)

Print pointer value into specified string buffer. This string will not be NULL-terminated.

Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>pValue</i>	Pointer to ASCII encode.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

10.1.2.33 int CPL_DLL CPLPrintString (char * *pszDest*, const char * *pszSrc*, int *nMaxLen*)

Copy the string pointed to by *pszSrc*, NOT including the terminating '\0' character, to the array pointed to by *pszDest*.

Parameters

<i>pszDest</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
<i>pszSrc</i>	Pointer to the source buffer.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

10.1.2.34 int CPL_DLL CPLPrintStringFill (char * *pszDest*, const char * *pszSrc*, int *nMaxLen*)

Copy the string pointed to by *pszSrc*, NOT including the terminating '\0' character, to the array pointed to by *pszDest*. Remainder of the destination string will be filled with space characters. This is only difference from the *PrintString()*.

Parameters

<i>pszDest</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
<i>pszSrc</i>	Pointer to the source buffer.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

10.1.2.35 `int CPL_DLL CPLPrintTime ( char * pszBuffer, int nMaxLen, const char * pszFormat, const struct tm * poBrokenTime, const char * pszLocale )`

Print specified time value accordingly to the format options and specified locale name. This function does following:

- if locale parameter is not NULL, the current locale setting will be stored and replaced with the specified one;
- format time value with the `strftime(3)` function;
- restore back current locale, if was saved.

Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.
<i>pszFormat</i>	Controls the output format. Options are the same as for <code>strftime(3)</code> function.
<i>poBrokenTime</i>	Pointer to the broken-down time structure. May be requested with the <code>VSIGMTIME()</code> and <code>VSI-LocalTime()</code> functions.
<i>pszLocale</i>	Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. Be aware that it may be unsuitable to use current locale for printing time, because all names will be printed in your native language, as well as time format settings also may be adjusted differently from the C/POSIX defaults. To solve these problems this option was introduced.

Returns

Number of characters printed.

10.1.2.36 `int CPL_DLL CPLPrintUIntBig ( char * pszBuffer, GUIntBig iValue, int nMaxLen )`

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>iValue</i>	Numerical value to print.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

Returns

Number of characters printed.

10.1.2.37 `const char CPL_DLL* CPLProjectRelativeFilename ( const char * pszProjectDir, const char * pszSecondaryFilename )`

Find a file relative to a project file.

Given the path to a "project" directory, and a path to a secondary file referenced from that project, build a path to the secondary file that the current application can use. If the secondary path is already absolute, rather than relative, then it will be returned unaltered.

Examples:

```
CPLProjectRelativeFilename("abc/def", "tmp/abc.gif") == "abc/def/tmp/abc.gif"
CPLProjectRelativeFilename("abc/def", "/tmp/abc.gif") == "/tmp/abc.gif"
CPLProjectRelativeFilename("/xy", "abc.gif") == "/xy/abc.gif"
CPLProjectRelativeFilename("/abc/def", "../abc.gif") == "/abc/def/../abc.gif"
CPLProjectRelativeFilename("C:\\WIN", "abc.gif") == "C:\\WIN\\abc.gif"
```

Parameters

<i>pszProjectDir</i>	the directory relative to which the secondary files path should be interpreted.
<i>pszSecondary-Filename</i>	the filename (potentially with path) that is to be interpreted relative to the project directory.

Returns

a composed path to the secondary file. The returned string is internal and should not be altered, freed, or depending on past the next CPL call.

10.1.2.38 `const char CPL_DLL* CPLReadLine ( FILE * fp )`

Simplified line reading from text file.

Read a line of text from the given file handle, taking care to capture CR and/or LF and strip off ... equivalent of DKReadLine(). Pointer to an internal buffer is returned. The application shouldn't free it, or depend on it's value past the next call to [CPLReadLine\(\)](#).

Note that [CPLReadLine\(\)](#) uses VSIFGets(), so any hooking of VSI file services should apply to [CPLReadLine\(\)](#) as well.

[CPLReadLine\(\)](#) maintains an internal buffer, which will appear as a single block memory leak in some circumstances. [CPLReadLine\(\)](#) may be called with a NULL FILE * at any time to free this working buffer.

Parameters

<i>fp</i>	file pointer opened with VSIFOpen().
-----------	--------------------------------------

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

10.1.2.39 `const char CPL_DLL* CPLReadLine2L ( VSILFILE * fp, int nMaxCars, char ** papszOptions )`

Simplified line reading from text file.

Similar to [CPLReadLine\(\)](#), but reading from a large file API handle.

Parameters

<i>fp</i>	file pointer opened with VSIFOpenL() .
<i>nMaxCars</i>	maximum number of characters allowed, or -1 for no limit.
<i>papszOptions</i>	NULL-terminated array of options. Unused for now.

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered or the maximum number of characters allowed reached.

Since

GDAL 1.7.0

10.1.2.40 `const char CPL_DLL* CPLReadLineL ( VSILFILE * fp )`

Simplified line reading from text file.

Similar to [CPLReadLine\(\)](#), but reading from a large file API handle.

Parameters

<i>fp</i>	file pointer opened with VSIFOpenL() .
-----------	--

Returns

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

10.1.2.41 `void CPL_DLL* 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() 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 VSIFRealloc() to get the memory, so any hooking of VSIFRealloc() will apply to [CPLRealloc\(\)](#) as well. CPLFree() or VSIFree() can be used free memory allocated by [CPLRealloc\(\)](#).

It is also safe to pass NULL in as the existing memory block for [CPLRealloc\(\)](#), in which case it uses VSIMalloc() to allocate a new block.

Parameters

<i>pData</i>	existing memory block which should be copied to the new block.
<i>nNewSize</i>	new size (in bytes) of memory block to allocate.

Returns

pointer to allocated memory, only NULL if nNewSize is zero.

10.1.2.42 `const char CPL_DLL* CPLResetExtension ( const char * pszPath, const char * pszExt )`

Replace the extension with the provided one.

Parameters

<i>pszPath</i>	the input path, this string is not altered.
<i>pszExt</i>	the new extension to apply to the given path.

Returns

an altered filename with the new extension. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

10.1.2.43 double CPL_DLL 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\(\)](#) to convert string to double value, so it uses a comma as a decimal delimiter.

Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Double value, converted from its ASCII form.

10.1.2.44 long CPL_DLL CPLScanLong (const char * *pszString*, int *nMaxLength*)

Scan up to a maximum number of characters from a string and convert the result to a long.

Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Long value, converted from its ASCII form.

10.1.2.45 void CPL_DLL* CPLScanPointer (const char * *pszString*, int *nMaxLength*)

Extract pointer from string.

Scan up to a maximum number of characters from a string and convert the result to a pointer.

Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

pointer value, converted from its ASCII form.

10.1.2.46 char CPL_DLL* CPLScanString (const char * *pszString*, int *nMaxLength*, int *bTrimSpaces*, int *bNormalize*)

Scan up to a maximum number of characters from a given string, allocate a buffer for a new string and fill it with scanned characters.

Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to read. Less characters will be read if a null character is encountered.
<i>bTrimSpaces</i>	If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.
<i>bNormalize</i>	If TRUE, replace ':' symbol with the '_'. It is needed if resulting string will be used in CPL dictionaries.

Returns

Pointer to the resulting string buffer. Caller responsible to free this buffer with CPLFree().

10.1.2.47 GUIntBig CPL_DLL CPLScanUIntBig (const char * *pszString*, int *nMaxLength*)

Extract big integer from string.

Scan up to a maximum number of characters from a string and convert the result to a GUIntBig.

Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

GUIntBig value, converted from its ASCII form.

10.1.2.48 unsigned long CPL_DLL CPLScanULong (const char * *pszString*, int *nMaxLength*)

Scan up to a maximum number of characters from a string and convert the result to a unsigned long.

Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns

Unsigned long value, converted from its ASCII form.

10.1.2.49 void CPL_DLL CPL_STDCALL CPLSetConfigOption (const char * *pszKey*, const char * *pszValue*)

Set a configuration option for GDAL/OGR use.

Those options are defined as a (key, value) couple. The value corresponding to a key can be got later with the [CPLGetConfigOption\(\)](#) method.

This mechanism is similar to environment variables, but options set with [CPLSetConfigOption\(\)](#) overrides, for [CPLGetConfigOption\(\)](#) point of view, values defined in the environment.

If [CPLSetConfigOption\(\)](#) is called several times with the same key, the value provided during the last call will be used.

Options can also be passed on the command line of most GDAL utilities with the with '-config KEY VALUE'. For example, ogrinfo -config CPL_DEBUG ON ~/data/test/point.shp

This function can also be used to clear a setting by passing NULL as the value (note: passing NULL will not unset an existing environment variable; it will just unset a value previously set by [CPLSetConfigOption\(\)](#)).

Parameters

<i>pszKey</i>	the key of the option
<i>pszValue</i>	the value of the option, or NULL to clear a setting.

See Also

<http://trac.osgeo.org/gdal/wiki/ConfigOptions>

10.1.2.50 void CPL_DLL CPL_STDCALL CPLSetThreadLocalConfigOption (const char * *pszKey*, const char * *pszValue*)

Set a configuration option for GDAL/OGR use.

Those options are defined as a (key, value) couple. The value corresponding to a key can be got later with the [CPLGetConfigOption\(\)](#) method.

This function sets the configuration option that only applies in the current thread, as opposed to [CPLSetConfigOption\(\)](#) which sets an option that applies on all threads.

This function can also be used to clear a setting by passing NULL as the value (note: passing NULL will not unset an existing environment variable; it will just unset a value previously set by [CPLSetThreadLocalConfigOption\(\)](#)).

Parameters

<i>pszKey</i>	the key of the option
<i>pszValue</i>	the value of the option, or NULL to clear a setting.

10.1.2.51 char CPL_DLL* 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() instead of returning NULL. It uses VSIStrdup(), so any hooking of that function will apply to [CPLStrdup\(\)](#) as well. Memory allocated with [CPLStrdup\(\)](#) can be freed with CPLFree() or VSIFree().

It is also safe to pass a NULL string into [CPLStrdup\(\)](#). [CPLStrdup\(\)](#) will allocate and return a zero length string (as opposed to a NULL string).

Parameters

<i>pszString</i>	input string to be duplicated. May be NULL.
------------------	---

Returns

pointer to a newly allocated copy of the string. Free with CPLFree() or VSIFree().

10.1.2.52 char CPL_DLL* CPLStrlwr (char * *pszString*)

Convert each characters of the string to lower case.

For example, "ABcdE" will be converted to "abcde". This function is locale dependent.

Parameters

<i>pszString</i>	input string to be converted.
------------------	-------------------------------

Returns

pointer to the same string, *pszString*.

10.1.2.53 double CPL_DLL 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\(\)](#) function if you want to specify custom delimiter. Also see notes for [CPLAtof\(\)](#) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .

Returns

Converted value, if any.

10.1.2.54 double CPL_DLL 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\(\)](#) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .
<i>point</i>	Decimal delimiter.

Returns

Converted value, if any.

10.1.2.55 float CPL_DLL 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\(\)](#) function if you want to specify custom delimiter. Also see notes for [CPLAtof\(\)](#) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .

Returns

Converted value, if any.

10.1.2.56 float CPL_DLL 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\(\)](#) function.

Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .
<i>point</i>	Decimal delimiter.

Returns

Converted value, if any.

10.1.2.57 int CPL_DLL CPLUnlinkTree (const char * *pszPath*)

Returns

0 on successful completion, -1 if function fails.

10.1.2.58 void CPL_DLL* CPLZLibDeflate (const void * *ptr*, size_t *nBytes*, int *nLevel*, void * *outptr*, size_t *nOutAvailableBytes*, size_t * *pnOutBytes*)

Compress a buffer with ZLib DEFLATE compression.

Parameters

<i>ptr</i>	input buffer.
<i>nBytes</i>	size of input buffer in bytes.
<i>nLevel</i>	ZLib compression level (-1 for default).
<i>outptr</i>	output buffer, or NULL to let the function allocate it.
<i>nOutAvailable-Bytes</i>	size of output buffer if provided, or ignored.
<i>pnOutBytes</i>	pointer to a <code>size_t</code> , where to store the size of the output buffer.

Returns

the output buffer (to be freed with VSIFree() if not provided) or NULL in case of error.

Since

GDAL 1.10.0

10.1.2.59 void CPL_DLL* CPLZLibInflate (const void * *ptr*, size_t *nBytes*, void * *outptr*, size_t *nOutAvailableBytes*, size_t * *pnOutBytes*)

Uncompress a buffer compressed with ZLib DEFLATE compression.

Parameters

<i>ptr</i>	input buffer.
<i>nBytes</i>	size of input buffer in bytes.
<i>outptr</i>	output buffer, or NULL to let the function allocate it.
<i>nOutAvailableBytes</i>	size of output buffer if provided, or ignored.
<i>pnOutBytes</i>	pointer to a size_t, where to store the size of the output buffer.

Returns

the output buffer (to be freed with VSIFree() if not provided) or NULL in case of error.

Since

GDAL 1.10.0

10.2 cpl_error.h File Reference

```
#include "cpl_port.h"
```

Functions

- void CPL_DLL [CPLEmergencyError](#) (const char *)
- void CPL_DLL CPL_STDCALL [CPLErrorReset](#) (void)
- int CPL_DLL CPL_STDCALL [CPLGetLastErrorNo](#) (void)
- CPLerr CPL_DLL CPL_STDCALL [CPLGetLastErrorType](#) (void)
- const char CPL_DLL *CPL_STDCALL [CPLGetLastErrorMsg](#) (void)
- void CPL_DLL *CPL_STDCALL [CPLGetErrorHandlerUserData](#) (void)
- CPLErrorHandler CPL_DLL CPL_STDCALL [CPLSetErrorHandler](#) (CPLErrorHandler)
- CPLErrorHandler CPL_DLL CPL_STDCALL [CPLSetErrorHandlerEx](#) (CPLErrorHandler, void *)
- void CPL_DLL CPL_STDCALL [CPLPushErrorHandler](#) (CPLErrorHandler)
- void CPL_DLL CPL_STDCALL [CPLPushErrorHandlerEx](#) (CPLErrorHandler, void *)
- void CPL_DLL CPL_STDCALL [CPLPopErrorHandler](#) (void)
- void CPL_DLL CPL_STDCALL void [CPLAssert](#) (const char *, const char *, int)

10.2.1 Detailed Description

CPL error handling services.

10.2.2 Function Documentation

10.2.2.1 void CPL_DLL CPL_STDCALL void CPL_DLL CPL_STDCALL _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() only if the condition fails. _CPLAssert() will generate a CE_Fatal error call to CPLError(), 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() directly.

10.2.2.2 void CPL_DLL CPLEmergencyError (const char * *pszMessage*)

Fatal error when things are bad.

This function should be called in an emergency situation where it is unlikely that a regular error report would work. This would include in the case of heap exhaustion for even small allocations, or any failure in the process of reporting an error (such as TLS allocations).

This function should never return. After the error message has been reported as best possible, the application will abort() similarly to how CPLError() aborts on CE_Fatal class errors.

Parameters

<i>pszMessage</i>	the error message to report.
-------------------	------------------------------

10.2.2.3 void CPL_DLL CPL_STDCALL 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.

10.2.2.4 void CPL_DLL* CPL_STDCALL CPLGetErrorHandlerUserData (void)

Fetch the user data for the error context

Fetches the user data for the current error context. You can set the user data for the error context when you add your handler by issuing CPLSetErrorHandlerEx() and CPLPushErrorHandlerEx(). Note that user data is primarily intended for providing context within error handlers themselves, but they could potentially be abused in other useful ways with the usual caveat emptor understanding.

Returns

the user data pointer for the error context

10.2.2.5 const char CPL_DLL* CPL_STDCALL CPLGetLastErrorMsg (void)

Get the last error message.

Fetches the last error message posted with CPLError(), that hasn't been cleared by CPLErrorReset(). 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.

10.2.2.6 `int CPL_DLL CPL_STDCALL CPLGetLastErrorNo ( void )`

Fetch the last error number.

Fetches the last error number posted with `CPL_Error()`, that hasn't been cleared by [CPL_ErrorReset\(\)](#). This is the error number, not the error class.

Returns

the error number of the last error to occur, or `CPL_None` (0) if there are no posted errors.

10.2.2.7 `CPL_Err CPL_DLL CPL_STDCALL CPLGetLastErrorType ( void )`

Fetch the last error type.

Fetches the last error type posted with `CPL_Error()`, that hasn't been cleared by [CPL_ErrorReset\(\)](#). This is the error class, not the error number.

Returns

the error type of the last error to occur, or `CE_None` (0) if there are no posted errors.

10.2.2.8 `void CPL_DLL CPL_STDCALL 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\(\)](#) call. This method has no effect if there are no error handlers on the current threads error handler stack.

10.2.2.9 `void CPL_DLL CPL_STDCALL CPLPushErrorHandler ( CPLErrorHandler pfnErrorHandlerNew )`

Push a new `CPL_Error` handler.

This pushes a new error handler on the thread-local error handler stack. This handler will be used until removed with [CPLPopErrorHandler\(\)](#).

The [CPLSetErrorHandler\(\)](#) docs have further information on how `CPL_Error` handlers work.

Parameters

<i>pfnErrorHandlerNew</i>	new error handler function.
---------------------------	-----------------------------

10.2.2.10 `void CPL_DLL CPL_STDCALL CPLPushErrorHandlerEx ( CPLErrorHandler pfnErrorHandlerNew, void * pUserData )`

Push a new `CPL_Error` handler with user data on the error context.

This pushes a new error handler on the thread-local error handler stack. This handler will be used until removed with [CPLPopErrorHandler\(\)](#). Obtain the user data back by using [CPLGetErrorContext\(\)](#).

The [CPLSetErrorHandler\(\)](#) docs have further information on how `CPL_Error` handlers work.

Parameters

<i>pfnErrorHandlerNew</i>	new error handler function.
<i>pUserData</i>	User data to put on the error context.

10.2.2.11 CPLErrorHandler CPL_DLL CPL_STDCALL 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() before calling the handler. If the error handler function is passed a CE_Fatal class error and returns, then CPLError() 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\(\)](#) apply to all threads in an application, while error handlers set with CPLPushErrorHandler are thread-local. However, any error handlers pushed with CPLPushErrorHandler (and not removed with CPLPopErrorHandler) take precedence over the global error handlers set with [CPLSetErrorHandler\(\)](#). Generally speaking [CPLSetErrorHandler\(\)](#) would be used to set a desired global error handler, while [CPLPushErrorHandler\(\)](#) would be used to install a temporary local error handler, such as CPLQuietErrorHandler() to suppress error reporting in a limited segment of code.

Parameters

<i>pfnErrorHandler-New</i>	new error handler function.
----------------------------	-----------------------------

Returns

returns the previously installed error handler.

10.2.2.12 CPLErrorHandler CPL_DLL CPL_STDCALL CPLSetErrorHandlerEx (CPLErrorHandler *pfnErrorHandlerNew*, void * *pUserData*)

Install custom error handle with user's data. This method is essentially CPLSetErrorHandler with an added pointer to pUserData. The pUserData is not returned in the CPLErrorHandler, however, and must be fetched via CPLGetLastErrorUserData

Parameters

<i>pfnErrorHandler-New</i>	new error handler function.
<i>pUserData</i>	User data to carry along with the error context.

Returns

returns the previously installed error handler.

10.3 cpl_hash_set.h File Reference

```
#include "cpl_port.h"
```

Functions

- `CPLHashSet CPL_DLL * CPLHashSetNew` (`CPLHashSetHashFunc fnHashFunc`, `CPLHashSetEqualFunc fnEqualFunc`, `CPLHashSetFreeEltFunc fnFreeEltFunc`)
- `void CPL_DLL CPLHashSetDestroy` (`CPLHashSet *set`)
- `int CPL_DLL CPLHashSetSize` (`const CPLHashSet *set`)
- `void CPL_DLL CPLHashSetForeach` (`CPLHashSet *set`, `CPLHashSetIterEltFunc fnIterFunc`, `void *user_data`)
- `int CPL_DLL CPLHashSetInsert` (`CPLHashSet *set`, `void *elt`)
- `void CPL_DLL * CPLHashSetLookup` (`CPLHashSet *set`, `const void *elt`)
- `int CPL_DLL CPLHashSetRemove` (`CPLHashSet *set`, `const void *elt`)
- `unsigned long CPL_DLL CPLHashSetHashPointer` (`const void *elt`)
- `int CPL_DLL CPLHashSetEqualPointer` (`const void *elt1`, `const void *elt2`)
- `unsigned long CPL_DLL CPLHashSetHashStr` (`const void *pszStr`)
- `int CPL_DLL CPLHashSetEqualStr` (`const void *pszStr1`, `const void *pszStr2`)

10.3.1 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.

10.3.2 Function Documentation

10.3.2.1 `void CPL_DLL CPLHashSetDestroy ( CPLHashSet * set )`

Destroys an allocated hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

Parameters

<i>set</i>	the hash set
------------	--------------

10.3.2.2 `int CPL_DLL CPLHashSetEqualPointer ( const void * elt1, const void * elt2 )`

Equality function for arbitrary pointers

Parameters

<i>elt1</i>	the first arbitrary pointer to compare
<i>elt2</i>	the second arbitrary pointer to compare

Returns

TRUE if the pointers are equal

10.3.2.3 `int CPL_DLL CPLHashSetEqualStr ( const void * elt1, const void * elt2 )`

Equality function for strings

Parameters

<i>elt1</i>	the first string to compare. May be NULL.
<i>elt2</i>	the second string to compare. May be NULL.

Returns

TRUE if the strings are equal

10.3.2.4 void CPL_DLL CPLHashSetForeach (CPLHashSet * *set*, CPLHashSetIterEltFunc *fnIterFunc*, void * *user_data*)

Walk through the hash set and runs the provided function on all the elements

This function is provided the *user_data* argument of CPLHashSetForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the hash set must *NOT* be modified during the walk.

Parameters

<i>set</i>	the hash set.
<i>fnIterFunc</i>	the function called on each element.
<i>user_data</i>	the user data provided to the function.

10.3.2.5 unsigned long CPL_DLL CPLHashSetHashPointer (const void * *elt*)

Hash function for an arbitrary pointer

Parameters

<i>elt</i>	the arbitrary pointer to hash
------------	-------------------------------

Returns

the hash value of the pointer

10.3.2.6 unsigned long CPL_DLL CPLHashSetHashStr (const void * *elt*)

Hash function for a zero-terminated string

Parameters

<i>elt</i>	the string to hash. May be NULL.
------------	----------------------------------

Returns

the hash value of the string

10.3.2.7 int CPL_DLL CPLHashSetInsert (CPLHashSet * *set*, void * *elt*)

Inserts an element into a hash set.

If the element was already inserted in the hash set, the previous element is replaced by the new element. If a free function was provided, it is used to free the previously inserted element

Parameters

<i>set</i>	the hash set
<i>elt</i>	the new element to insert in the hash set

Returns

TRUE if the element was not already in the hash set

10.3.2.8 void CPL_DLL* CPLHashSetLookup (CPLHashSet * *set*, const void * *elt*)

Returns the element found in the hash set corresponding to the element to look up The element must not be modified.

Parameters

<i>set</i>	the hash set
<i>elt</i>	the element to look up in the hash set

Returns

the element found in the hash set or NULL

10.3.2.9 CPLHashSet CPL_DLL* CPLHashSetNew (CPLHashSetHashFunc *fnHashFunc*, CPLHashSetEqualFunc *fnEqualFunc*, CPLHashSetFreeEltFunc *fnFreeEltFunc*)

Creates a new hash set

The hash function must return a hash value for the elements to insert. If *fnHashFunc* is NULL, *CPLHashSetHashPointer* will be used.

The equal function must return if two elements are equal. If *fnEqualFunc* is NULL, *CPLHashSetEqualPointer* will be used.

The free function is used to free elements inserted in the hash set, when the hash set is destroyed, when elements are removed or replaced. If *fnFreeEltFunc* is NULL, elements inserted into the hash set will not be freed.

Parameters

<i>fnHashFunc</i>	hash function. May be NULL.
<i>fnEqualFunc</i>	equal function. May be NULL.
<i>fnFreeEltFunc</i>	element free function. May be NULL.

Returns

a new hash set

10.3.2.10 int CPL_DLL CPLHashSetRemove (CPLHashSet * *set*, const void * *elt*)

Removes an element from a hash set

Parameters

<i>set</i>	the hash set
<i>elt</i>	the new element to remove from the hash set

Returns

TRUE if the element was in the hash set

10.3.2.11 `int CPL_DLL 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

<code>set</code>	the hash set
------------------	--------------

Returns

the number of elements in the hash set

10.4 `cpl_http.h` File Reference

```
#include "cpl_conv.h"
#include "cpl_string.h"
#include "cpl_vsi.h"
```

Classes

- struct [CPLMimePart](#)
- struct [CPLHTTPResult](#)

Functions

- `int CPL_DLL CPLHTTPEnabled (void)`
Return if CPLHTTP services can be usefull.
- `CPLHTTPResult CPL_DLL * CPLHTTPFetch (const char *pszURL, char **papszOptions)`
Fetch a document from an url and return in a string.
- `void CPL_DLL CPLHTTPCleanup (void)`
Cleanup function to call at application termination.
- `void CPL_DLL CPLHTTPDestroyResult (CPLHTTPResult *psResult)`
Clean the memory associated with the return value of [CPLHTTPFetch\(\)](#)
- `int CPL_DLL CPLHTTPParseMultipartMime (CPLHTTPResult *psResult)`
Parses a a MIME multipart message.
- `char CPL_DLL * GOA2GetAuthorizationURL (const char *pszScope)`
- `char CPL_DLL * GOA2GetRefreshToken (const char *pszAuthToken, const char *pszScope)`
- `char CPL_DLL * GOA2GetAccessToken (const char *pszRefreshToken, const char *pszScope)`

10.4.1 Detailed Description

Interface for downloading HTTP, FTP documents

10.4.2 Function Documentation

10.4.2.1 void CPL_DLL CPLHTTPDestroyResult (CPLHTTPResult * *psResult*)

Clean the memory associated with the return value of [CPLHTTPFetch\(\)](#)

Parameters

<i>psResult</i>	pointer to the return value of CPLHTTPFetch()
-----------------	---

10.4.2.2 int CPL_DLL CPLHTTPEnabled (void)

Return if CPLHTTP services can be usefull.

Those services depend on GDAL being build with libcurl support.

Returns

TRUE if libcurl support is enabled

10.4.2.3 CPLHTTPResult CPL_DLL* CPLHTTPFetch (const char * *pszURL*, char ** *papszOptions*)

Fetch a document from an url and return in a string.

Parameters

<i>pszURL</i>	valid URL recognized by underlying download library (libcurl)
<i>papszOptions</i>	option list as a NULL-terminated array of strings. May be NULL. The following options are handled : • TIMEOUT=val, where val is in seconds • HEADERS=val, where val is an extra header to use when getting a web page. For example "Accept: application/x-ogcwk" • HTTPAUTH=[BASIC/NTLM/GSSNEGOTIATE/ANY] to specify an authentication scheme to use. • USERPWD=userid:password to specify a user and password for authentication • POSTFIELDS=val, where val is a nul-terminated string to be passed to the server with a POST request. • PROXY=val, to make requests go through a proxy server, where val is of the form proxy-server.com:port_number • PROXYUSERPWD=val, where val is of the form username:password • PROXYAUTH=[BASIC/NTLM/DIGEST/ANY] to specify an proxy authentication scheme to use. • CUSTOMREQUEST=val, where val is GET, PUT, POST, DELETE, etc.. (GDAL >= 1.9.0)

Alternatively, if not defined in the *papszOptions* arguments, the PROXY, PROXYUSERPWD and PROXYAUTH values are searched in the configuration options named GDAL_HTTP_PROXY, GDAL_HTTP_PROXYUSERPWD and GDAL_PROXY_AUTH, as proxy configuration belongs to networking setup and makes more sense at the configuration option level than at the connection level.

Returns

a `CPLHTTPResult*` structure that must be freed by [CPLHTTPDestroyResult\(\)](#), or NULL if libcurl support is disabled

10.4.2.4 int CPL_DLL CPLHTTPParseMultipartMime (CPLHTTPResult * *psResult*)

Parses a a MIME multipart message.

This function will iterate over each part and put it in a separate element of the `pasMimePart` array of the provided `psResult` structure.

Parameters

<i>psResult</i>	pointer to the return value of CPLHTTPFetch()
-----------------	---

Returns

TRUE if the message contains MIME multipart message.

10.4.2.5 char CPL_DLL* GOA2GetAccessToken (const char * *pszRefreshToken*, const char * *pszScope*)

Fetch access token using refresh token.

The permanent refresh token is used to fetch a temporary (usually one hour) access token using Google OAuth2 web services.

A `CPLError` will be reported if the request fails for some reason. Common reasons include the refresh token having been revoked by the user or http connection problems.

Parameters

<i>pszRefreshToken</i>	the refresh token from GOA2GetRefreshToken() .
<i>pszScope</i>	the scope for which it is valid.

Returns

access token, to be freed with `CPLFree()`, null on failure.

10.4.2.6 char CPL_DLL* GOA2GetAuthorizationURL (const char * *pszScope*)

Return authorization url for a given scope.

Returns the URL that a user should visit, and use for authentication in order to get an "auth token" indicating their willingness to use a service.

Note that when the user visits this url they will be asked to login (using a google/gmail/etc) account, and to authorize use of the requested scope for the application "GDAL/OGR". Once they have done so, they will be presented with a lengthy string they should "enter into their application". This is the "auth token" to be passed to [GOA2GetRefreshToken\(\)](#). The "auth token" can only be used once.

This function should never fail.

Parameters

<i>pszScope</i>	the service being requested, not yet URL encoded, such as "https://www.googleapis.com/auth/fusiontables".
-----------------	---

Returns

the URL to visit - should be freed with CPLFree().

10.4.2.7 char CPL_DLL* GOA2GetRefreshToken (const char * *pszAuthToken*, const char * *pszScope*)

Turn Auth Token into a Refresh Token.

A one time "auth token" provided by the user is turned into a reusable "refresh token" using a google oauth2 web service.

A CPL_Error will be reported if the translation fails for some reason. Common reasons include the auth token already having been used before, it not being appropriate for the passed scope and configured client api or http connection problems. NULL is returned on error.

Parameters

<i>pszAuthToken</i>	the authorization token from the user.
<i>pszScope</i>	the scope for which it is valid.

Returns

refresh token, to be freed with CPLFree(), null on failure.

10.5 cpl_list.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct [_CPLList](#)

Typedefs

- typedef CPL_C_START struct [_CPLList](#) [CPLList](#)

Functions

- [CPLList](#) CPL_DLL * [CPLListAppend](#) ([CPLList](#) *psList, void *pData)
- [CPLList](#) CPL_DLL * [CPLListInsert](#) ([CPLList](#) *psList, void *pData, int nPosition)
- [CPLList](#) CPL_DLL * [CPLListGetLast](#) ([CPLList](#) *psList)
- [CPLList](#) CPL_DLL * [CPLListGet](#) ([CPLList](#) *psList, int nPosition)
- int CPL_DLL [CPLListCount](#) ([CPLList](#) *psList)
- [CPLList](#) CPL_DLL * [CPLListRemove](#) ([CPLList](#) *psList, int nPosition)
- void CPL_DLL [CPLListDestroy](#) ([CPLList](#) *psList)
- [CPLList](#) CPL_DLL * [CPLListGetNext](#) ([CPLList](#) *psElement)
- void CPL_DLL * [CPLListGetData](#) ([CPLList](#) *psElement)

10.5.1 Detailed Description

Simplest list implementation. List contains only pointers to stored objects, not objects itself. All operations regarding allocation and freeing memory for objects should be performed by the caller.

10.5.2 Typedef Documentation

10.5.2.1 typedef CPL_DLL struct _CPLList CPLList

List element structure.

10.5.3 Function Documentation

10.5.3.1 CPLList CPL_DLL* CPLListAppend (CPLList * *psList*, void * *pData*)

Append an object list and return a pointer to the modified list. If the input list is NULL, then a new list is created.

Parameters

<i>psList</i>	pointer to list head.
<i>pData</i>	pointer to inserted data object. May be NULL.

Returns

pointer to the head of modified list.

10.5.3.2 int CPL_DLL CPLListCount (CPLList * *psList*)

Return the number of elements in a list.

Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

Returns

number of elements in a list.

10.5.3.3 void CPL_DLL CPLListDestroy (CPLList * *psList*)

Destroy a list. Caller responsible for freeing data objects contained in list elements.

Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

10.5.3.4 CPLList CPL_DLL* CPLListGet (CPLList * *psList*, int *nPosition*)

Return the pointer to the specified element in a list.

Parameters

<i>psList</i>	pointer to list head.
<i>nPosition</i>	the index of the element in the list, 0 being the first element

Returns

pointer to the specified element in a list.

10.5.3.5 void CPL_DLL* CPLListGetData (CPLList * *psElement*)

Return pointer to the data object contained in given list element.

Parameters

<i>psElement</i>	pointer to list element.
------------------	--------------------------

Returns

pointer to the data object contained in given list element.

10.5.3.6 CPLList CPL_DLL* CPLListGetLast (CPLList * *psList*)

Return the pointer to last element in a list.

Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

Returns

pointer to last element in a list.

10.5.3.7 CPLList CPL_DLL* CPLListGetNext (CPLList * *psElement*)

Return the pointer to next element in a list.

Parameters

<i>psElement</i>	pointer to list element.
------------------	--------------------------

Returns

pointer to the list element preceded by the given element.

10.5.3.8 CPLList CPL_DLL* CPLListInsert (CPLList * *psList*, void * *pData*, int *nPosition*)

Insert an object into list at specified position (zero based). If the input list is NULL, then a new list is created.

Parameters

<i>psList</i>	pointer to list head.
<i>pData</i>	pointer to inserted data object. May be NULL.
<i>nPosition</i>	position number to insert an object.

Returns

pointer to the head of modified list.

10.5.3.9 CPL_DLL* CPLListRemove (CPLList * *psList*, int *nPosition*)

Remove the element from the specified position (zero based) in a list. Data object contained in removed element must be freed by the caller first.

Parameters

<i>psList</i>	pointer to list head.
<i>nPosition</i>	position number to delet an element.

Returns

pointer to the head of modified list.

10.6 cpl_minixml.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct [CPLXMLNode](#)

Typedefs

- typedef struct [CPLXMLNode](#) [CPLXMLNode](#)

Enumerations

- enum [CPLXMLNodeType](#) {
[CXT_Element](#) = 0, [CXT_Text](#) = 1, [CXT_Attribute](#) = 2, [CXT_Comment](#) = 3,
[CXT_Literal](#) = 4 }

Functions

- [CPLXMLNode](#) CPL_DLL * [CPLParseXMLString](#) (const char *)
Parse an XML string into tree form.
- void CPL_DLL [CPLDestroyXMLNode](#) ([CPLXMLNode](#) *)
Destroy a tree.
- [CPLXMLNode](#) CPL_DLL * [CPLGetXMLNode](#) ([CPLXMLNode](#) *poRoot, const char *pszPath)
Find node by path.
- [CPLXMLNode](#) CPL_DLL * [CPLSearchXMLNode](#) ([CPLXMLNode](#) *poRoot, const char *pszTarget)
Search for a node in document.
- const char CPL_DLL * [CPLGetXMLValue](#) ([CPLXMLNode](#) *poRoot, const char *pszPath, const char *psz-Default)
Fetch element/attribute value.

- `CPLXMLNode` `CPL_DLL` * `CPLCreateXMLNode` (`CPLXMLNode` *`poParent`, `CPLXMLNodeType` `eType`, const char *`pszText`)
Create an document tree item.
- char `CPL_DLL` * `CPLSerializeXMLTree` (const `CPLXMLNode` *`psNode`)
Convert tree into string document.
- void `CPL_DLL` `CPLAddXMLChild` (`CPLXMLNode` *`psParent`, `CPLXMLNode` *`psChild`)
Add child node to parent.
- int `CPL_DLL` `CPLRemoveXMLChild` (`CPLXMLNode` *`psParent`, `CPLXMLNode` *`psChild`)
Remove child node from parent.
- void `CPL_DLL` `CPLAddXMLSibling` (`CPLXMLNode` *`psOlderSibling`, `CPLXMLNode` *`psNewSibling`)
Add new sibling.
- `CPLXMLNode` `CPL_DLL` * `CPLCreateXMLElementAndValue` (`CPLXMLNode` *`psParent`, const char *`pszName`, const char *`pszValue`)
Create an element and text value.
- `CPLXMLNode` `CPL_DLL` * `CPLCloneXMLTree` (`CPLXMLNode` *`psTree`)
Copy tree.
- int `CPL_DLL` `CPLSetXMLValue` (`CPLXMLNode` *`psRoot`, const char *`pszPath`, const char *`pszValue`)
Set element value by path.
- void `CPL_DLL` `CPLStripXMLNamespace` (`CPLXMLNode` *`psRoot`, const char *`pszNameSpace`, int `bRecurse`)
Strip indicated namespaces.
- void `CPL_DLL` `CPLCleanXMLElementName` (char *)
Make string into safe XML token.
- `CPLXMLNode` `CPL_DLL` * `CPLParseXMLFile` (const char *`pszFilename`)
Parse XML file into tree.
- int `CPL_DLL` `CPLSerializeXMLTreeToFile` (const `CPLXMLNode` *`psTree`, const char *`pszFilename`)
Write document tree to a file.

10.6.1 Detailed Description

Definitions for CPL mini XML Parser/Serializer.

10.6.2 Typedef Documentation

10.6.2.1 typedef struct `CPLXMLNode` `CPLXMLNode`

Document node structure.

This C structure is used to hold a single text fragment representing a component of the document when parsed. It should be allocated with the appropriate CPL function, and freed with `CPLDestroyXMLNode()`. 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` structures.

10.6.3 Enumeration Type Documentation

10.6.3.1 enum `CPLXMLNodeType`

Enumerator

`CXT_Element` Node is an element

`CXT_Text` Node is a raw text value

CXT_Attribute Node is attribute

CXT_Comment Node is an XML comment.

CXT_Literal Node is a special literal

10.6.4 Function Documentation

10.6.4.1 void CPL_DLL CPLAddXMLChild (CPLXMLNode * *psParent*, CPLXMLNode * *psChild*)

Add child node to parent.

The passed child is added to the list of children of the indicated parent. Normally the child is added at the end of the parents child list, but attributes (CXT_Attribute) will be inserted after any other attributes but before any other element type. Ownership of the child node is effectively assumed by the parent node. If the child has siblings (it's psNext is not NULL) they will be trimmed, but if the child has children they are carried with it.

Parameters

<i>psParent</i>	the node to attach the child to. May not be NULL.
<i>psChild</i>	the child to add to the parent. May not be NULL. Should not be a child of any other parent.

10.6.4.2 void CPL_DLL 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\(\)](#).

Parameters

<i>psOlderSibling</i>	the node to attach the sibling after.
<i>psNewSibling</i>	the node to add at the end of psOlderSiblings psNext chain.

10.6.4.3 void CPL_DLL CPLCleanXMLElementName (char * *pszTarget*)

Make string into safe XML token.

Modifies a string in place to try and make it into a legal XML token that can be used as an element name. This is accomplished by changing any characters not legal in a token into an underscore.

NOTE: This function should implement the rules in section 2.3 of <http://www.w3.org/TR/xml11/> but it doesn't yet do that properly. We only do a rough approximation of that.

Parameters

<i>pszTarget</i>	the string to be adjusted. It is altered in place.
------------------	--

10.6.4.4 CPLXMLNode CPL_DLL* CPLCloneXMLTree (CPLXMLNode * *psTree*)

Copy tree.

Creates a deep copy of a [CPLXMLNode](#) tree.

Parameters

<i>psTree</i>	the tree to duplicate.
---------------	------------------------

Returns

a copy of the whole tree.

10.6.4.5 **CPLXMLNode CPL_DLL* CPLCreateXMLElementAndValue (CPLXMLNode * *psParent*, const char * *pszName*, const char * *pszValue*)**

Create an element and text value.

This function is a convenient short form for:

```
CPLXMLNode *psTextNode;
CPLXMLNode *psElementNode;

psElementNode = CPLCreateXMLNode( psParent, CXT_Element, pszName );
psTextNode = CPLCreateXMLNode( psElementNode, CXT_Text, pszValue );

return psElementNode;
```

It creates a CXT_Element node, with a CXT_Text child, and attaches the element to the passed parent.

Parameters

<i>psParent</i>	the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.
<i>pszName</i>	the element name to create.
<i>pszValue</i>	the text to attach to the element. Must not be NULL.

Returns

the pointer to the new element node.

10.6.4.6 **CPLXMLNode CPL_DLL* CPLCreateXMLNode (CPLXMLNode * *poParent*, CPLXMLNodeType *eType*, const char * *pszText*)**

Create an document tree item.

Create a single [CPLXMLNode](#) object with the desired value and type, and attach it as a child of the indicated parent.

Parameters

<i>poParent</i>	the parent to which this node should be attached as a child. May be NULL to keep as free standing.
<i>eType</i>	the type of the newly created node
<i>pszText</i>	the value of the newly created node

Returns

the newly created node, now owned by the caller (or parent node).

10.6.4.7 **void CPL_DLL CPLDestroyXMLNode (CPLXMLNode * *psNode*)**

Destroy a tree.

This function frees resources associated with a [CPLXMLNode](#) and all its children nodes.

Parameters

<i>psNode</i>	the tree to free.
---------------	-------------------

10.6.4.8 CPLXMLNode CPL_DLL* CPLGetXMLNode (CPLXMLNode * *psRoot*, const char * *pszPath*)

Find node by path.

Searches the document or subdocument indicated by *psRoot* for an element (or attribute) with the given path. The path should consist of a set of element names separated by dots, not including the name of the root element (*psRoot*). If the requested element is not found NULL is returned.

Attribute names may only appear as the last item in the path.

The search is done from the root nodes children, but all intermediate nodes in the path must be specified. Searching for "name" would only find a name element or attribute if it is a direct child of the root, not at any level in the subdocument.

If the *pszPath* is prefixed by "=" then the search will begin with the root node, and it's siblings, instead of the root nodes children. This is particularly useful when searching within a whole document which is often prefixed by one or more "junk" nodes like the <?xml> declaration.

Parameters

<i>psRoot</i>	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszPath</i>	the list of element names in the path (dot separated).

Returns

the requested element node, or NULL if not found.

10.6.4.9 const char CPL_DLL* 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\(\)](#) (see there for details on path handling). Once found, the value is considered to be the first CXT_Text child of the node.

If the attribute/element search fails, or if the found node has not value then the passed default value is returned.

The returned value points to memory within the document tree, and should not be altered or freed.

Parameters

<i>psRoot</i>	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszPath</i>	the list of element names in the path (dot separated). An empty path means get the value of the <i>psRoot</i> node.
<i>pszDefault</i>	the value to return if a corresponding value is not found, may be NULL.

Returns

the requested value or *pszDefault* if not found.

10.6.4.10 CPLXMLNode CPL_DLL* 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\(\)](#). Errors in reading the file or parsing the XML will be reported by [CPL_Error\(\)](#).

The "large file" API is used, so XML files can come from virtualized files.

Parameters

<i>pszFilename</i>	the file to open.
--------------------	-------------------

Returns

NULL on failure, or the document tree on success.

10.6.4.11 **CPLXMLNode** CPL_DLL* CPLParseXMLString (const char * *pszString*)

Parse an XML string into tree form.

The passed document is parsed into a [CPLXMLNode](#) tree representation. If the document is not well formed XML then NULL is returned, and errors are reported via [CPL_Error\(\)](#). No validation beyond wellformedness is done. The [CPLParseXMLFile\(\)](#) convenience function can be used to parse from a file.

The returned document tree is owned by the caller and should be freed with [CPL_DestroyXMLNode\(\)](#) when no longer needed.

If the document has more than one "root level" element then those after the first will be attached to the first as siblings (via the *psNext* pointers) even though there is no common parent. A document with no XML structure (no angle brackets for instance) would be considered well formed, and returned as a single *CXT_Text* node.

Parameters

<i>pszString</i>	the document to parse.
------------------	------------------------

Returns

parsed tree or NULL on error.

10.6.4.12 int CPL_DLL CPLRemoveXMLChild (**CPLXMLNode** * *psParent*, **CPLXMLNode** * *psChild*)

Remove child node from parent.

The passed child is removed from the child list of the passed parent, but the child is not destroyed. The child retains ownership of it's own children, but is cleanly removed from the child list of the parent.

Parameters

<i>psParent</i>	the node to the child is attached to.
<i>psChild</i>	the child to remove.

Returns

TRUE on success or FALSE if the child was not found.

10.6.4.13 **CPLXMLNode** CPL_DLL* CPLSearchXMLNode (**CPLXMLNode** * *psRoot*, const char * *pszElement*)

Search for a node in document.

Searches the children (and potentially siblings) of the documented passed in for the named element or attribute. To search following siblings as well as children, prefix the *pszElement* name with an equal sign. This function does an in-order traversal of the document tree. So it will first match against the current node, then it's first child, that child's first child, and so on.

Use [CPLGetXMLNode\(\)](#) to find a specific child, or along a specific node path.

Parameters

<i>psRoot</i>	the subtree to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszElement</i>	the name of the element or attribute to search for.

Returns

The matching node or NULL on failure.

10.6.4.14 char CPL_DLL* CPLSerializeXMLTree (const CPLXMLNode * *psNode*)

Convert tree into string document.

This function converts a [CPLXMLNode](#) tree representation of a document into a flat string representation. White space indentation is used visually preserve the tree structure of the document. The returned document becomes owned by the caller and should be freed with [CPLFree\(\)](#) when no longer needed.

Parameters

<i>psNode</i>	the node to serialize.
---------------	------------------------

Returns

the document on success or NULL on failure.

10.6.4.15 int CPL_DLL CPLSerializeXMLTreeToFile (const CPLXMLNode * *psTree*, const char * *pszFilename*)

Write document tree to a file.

The passed document tree is converted into one big string (with [CPLSerializeXMLTree\(\)](#)) and then written to the named file. Errors writing the file will be reported by [CPLError\(\)](#). The source document tree is not altered. If the output file already exists it will be overwritten.

Parameters

<i>psTree</i>	the document tree to write.
<i>pszFilename</i>	the name of the file to write to.

Returns

TRUE on success, FALSE otherwise.

10.6.4.16 int CPL_DLL CPLSetXMLValue (CPLXMLNode * *psRoot*, const char * *pszPath*, const char * *pszValue*)

Set element value by path.

Find (or create) the target element or attribute specified in the path, and assign it the indicated value.

Any path elements that do not already exist will be created. The target nodes value (the first CXT_Text child) will be replaced with the provided value.

If the target node is an attribute instead of an element, the name should be prefixed with a #.

Example: `CPLSetXMLValue( "Citation.Id.Description", "DOQ dataset" ); CPLSetXMLValue( "Citation.Id.-Description.#name", "doq" );`

Parameters

<i>psRoot</i>	the subdocument to be updated.
<i>pszPath</i>	the dot seperated path to the target element/attribute.
<i>pszValue</i>	the text value to assign.

Returns

TRUE on success.

10.6.4.17 void CPL_DLL CPLStripXMLNamespace (CPLXMLNode * *psRoot*, const char * *pszNamespace*, int *bRecurse*)

Strip indicated namespaces.

The subdocument (*psRoot*) is recursively examined, and any elements with the indicated namespace prefix will have the namespace prefix stripped from the element names. If the passed namespace is NULL, then all namespace prefixes will be stripped.

Nodes other than elements should remain unaffected. The changes are made "in place", and should not alter any node locations, only the *pszValue* field of affected nodes.

Parameters

<i>psRoot</i>	the document to operate on.
<i>pszNamespace</i>	the name space prefix (not including colon), or NULL.
<i>bRecurse</i>	TRUE to recurse over whole document, or FALSE to only operate on the passed node.

10.7 cpl_odbc.h File Reference

```
#include "cpl_port.h"
#include <sql.h>
#include <sqlext.h>
#include <odbcinst.h>
#include "cpl_string.h"
```

Classes

- class [CPLODBCDriverInstaller](#)
- class [CPLODBCSession](#)
- class [CPLODBCStatement](#)

10.7.1 Detailed Description

ODBC Abstraction Layer (C++).

10.8 cpl_port.h File Reference

```
#include "cpl_config.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <time.h>
#include <errno.h>
#include <locale.h>
```

Macros

- **#define** [CPL_LSBINT16PTR\(x\)](#) `((*(GByte*)(x)) | (*(GByte*)((x)+1)) << 8)`
- **#define** [CPL_LSBINT32PTR\(x\)](#)
- **#define** [CPL_LSBSINT16PTR\(x\)](#) `((GInt16) CPL_LSBINT16PTR(x))`
- **#define** [CPL_LSBUINT16PTR\(x\)](#) `((GUInt16)CPL_LSBINT16PTR(x))`
- **#define** [CPL_LSBSINT32PTR\(x\)](#) `((GInt32) CPL_LSBINT32PTR(x))`
- **#define** [CPL_LSBUINT32PTR\(x\)](#) `((GUInt32)CPL_LSBINT32PTR(x))`

10.8.1 Detailed Description

Core portability definitions for CPL.

10.8.2 Macro Definition Documentation

10.8.2.1 **#define** [CPL_LSBINT16PTR\(x \)](#) `((*(GByte*)(x)) | (*(GByte*)((x)+1)) << 8)`

Return a Int16 from the 2 bytes ordered in LSB order at address x

10.8.2.2 **#define** [CPL_LSBINT32PTR\(x \)](#)

Value:

```
((*(GByte*)(x)) | (*(GByte*)((x)+1)) << 8) | \
((*(GByte*)((x)+2)) << 16) | (*(GByte*)((x)+3)) << 24)
```

Return a Int32 from the 4 bytes ordered in LSB order at address x

10.8.2.3 **#define** [CPL_LSBSINT16PTR\(x \)](#) `((GInt16) CPL_LSBINT16PTR(x))`

Return a signed Int16 from the 2 bytes ordered in LSB order at address x

10.8.2.4 **#define** [CPL_LSBSINT32PTR\(x \)](#) `((GInt32) CPL_LSBINT32PTR(x))`

Return a signed Int32 from the 4 bytes ordered in LSB order at address x

10.8.2.5 `#define CPL_LSBUINT16PTR( x ) ((GUInt16)CPL_LSBINT16PTR(x))`

Return a unsigned Int16 from the 2 bytes ordered in LSB order at address x

10.8.2.6 `#define CPL_LSBUINT32PTR( x ) ((GUInt32)CPL_LSBINT32PTR(x))`

Return a unsigned Int32 from the 4 bytes ordered in LSB order at address x

10.9 cpl_quad_tree.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct [CPLRectObj](#)

Functions

- [CPLQuadTree](#) CPL_DLL * [CPLQuadTreeCreate](#) (const [CPLRectObj](#) *pGlobalBounds, [CPLQuadTreeGetBoundsFunc](#) pfnGetBounds)
- void CPL_DLL [CPLQuadTreeDestroy](#) ([CPLQuadTree](#) *hQuadtree)
- void CPL_DLL [CPLQuadTreeSetBucketCapacity](#) ([CPLQuadTree](#) *hQuadtree, int nBucketCapacity)
- int CPL_DLL [CPLQuadTreeGetAdvisedMaxDepth](#) (int nExpectedFeatures)
- void CPL_DLL [CPLQuadTreeSetMaxDepth](#) ([CPLQuadTree](#) *hQuadtree, int nMaxDepth)
- void CPL_DLL [CPLQuadTreeInsert](#) ([CPLQuadTree](#) *hQuadtree, void *hFeature)
- void CPL_DLL ** [CPLQuadTreeSearch](#) (const [CPLQuadTree](#) *hQuadtree, const [CPLRectObj](#) *pAoi, int *pnFeatureCount)
- void CPL_DLL [CPLQuadTreeForeach](#) (const [CPLQuadTree](#) *hQuadtree, [CPLQuadTreeForeachFunc](#) pfnForeach, void *pUserData)

10.9.1 Detailed Description

Quad tree implementation.

A quadtree is a tree data structure in which each internal node has up to four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions

10.9.2 Function Documentation

10.9.2.1 **`CPLQuadTree`** CPL_DLL* [CPLQuadTreeCreate](#) (const [CPLRectObj](#) * *pGlobalBounds*, [CPLQuadTreeGetBoundsFunc](#) *pfnGetBounds*)

Create a new quadtree

Parameters

<i>pGlobalBounds</i>	a pointer to the global extent of all the elements that will be inserted
<i>pfnGetBounds</i>	a user provided function to get the bounding box of the inserted elements

Returns

a newly allocated quadtree

10.9.2.2 void CPL_DLL CPLQuadTreeDestroy (CPLQuadTree * *hQuadTree*)

Destroy a quadtree

Parameters

<i>hQuadTree</i>	the quad tree to destroy
------------------	--------------------------

10.9.2.3 void CPL_DLL CPLQuadTreeForeach (const CPLQuadTree * *hQuadTree*, CPLQuadTreeForeachFunc *pfnForeach*, void * *pUserData*)

Walk through the quadtree and runs the provided function on all the elements

This function is provided with the user_data argument of pfnForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the quadtree must *NOT* be modified during the walk.

Parameters

<i>hQuadTree</i>	the quad tree
<i>pfnForeach</i>	the function called on each element.
<i>pUserData</i>	the user data provided to the function.

10.9.2.4 int CPL_DLL CPLQuadTreeGetAdvisedMaxDepth (int *nExpectedFeatures*)

Returns the optimal depth of a quadtree to hold nExpectedFeatures

Parameters

<i>nExpectedFeatures</i>	the expected maximum number of elements to be inserted
--------------------------	--

Returns

the optimal depth of a quadtree to hold nExpectedFeatures

10.9.2.5 void CPL_DLL CPLQuadTreeInsert (CPLQuadTree * *hQuadTree*, void * *hFeature*)

Insert a feature into a quadtree

Parameters

<i>hQuadTree</i>	the quad tree
<i>hFeature</i>	the feature to insert

10.9.2.6 void CPL_DLL CPLQuadTreeSearch (const CPLQuadTree * *hQuadTree*, const CPLRectObj * *pAoi*, int * *pnFeatureCount*)**

Returns all the elements inserted whose bounding box intersects the provided area of interest

Parameters

<i>hQuadTree</i>	the quad tree
<i>pAoi</i>	the pointer to the area of interest
<i>pnFeatureCount</i>	the user data provided to the function.

Returns

an array of features that must be freed with CPLFree

10.9.2.7 void CPL_DLL CPLQuadTreeSetBucketCapacity (CPLQuadTree * hQuadTree, int nBucketCapacity)

Set the maximum capacity of a node of a quadtree. The default value is 8. Note that the maximum capacity will only be honoured if the features inserted have a point geometry. Otherwise it may be exceeded.

Parameters

<i>hQuadTree</i>	the quad tree
<i>nBucketCapacity</i>	the maximum capacity of a node of a quadtree

10.9.2.8 void CPL_DLL CPLQuadTreeSetMaxDepth (CPLQuadTree * hQuadTree, int nMaxDepth)

Set the maximum depth of a quadtree. By default, quad trees have no maximum depth, but a maximum bucket capacity.

Parameters

<i>hQuadTree</i>	the quad tree
<i>nMaxDepth</i>	the maximum depth allowed

10.10 cpl_string.h File Reference

```
#include "cpl_vsi.h"
#include "cpl_error.h"
#include "cpl_conv.h"
```

Functions

- int CPL_DLL [CSLCount](#) (char **papszStrList)
- void CPL_DLL CPL_STDCALL [CSLDestroy](#) (char **papszStrList)
- char CPL_DLL ** [CSLDuplicate](#) (char **papszStrList) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** [CSLMerge](#) (char **papszOrig, char **papszOverride) CPL_WARN_UNUSED_RESULT
- *Merge two lists.*
- char CPL_DLL ** [CSLTokenizeString2](#) (const char *pszString, const char *pszDelimiter, int nCSLTFlags) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** [CSLLoad](#) (const char *pszFname) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** [CSLLoad2](#) (const char *pszFname, int nMaxLines, int nMaxCols, char **papszOptions) CPL_WARN_UNUSED_RESULT
- int CPL_DLL [CSLFindString](#) (char **, const char *)
- int CPL_DLL [CSLPartialFindString](#) (char **papszHaystack, const char *pszNeedle)
- int CPL_DLL [CSLFindName](#) (char **papszStrList, const char *pszName)
- int CPL_DLL [CSLTestBoolean](#) (const char *pszValue)

- const char CPL_DLL * [CPLParseNameValue](#) (const char *pszNameValue, char **ppszKey)
- char CPL_DLL ** [CSLSetNameValue](#) (char **papszStrList, const char *pszName, const char *pszValue) CPL_WARN_UNUSED_RESULT
- void CPL_DLL [CSLSetNameValueSeparator](#) (char **papszStrList, const char *pszSeparator)
- char CPL_DLL * [CPLEscapeString](#) (const char *pszString, int nLength, int nScheme) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * [CPLUnescapeString](#) (const char *pszString, int *pnLength, int nScheme) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * [CPLBinaryToHex](#) (int nBytes, const GByte *pabyData) CPL_WARN_UNUSED_RESULT
- GByte CPL_DLL * [CPLHexToBinary](#) (const char *pszHex, int *pnBytes) CPL_WARN_UNUSED_RESULT
- CPLValueType CPL_DLL [CPLGetValueType](#) (const char *pszValue)
- size_t CPL_DLL [CPLStrncpy](#) (char *pszDest, const char *pszSrc, size_t nDestSize)
- size_t CPL_DLL [CPLStrcat](#) (char *pszDest, const char *pszSrc, size_t nDestSize)
- size_t CPL_DLL [CPLStrnlen](#) (const char *pszStr, size_t nMaxLen)
- int CPL_DLL [CPLEncodingCharSize](#) (const char *pszEncoding)
- char CPL_DLL * [CPLRecode](#) (const char *pszSource, const char *pszSrcEncoding, const char *pszDstEncoding) CPL_WARN_UNUSED_RESULT
- char CPL_DLL * [CPLRecodeFromWChar](#) (const wchar_t *pwszSource, const char *pszSrcEncoding, const char *pszDstEncoding) CPL_WARN_UNUSED_RESULT
- wchar_t CPL_DLL * [CPLRecodeToWChar](#) (const char *pszSource, const char *pszSrcEncoding, const char *pszDstEncoding) CPL_WARN_UNUSED_RESULT
- int CPL_DLL [CPLIsUTF8](#) (const char *pabyData, int nLen)
- char CPL_DLL * [CPLForceToASCII](#) (const char *pabyData, int nLen, char chReplacementChar) CPL_WARN_UNUSED_RESULT

10.10.1 Detailed Description

Various convenience functions for working with strings and string lists.

A StringList is just an array of strings with the last pointer being NULL. An empty StringList may be either a NULL pointer, or a pointer to a pointer memory location with a NULL value.

A common convention for StringLists is to use them to store name/value lists. In this case the contents are treated like a dictionary of name/value pairs. The actual data is formatted with each string having the format "<name>:<value>" (though "=" is also an acceptable separator). A number of the functions in the file operate on name/value style string lists (such as [CSLSetNameValue\(\)](#), and [CSLFetchNameValue\(\)](#)).

To some extent the CPLStringList C++ class can be used to abstract managing string lists a bit but still be able to return them from C functions.

10.10.2 Function Documentation

10.10.2.1 char CPL_DLL* CPLBinaryToHex (int nBytes, const GByte * pabyData)

Binary to hexadecimal translation.

Parameters

<i>nBytes</i>	number of bytes of binary data in pabyData.
<i>pabyData</i>	array of data bytes to translate.

Returns

hexadecimal translation, zero terminated. Free with [CPLFree\(\)](#).

10.10.2.2 int CPL_DLL CPLEncodingCharSize (const char * *pszEncoding*)

Return bytes per character for encoding.

This function returns the size in bytes of the smallest character in this encoding. For fixed width encodings (ASCII, UCS-2, UCS-4) this is straight forward. For encodings like UTF8 and UTF16 which represent some characters as a sequence of atomic character sizes the function still returns the atomic character size (1 for UTF8, 2 for UTF16).

This function will return the correct value for well known encodings with corresponding CPL_ENC_ values. It may not return the correct value for other encodings even if they are supported by the underlying iconv or windows transliteration services. Hopefully it will improve over time.

Parameters

<i>pszEncoding</i>	the name of the encoding.
--------------------	---------------------------

Returns

the size of a minimal character in bytes or -1 if the size is unknown.

10.10.2.3 char CPL_DLL* CPLEscapeString (const char * *pszInput*, int *nLength*, int *nScheme*)

Apply escaping to string to preserve special characters.

This function will "escape" a variety of special characters to make the string suitable to embed within a string constant or to write within a text stream but in a form that can be reconstituted to it's original form. The escaping will even preserve zero bytes allowing preservation of raw binary data.

CPLES_BackslashQuotable(0): This scheme turns a binary string into a form suitable to be placed within double quotes as a string constant. The backslash, quote, '\0' and newline characters are all escaped in the usual C style.

CPLES_XML(1): This scheme converts the '<', '>', '"' and '&' characters into their XML/HTML equivalent (<, >, " and &) making a string safe to embed as CDATA within an XML element. The '\0' is not escaped and should not be included in the input.

CPLES_URL(2): Everything except alphanumerics and the underscore are converted to a percent followed by a two digit hex encoding of the character (leading zero supplied if needed). This is the mechanism used for encoding values to be passed in URLs.

CPLES_SQL(3): All single quotes are replaced with two single quotes. Suitable for use when constructing literal values for SQL commands where the literal will be enclosed in single quotes.

CPLES_CSV(4): If the values contains commas, semicolons, tabs, double quotes, or newlines it placed in double quotes, and double quotes in the value are doubled. Suitable for use when constructing field values for .csv files. Note that [CPLUnescapeString\(\)](#) currently does not support this format, only [CPLEscapeString\(\)](#). See `cpl_csv.cpp` for csv parsing support.

Parameters

<i>pszInput</i>	the string to escape.
<i>nLength</i>	The number of bytes of data to preserve. If this is -1 the <code>strlen(pszString)</code> function will be used to compute the length.
<i>nScheme</i>	the encoding scheme to use.

Returns

an escaped, zero terminated string that should be freed with CPLFree() when no longer needed.

10.10.2.4 char CPL_DLL* CPLForceToASCII (const char * *pabyData*, int *nLen*, char *chReplacementChar*)

Return a new string that is made only of ASCII characters. If non-ASCII characters are found in the input string, they will be replaced by the provided replacement character.

Parameters

<i>pabyData</i>	input string to test
<i>nLen</i>	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.
<i>chReplacementChar</i>	character which will be used when the input stream contains a non ASCII character. Must be valid ASCII !

Returns

a new string that must be freed with CPLFree().

Since

GDAL 1.7.0

10.10.2.5 CPLValueType CPL_DLL CPLGetValueType (const char * *pszValue*)

Detect the type of the value contained in a string, whether it is a real, an integer or a string. Leading and trailing spaces are skipped in the analysis.

Note: in the context of this function, integer must be understood in a broad sense. It does not mean that the value can fit into a 32 bit integer for example. It might be larger.

Parameters

<i>pszValue</i>	the string to analyze
-----------------	-----------------------

Returns

returns the type of the value contained in the string.

10.10.2.6 GByte CPL_DLL* CPLHexToBinary (const char * *pszHex*, int * *pnBytes*)

Hexadecimal to binary translation

Parameters

<i>pszHex</i>	the input hex encoded string.
<i>pnBytes</i>	the returned count of decoded bytes placed here.

Returns

returns binary buffer of data - free with CPLFree().

10.10.2.7 int CPL_DLL CPLIsUTF8 (const char * *pabyData*, int *nLen*)

Test if a string is encoded as UTF-8.

Parameters

<i>pabyData</i>	input string to test
<i>nLen</i>	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.

Returns

TRUE if the string is encoded as UTF-8. FALSE otherwise

Since

GDAL 1.7.0

10.10.2.8 const char CPL_DLL* CPLParseNameValue (const char * *pszNameValue*, char ** *ppszKey*)

Parse NAME=VALUE string into name and value components.

Note that if *ppszKey* is non-NULL, the key (or name) portion will be allocated using VSIMalloc(), and returned in that pointer. It is the applications responsibility to free this string, but the application should not modify or free the returned value portion.

This function also support "NAME:VALUE" strings and will strip white space from around the delimiter when forming name and value strings.

Eventually CSLFetchNameValue() and friends may be modified to use [CPLParseNameValue\(\)](#).

Parameters

<i>pszNameValue</i>	string in "NAME=VALUE" format.
<i>ppszKey</i>	optional pointer though which to return the name portion.

Returns

the value portion (pointing into original string).

10.10.2.9 char CPL_DLL* CPLRecode (const char * *pszSource*, const char * *pszSrcEncoding*, const char * *pszDstEncoding*)

Convert a string from a source encoding to a destination encoding.

The only guaranteed supported encodings are CPL_ENC_UTF8, CPL_ENC_ASCII and CPL_ENC_ISO8859_1. Currently, the following conversions are supported :

- CPL_ENC_ASCII -> CPL_ENC_UTF8 or CPL_ENC_ISO8859_1 (no conversion in fact)
- CPL_ENC_ISO8859_1 -> CPL_ENC_UTF8
- CPL_ENC_UTF8 -> CPL_ENC_ISO8859_1

If an error occurs an error may, or may not be posted with CPLError().

Parameters

<i>pszSource</i>	a NULL terminated string.
<i>pszSrcEncoding</i>	the source encoding.
<i>pszDstEncoding</i>	the destination encoding.

Returns

a NULL terminated string which should be freed with CPLFree().

Since

GDAL 1.6.0

10.10.2.10 `char CPL_DLL* CPLRecodeFromWChar ( const wchar_t * pszSource, 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().

Parameters

<i>pszSource</i>	the source wchar_t string, terminated with a 0 wchar_t.
<i>pszSrcEncoding</i>	the source encoding, typically CPL_ENC_UCS2.
<i>pszDstEncoding</i>	the destination encoding, typically CPL_ENC_UTF8.

Returns

a zero terminated multi-byte string which should be freed with CPLFree(), or NULL if an error occurs.

Since

GDAL 1.6.0

10.10.2.11 `wchar_t CPL_DLL* 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().

Parameters

<i>pszSource</i>	input multi-byte character string.
<i>pszSrcEncoding</i>	source encoding, typically CPL_ENC_UTF8.
<i>pszDstEncoding</i>	destination encoding, typically CPL_ENC_UCS2.

Returns

the zero terminated wchar_t string (to be freed with CPLFree()) or NULL on error.

Since

GDAL 1.6.0

10.10.2.12 `size_t CPL_DLL CPLStrcat ( char * pszDest, const char * pszSrc, size_t nDestSize )`

Appends a source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1 and that there is at least one byte free in pszDest, that is to say `strlen(pszDest_before) < nDestSize`)

This function is designed to be a safer, more consistent, and less error prone replacement for `strncat`. Its contract is identical to `libbsd's strlcat`.

Truncation can be detected by testing if the return value of `CPLStrcat` is greater or equal to `nDestSize`.

```
char szDest[5];
CPLStrncpy(szDest, "ab", sizeof(szDest));
if (CPLStrcat(szDest, "cde", sizeof(szDest)) >= sizeof(szDest))
    fprintf(stderr, "truncation occurred !\n");
```

Parameters

<i>pszDest</i>	destination buffer. Must be NUL terminated before running <code>CPLStrcat</code>
<i>pszSrc</i>	source string. Must be NUL terminated
<i>nDestSize</i>	size of destination buffer (including space for the NUL terminator character)

Returns

the theoretical length of the destination string after concatenation (`=strlen(pszDest_before) + strlen(pszSrc)`). If `strlen(pszDest_before) >= nDestSize`, then it returns `nDestSize + strlen(pszSrc)`

Since

GDAL 1.7.0

10.10.2.13 `size_t CPL_DLL CPLStrncpy ( char * pszDest, const char * pszSrc, size_t nDestSize )`

Copy source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1).

This function is designed to be a safer, more consistent, and less error prone replacement for `strncpy`. Its contract is identical to `libbsd's strlcpy`.

Truncation can be detected by testing if the return value of `CPLStrncpy` is greater or equal to `nDestSize`.

```
char szDest[5];
if (CPLStrncpy(szDest, "abcde", sizeof(szDest)) >= sizeof(szDest))
    fprintf(stderr, "truncation occurred !\n");
```

Parameters

<i>pszDest</i>	destination buffer
<i>pszSrc</i>	source string. Must be NUL terminated
<i>nDestSize</i>	size of destination buffer (including space for the NUL terminator character)

Returns

the length of the source string (=strlen(pszSrc))

Since

GDAL 1.7.0

10.10.2.14 `size_t CPL_DLL CPLStrlen ( const char * pszStr, size_t nMaxLen )`

Returns the length of a NUL terminated string by reading at most the specified number of bytes.

The [CPLStrlen\(\)](#) function returns MIN(strlen(pszStr), nMaxLen). Only the first nMaxLen bytes of the string will be read. Usefull to test if a string contains at least nMaxLen characters without reading the full string up to the NUL terminating character.

Parameters

<i>pszStr</i>	a NUL terminated string
<i>nMaxLen</i>	maximum number of bytes to read in pszStr

Returns

strlen(pszStr) if the length is lesser than nMaxLen, otherwise nMaxLen if the NUL character has not been found in the first nMaxLen bytes.

Since

GDAL 1.7.0

10.10.2.15 `char CPL_DLL* CPLUnescapeString ( const char * pszInput, int * pnLength, int nScheme )`

Unescape a string.

This function does the opposite of [CPLEscapeString\(\)](#). Given a string with special values escaped according to some scheme, it will return a new copy of the string returned to it's original form.

Parameters

<i>pszInput</i>	the input string. This is a zero terminated string.
<i>pnLength</i>	location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.
<i>nScheme</i>	the escaped scheme to undo (see CPLEscapeString() for a list).

Returns

a copy of the unescaped string that should be freed by the application using CPLFree() when no longer needed.

10.10.2.16 `int CPL_DLL 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\(\)](#) as an end condition for loops as it will result in $O(n^2)$ behavior.

Parameters

<i>papszStrList</i>	the string list to count.
---------------------	---------------------------

Returns

the number of entries.

10.10.2.17 void CPL_DLL CPL_STDCALL CSLDestroy (char ** *papszStrList*)

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

Parameters

<i>papszStrList</i>	the list to free.
---------------------	-------------------

10.10.2.18 char CPL_DLL** 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 [CSL-Destroy\(\)](#).

Parameters

<i>papszStrList</i>	the input string list.
---------------------	------------------------

Returns

newly allocated copy.

10.10.2.19 int CPL_DLL CSLFindName (char ** *papszStrList*, const char * *pszName*)

Find StringList entry with given key name.

Parameters

<i>papszStrList</i>	the string list to search.
<i>pszName</i>	the key value to look for (case insensitive).

Returns

-1 on failure or the list index of the first occurrence matching the given key.

10.10.2.20 int CPL_DLL CSLFindString (char ** *papszList*, const char * *pszTarget*)

Find a string within a string list.

Returns the index of the entry in the string list that contains the target string. The string in the string list must be a full match for the target, but the search is case insensitive.

Parameters

<i>papszList</i>	the string list to be searched.
<i>pszTarget</i>	the string to be searched for.

Returns

the index of the string within the list or -1 on failure.

10.10.2.21 char CPL_DLL** CSLLoad (const char * *pszFname*)

Load a text file into a string list.

The VSI*L API is used, so [VSIFOpenL\(\)](#) 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\(\)](#)).

If reading the file fails a [CPL_Error\(\)](#) will be issued and NULL returned.

Parameters

<i>pszFname</i>	the name of the file to read.
-----------------	-------------------------------

Returns

a string list with the files lines, now owned by caller. To be freed with [CSL_Destroy\(\)](#)

10.10.2.22 char CPL_DLL** CSLLoad2 (const char * *pszFname*, int *nMaxLines*, int *nMaxCols*, char ** *papszOptions*)

Load a text file into a string list.

The VSI*L API is used, so [VSIFOpenL\(\)](#) 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\(\)](#)).

If reading the file fails a [CPL_Error\(\)](#) will be issued and NULL returned.

Parameters

<i>pszFname</i>	the name of the file to read.
<i>nMaxLines</i>	maximum number of lines to read before stopping, or -1 for no limit.
<i>nMaxCols</i>	maximum number of characters in a line before stopping, or -1 for no limit.
<i>papszOptions</i>	NULL-terminated array of options. Unused for now.

Returns

a string list with the files lines, now owned by caller. To be freed with [CSL_Destroy\(\)](#)

Since

GDAL 1.7.0

10.10.2.23 char CPL_DLL** CSLMerge (char ** *papszOrig*, char ** *papszOverride*)

Merge two lists.

The two lists are merged, ensuring that if any keys appear in both that the value from the second (*papszOverride*) list take precedence.

Parameters

<i>papszOrig</i>	the original list, being modified.
<i>papszOverride</i>	the list of items being merged in. This list is unaltered and remains owned by the caller.

Returns

updated list.

10.10.2.24 `int CPL_DLL 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\(\)](#)).

Parameters

<i>papszHaystack</i>	the string list to be searched.
<i>pszNeedle</i>	the substring to be searched for.

Returns

the index of the string within the list or -1 on failure.

10.10.2.25 `char CPL_DLL** CSLSetNameValue ( char ** papszList, const char * pszName, const char * pszValue )`

Assign value to name in StringList.

Set the value for a given name in a StringList of "Name=Value" pairs ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

If there is already a value for that name in the list then the value is changed, otherwise a new "Name=Value" pair is added.

Parameters

<i>papszList</i>	the original list, the modified version is returned.
<i>pszName</i>	the name to be assigned a value. This should be a well formed token (no spaces or very special characters).
<i>pszValue</i>	the value to assign to the name. This should not contain any newlines (CR or LF) but is otherwise pretty much unconstrained. If NULL any corresponding value will be removed.

Returns

modified stringlist.

10.10.2.26 `void CPL_DLL 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\(\)](#) function is used to break the existing lines, and it also strips white space from around the existing delimiter, thus the old separator, and any white space will be replaced by the new separator. For formatting purposes it may be desirable to include some white space in the new separator. eg. ": " or "= ".

Parameters

<i>papszList</i>	the list to update. Component strings may be freed but the list array will remain at the same location.
<i>pszSeparator</i>	the new separator string to insert.

10.10.2.27 int CPL_DLL CSLTestBoolean (const char * *pszValue*)

Test what boolean value contained in the string.

If *pszValue* is "NO", "FALSE", "OFF" or "0" will be returned FALSE. Otherwise, TRUE will be returned.

Parameters

<i>pszValue</i>	the string should be tested.
-----------------	------------------------------

Returns

TRUE or FALSE.

10.10.2.28 char CPL_DLL** CSLTokenizeString2 (const char * *pszString*, const char * *pszDelimiters*, int *nCSLTFlags*)

Tokenize a string.

This function will split a string into tokens based on specified' delimiter(s) with a variety of options. The returned result is a string list that should be freed with [CSLDestroy\(\)](#) when no longer needed.

The available parsing options are:

- CSLT_ALLOWEMPTYTOKENS: allow the return of empty tokens when two delimiters in a row occur with no other text between them. If not set, empty tokens will be discarded;
- CSLT_STRIPLEADSPACES: strip leading space characters from the token (as reported by isspace());
- CSLT_STRIPENDSPACES: strip ending space characters from the token (as reported by isspace());
- CSLT_HONOURSTRINGS: double quotes can be used to hold values that should not be broken into multiple tokens;
- CSLT_PRESERVEQUOTES: string quotes are carried into the tokens when this is set, otherwise they are removed;
- CSLT_PRESERVEESCAPES: if set backslash escapes (for backslash itself, and for literal double quotes) will be preserved in the tokens, otherwise the backslashes will be removed in processing.

Example:

Parse a string into tokens based on various white space (space, newline, tab) and then print out results and cleanup. Quotes may be used to hold white space in tokens.

```
char **papszTokens;
int i;

papszTokens =
    CSLTokenizeString2( pszCommand, " \t\n",
                       CSLT_HONOURSTRINGS | CSLT_ALLOWEMPTYTOKENS );

for( i = 0; papszTokens != NULL && papszTokens[i] != NULL; i++ )
    printf( "arg %d: '%s'", papszTokens[i] );
CSLDestroy( papszTokens );
```

Parameters

<i>pszString</i>	the string to be split into tokens.
<i>pszDelimiters</i>	one or more characters to be used as token delimiters.
<i>nCSLTFlags</i>	an ORing of one or more of the CSLT_ flag values.

Returns

a string list of tokens owned by the caller.

10.11 cpl_vsi.h File Reference

```
#include "cpl_port.h"
#include <unistd.h>
#include <sys/stat.h>
```

Functions

- VSILFILE CPL_DLL * [VSIFOpenL](#) (const char *, const char *) CPL_WARN_UNUSED_RESULT
Open file.
- int CPL_DLL [VSIFCloseL](#) (VSILFILE *)
Close file.
- int CPL_DLL [VSIFSeekL](#) (VSILFILE *, vsi_l_offset, int)
Seek to requested offset.
- vsi_l_offset CPL_DLL [VSIFTellL](#) (VSILFILE *)
Tell current file offset.
- size_t CPL_DLL [VSIFReadL](#) (void *, size_t, size_t, VSILFILE *)
Read bytes from file.
- int CPL_DLL [VSIFReadMultiRangeL](#) (int nRanges, void **ppData, const vsi_l_offset *panOffsets, const size_t *panSizes, VSILFILE *)
Read several ranges of bytes from file.
- size_t CPL_DLL [VSIFWriteL](#) (const void *, size_t, size_t, VSILFILE *)
Write bytes to file.
- int CPL_DLL [VSIFEofL](#) (VSILFILE *)
Test for end of file.
- int CPL_DLL [VSIFTruncateL](#) (VSILFILE *, vsi_l_offset)
Truncate/expand the file to the specified size.
- int CPL_DLL [VSIFFlushL](#) (VSILFILE *)
Flush pending writes to disk.
- int CPL_DLL [VSISatL](#) (const char *, VSISatBufL *)
Get filesystem object info.
- int CPL_DLL [VSISatExL](#) (const char *pszFilename, VSISatBufL *psStatBuf, int nFlags)
Get filesystem object info.
- int CPL_DLL [VSIIsCaseSensitiveFS](#) (const char *pszFilename)
Returns if the filenames of the filesystem are case sensitive.
- void CPL_DLL * [VSIMalloc2](#) (size_t nSize1, size_t nSize2) CPL_WARN_UNUSED_RESULT
- void CPL_DLL * [VSIMalloc3](#) (size_t nSize1, size_t nSize2, size_t nSize3) CPL_WARN_UNUSED_RESULT
- char CPL_DLL ** [VSIReadDir](#) (const char *)
Read names in a directory.
- char CPL_DLL ** [VSIReadDirRecursive](#) (const char *pszPath)
Read names in a directory recursively.

- int CPL_DLL [VSIMkdir](#) (const char *pathname, long mode)
Create a directory.
- int CPL_DLL [VSIRmdir](#) (const char *pathname)
Delete a directory.
- int CPL_DLL [VSIUnlink](#) (const char *pathname)
Delete a file.
- int CPL_DLL [VSIRename](#) (const char *oldpath, const char *newpath)
Rename a file.
- void CPL_DLL [VSIInstallMemFileHandler](#) (void)
Install "memory" file system handler.
- void CPL_DLL [VSIInstallSubFileHandler](#) (void)
- void [VSIInstallGZipFileHandler](#) (void)
Install GZip file system handler.
- void [VSIInstallZipFileHandler](#) (void)
Install ZIP file system handler.
- void [VSIInstallStdinHandler](#) (void)
Install /vsistdin/ file system handler.
- void [VSIInstallStdoutHandler](#) (void)
Install /vsistdout/ file system handler.
- void CPL_DLL [VSIInstallSparseFileHandler](#) (void)
- void [VSIInstallTarFileHandler](#) (void)
Install /vsitar/ file system handler.
- VSILFILE CPL_DLL * [VSIFileFromMemBuffer](#) (const char *pszFilename, GByte *pabyData, vsi_I_offset n-DataLength, int bTakeOwnership)
Create memory "file" from a buffer.
- GByte CPL_DLL * [VSIGetMemFileBuffer](#) (const char *pszFilename, vsi_I_offset *pnDataLength, int bUnlink-AndSeize)
Fetch buffer underlying memory file.

10.11.1 Detailed Description

Standard C Covers

The VSI functions are intended to be hookable aliases for Standard C I/O, memory allocation and other system functions. They are intended to allow virtualization of disk I/O so that non file data sources can be made to appear as files, and so that additional error trapping and reporting can be interested. The memory access API is aliased so that special application memory management services can be used.

Is is intended that each of these functions retains exactly the same calling pattern as the original Standard C functions they relate to. This means we don't have to provide custom documentation, and also means that the default implementation is very simple.

10.11.2 Function Documentation

10.11.2.1 int CPL_DLL VSIFcloseL (VSILFILE * fp)

Close file.

This function closes the indicated file.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fclose() function.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() .
-----------	---

Returns

0 on success or -1 on failure.

10.11.2.2 int CPL_DLL VSIFeofL (VSILFILE * *fp*)

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful [VSIFSeekL\(\)](#) call.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() .
-----------	---

Returns

TRUE if at EOF else FALSE.

10.11.2.3 int CPL_DLL VSIFflushL (VSILFILE * *fp*)

Flush pending writes to disk.

For files in write or update mode and on filesystem types where it is applicable, all pending output on the file is flushed to the physical disk.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() .
-----------	---

Returns

0 on success or -1 on error.

10.11.2.4 VSILFILE CPL_DLL* VSIFFileFromMemBuffer (const char * *pszFilename*, GByte * *pabyData*, vsi_l_offset *nDataLength*, int *bTakeOwnership*)

Create memory "file" from a buffer.

A virtual memory file is created from the passed buffer with the indicated filename. Under normal conditions the filename would need to be absolute and within the /vsimem/ portion of the filesystem.

If *bTakeOwnership* is TRUE, then the memory file system handler will take ownership of the buffer, freeing it when the file is deleted. Otherwise it remains the responsibility of the caller, but should not be freed as long as it might be accessed as a file. In no circumstances does this function take a copy of the *pabyData* contents.

Parameters

<i>pszFilename</i>	the filename to be created.
<i>pabyData</i>	the data buffer for the file.
<i>nDataLength</i>	the length of buffer in bytes.
<i>bTakeOwnership</i>	TRUE to transfer "ownership" of buffer or FALSE.

Returns

open file handle on created file (see [VSIFOpenL\(\)](#)).

10.11.2.5 VSILFILE CPL_DLL* VSIFOpenL (const char * *pszFilename*, const char * *pszAccess*)

Open file.

This function opens a file with the desired access. Large files (larger than 2GB) should be supported. Binary access is always implied and the "b" does not need to be included in the *pszAccess* string.

Note that the "VSILFILE *" returned since GDAL 1.8.0 by this function is *NOT* a standard C library FILE *, and cannot be used with any functions other than the "VSI*L" family of functions. They aren't "real" FILE objects.

On windows it is possible to define the configuration option GDAL_FILE_IS_UTF8 to have *pszFilename* treated as being in the local encoding instead of UTF-8, retoring the pre-1.8.0 behavior of [VSIFOpenL\(\)](#).

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

Parameters

<i>pszFilename</i>	the file to open. UTF-8 encoded.
<i>pszAccess</i>	access requested (ie. "r", "r+", "w").

Returns

NULL on failure, or the file handle.

10.11.2.6 size_t CPL_DLL VSIFReadL (void * *pBuffer*, size_t *nSize*, size_t *nCount*, VSILFILE * *fp*)

Read bytes from file.

Reads *nCount* objects of *nSize* bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters

<i>pBuffer</i>	the buffer into which the data should be read (at least <i>nCount</i> * <i>nSize</i> bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.
<i>fp</i>	file handle opened with VSIFOpenL() .

Returns

number of objects successfully read.

10.11.2.7 `int CPL_DLL VSIFReadMultiRangeL ( int nRanges, void ** ppData, const vsi_l_offset * panOffsets, const size_t * panSizes, VSILFILE * fp )`

Read several ranges of bytes from file.

Reads *nRanges* objects of *panSizes*[*i*] bytes from the indicated file at the offset *panOffsets*[*i*] into the buffer *ppData*[*i*].

Ranges must be sorted in ascending start offset, and must not overlap each other.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory or /vsicurl/.

Parameters

<i>nRanges</i>	number of ranges to read.
<i>ppData</i>	array of <i>nRanges</i> buffer into which the data should be read (<i>ppData</i> [<i>i</i>] must be at list <i>panSizes</i> [<i>i</i>] bytes).
<i>panOffsets</i>	array of <i>nRanges</i> offsets at which the data should be read.
<i>panSizes</i>	array of <i>nRanges</i> sizes of objects to read (in bytes).
<i>fp</i>	file handle opened with VSIFOpenL() .

Returns

0 in case of success, -1 otherwise.

Since

GDAL 1.9.0

10.11.2.8 `int CPL_DLL VSIFSeekL ( VSILFILE * fp, vsi_l_offset nOffset, int nWhence )`

Seek to requested offset.

Seek to the desired offset (*nOffset*) in the indicated file.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX `fseek()` call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() .
<i>nOffset</i>	offset in bytes.
<i>nWhence</i>	one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns

0 on success or -1 one failure.

10.11.2.9 `vsi_l_offset CPL_DLL VSIFTellL ( VSILFILE * fp )`

Tell current file offset.

Returns the current file read/write offset in bytes from the beginning of the file.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX `ftell()` call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() .
-----------	---

Returns

file offset in bytes.

10.11.2.10 `int CPL_DLL VSIFTruncateL ( VSILFILE * fp, vsi_l_offset nNewSize )`

Truncate/expand the file to the specified size.

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX `ftruncate()` call.

Parameters

<i>fp</i>	file handle opened with VSIFOpenL() .
<i>nNewSize</i>	new size in bytes.

Returns

0 on success

Since

GDAL 1.9.0

10.11.2.11 `size_t CPL_DLL VSIFWriteL ( const void * pBuffer, size_t nSize, size_t nCount, VSILFILE * fp )`

Write bytes to file.

Writes *nCount* objects of *nSize* bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX `fwrite()` call.

Parameters

<i>pBuffer</i>	the buffer from which the data should be written (at least <i>nCount</i> * <i>nSize</i> bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.
<i>fp</i>	file handle opened with VSIFOpenL() .

Returns

number of objects successfully written.

10.11.2.12 `GByte CPL_DLL* VSIGetMemFileBuffer ( const char * pszFilename, vsi_l_offset * pnDataLength, int bUnlinkAndSeize )`

Fetch buffer underlying memory file.

This function returns a pointer to the memory buffer underlying a virtual "in memory" file. If `bUnlinkAndSeize` is `TRUE` the filesystem object will be deleted, and ownership of the buffer will pass to the caller otherwise the underlying file will remain in existence.

Parameters

<i>pszFilename</i>	the name of the file to grab the buffer of.
<i>pnDataLength</i>	(file) length returned in this variable.
<i>bUnlinkAnd-Seize</i>	TRUE to remove the file, or FALSE to leave unaltered.

Returns

pointer to memory buffer or NULL on failure.

10.11.2.13 void VSInstallGZipFileHandler (void)

Install GZip file system handler.

A special file handler is installed that allows reading on-the-fly and writing in GZip (.gz) files.

All portions of the file system underneath the base path `"/vsigzip/"` will be handled by this driver.

Additional documentation is to be found at <http://trac.osgeo.org/gdal/wiki/UserDocs/ReadIn-Zip>

Since

GDAL 1.6.0

10.11.2.14 void CPL_DLL VSInstallMemFileHandler (void)

Install "memory" file system handler.

A special file handler is installed that allows block of memory to be treated as files. All portions of the file system underneath the base path `"/vsimem/"` will be handled by this driver.

Normal `VSI*L` functions can be used freely to create and destroy memory arrays treating them as if they were real file system objects. Some additional methods exist to efficient create memory file system objects without duplicating original copies of the data or to "steal" the block of memory associated with a memory file.

At this time the memory handler does not properly handle directory semantics for the memory portion of the filesystem. The `VSIReadDir()` 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()`, `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( VSIFFileFromMemBuffer( "/vsimem/work.dat", pabyInData,
                                       nInDataLength, FALSE ) );

    // Open memory buffer for read.
    GDALDatasetH hDS = GDALOpen( "/vsimem/work.dat", GA_ReadOnly );

    // Get output format driver.
    GDALDriverH hDriver = GDALGetDriverByName( "GTiff" );
    GDALDatasetH hOutDS;

    hOutDS = GDALCreateCopy( hDriver, "/vsimem/out.tif", hDS, TRUE, NULL,
                           NULL, NULL );
}
```

```

// close source file, and "unlink" it.
GDALClose( hDS );
VSIUnlink( "/vsimem/work.dat" );

// seize the buffer associated with the output file.

return VSIGetMemFileBuffer( "/vsimem/out.tif", pnOutDataLength, TRUE );
}

```

10.11.2.15 void CPL_DLL VSIIInstallSparseFileHandler (void)

Install /vsisparse/ virtual file handler.

The sparse virtual file handler allows a virtual file to be composed from chunks of data in other files, potentially with large spaces in the virtual file set to a constant value. This can make it possible to test some sorts of operations on what seems to be a large file with image data set to a constant value. It is also helpful when wanting to add test files to the test suite that are too large, but for which most of the data can be ignored. It could, in theory, also be used to treat several files on different file systems as one large virtual file.

The file referenced by /vsisparse/ should be an XML control file formatted something like:

```

<VSISparseFile>
  <Length>87629264</Length>
  <SubfileRegion> Stuff at start of file.
    <Filename relative="1">251_head.dat</Filename>
    <DestinationOffset>0</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>2768</RegionLength>
  </SubfileRegion>

  <SubfileRegion> RasterDMS node.
    <Filename relative="1">251_rasterdms.dat</Filename>
    <DestinationOffset>87313104</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>160</RegionLength>
  </SubfileRegion>

  <SubfileRegion> Stuff at end of file.
    <Filename relative="1">251_tail.dat</Filename>
    <DestinationOffset>87611924</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>17340</RegionLength>
  </SubfileRegion>

  <ConstantRegion> Default for the rest of the file.
    <DestinationOffset>0</DestinationOffset>
    <RegionLength>87629264</RegionLength>
    <Value>0</Value>
  </ConstantRegion>
</VSISparseFile>

```

Hopefully the values and semantics are fairly obvious.

This driver is installed by default.

10.11.2.16 void VSIIInstallStdinHandler (void)

Install /vsistdin/ file system handler.

A special file handler is installed that allows reading from the standard input stream.

The file operations available are of course limited to Read() and forward Seek() (full seek in the first MB of a file).

Since

GDAL 1.8.0

10.11.2.17 void VSInstallStdoutHandler (void)

Install /vsistdout/ file system handler.

A special file handler is installed that allows writing to the standard output stream.

The file operations available are of course limited to Write().

Since

GDAL 1.8.0

10.11.2.18 void CPL_DLL VSInstallSubFileHandler (void)

Install /visubfile/ virtual file handler.

This virtual file system handler allows access to subregions of files, treating them as a file on their own to the virtual file system functions ([VSIFOpenL\(\)](#), etc).

A special form of the filename is used to indicate a subportion of another file:

/visubfile/<offset>[_<size>],<filename>

The size parameter is optional. Without it the remainder of the file from the start offset as treated as part of the subfile. Otherwise only <size> bytes from <offset> are treated as part of the subfile. The <filename> portion may be a relative or absolute path using normal rules. The <offset> and <size> values are in bytes.

eg. /visubfile/1000_3000,/data/abc.ntf /visubfile/5000,.../xyz/raw.dat

Unlike the /vsimem/ or conventional file system handlers, there is no meaningful support for filesystem operations for creating new files, traversing directories, and deleting files within the /visubfile/ area. Only the [VSISatL\(\)](#), [VSIFOpenL\(\)](#) and operations based on the file handle returned by [VSIFOpenL\(\)](#) operate properly.

10.11.2.19 void VSInstallTarFileHandler (void)

Install /vsitar/ file system handler.

A special file handler is installed that allows reading on-the-fly in TAR (regular .tar, or compressed .tar.gz/.tgz) archives.

All portions of the file system underneath the base path "/vsitar/" will be handled by this driver.

The syntax to open a file inside a zip file is /vsitar/path/to/the/file.tar/path/inside/the/tar/file where path/to/the/file.tar is relative or absolute and path/inside/the/tar/file is the relative path to the file inside the archive.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vsitar//home/gdal/... For example gdalinfo /vsitar/myarchive.tar/subdir1/file1.tif

Syntactic sugar : if the tar archive contains only one file located at its root, just mentioning "/vsitar/path/to/the/file.tar" will work

[VSISatL\(\)](#) will return the uncompressed size in st_size member and file nature- file or directory - in st_mode member.

Directory listing is available through [VSIRReadDir\(\)](#).

Since

GDAL 1.8.0

10.11.2.20 void VSInstallZipFileHandler (void)

Install ZIP file system handler.

A special file handler is installed that allows reading on-the-fly in ZIP (.zip) archives.

All portions of the file system underneath the base path "/vsizip/" will be handled by this driver.

The syntax to open a file inside a zip file is /vsizip/path/to/the/file.zip/path/inside/the/zip/file where path/to/the/file.zip is relative or absolute and path/inside/the/zip/file is the relative path to the file inside the archive.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vsizip//home/gdal/... For example gdalinfo /vsizip/myarchive.zip/subdir1/file1.tif

Syntactic sugar : if the .zip file contains only one file located at its root, just mentioning "/vsizip/path/to/the/file.zip" will work

VSIStatL() will return the uncompressed size in st_size member and file nature- file or directory - in st_mode member.

Directory listing is available through **VSIReadDir()**.

Since GDAL 1.8.0, write capabilities are available. They allow creating a new zip file and adding new files to an already existing (or just created) zip file. Read and write operations cannot be interleaved : the new zip must be closed before being re-opened for read.

Additional documentation is to be found at <http://trac.osgeo.org/gdal/wiki/UserDocs/ReadIn-Zip>

Since

GDAL 1.6.0

10.11.2.21 int CPL_DLL VSIsCaseSensitiveFS (const char * *pszFilename*)

Returns if the filenames of the filesystem are case sensitive.

This method retrieves to which filesystem belongs the passed filename and return TRUE if the filenames of that filesystem are case sensitive.

Currently, this will return FALSE only for Windows real filenames. Other VSI virtual filesystems are case sensitive.

This methods avoid ugly #ifndef WIN32 / #endif code, that is wrong when dealing with virtual filenames.

Parameters

<i>pszFilename</i>	the path of the filesystem object to be tested. UTF-8 encoded.
--------------------	--

Returns

TRUE if the filenames of the filesystem are case sensitive.

Since

GDAL 1.8.0

10.11.2.22 void CPL_DLL* 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(). 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.

10.11.2.23 void CPL_DLL* 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(). 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.

10.11.2.24 int CPL_DLL VSIMkdir (const char * *pszPathname*, long *mode*)

Create a directory.

Create a new directory with the indicated mode. The mode is ignored on some platforms. A reasonable default mode value would be 0666. This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX mkdir() function.

Parameters

<i>pszPathname</i>	the path to the directory to create. UTF-8 encoded.
<i>mode</i>	the permissions mode.

Returns

0 on success or -1 on an error.

10.11.2.25 char CPL_DLL** VSIReadDir (const char * *pszPath*)

Read names in a directory.

This function abstracts access to directory contents. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with [CSLDestroy\(\)](#) when no longer needed.

Note that no error is issued via CPLError() if the directory path is invalid, though NULL is returned.

This function used to be known as CPLReadDir(), but the old name is now deprecated.

Parameters

<i>pszPath</i>	the relative, or absolute path of a directory to read. UTF-8 encoded.
----------------	---

Returns

The list of entries in the directory, or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

10.11.2.26 char CPL_DLL** VSIReadDirRecursive (const char * *pszPathIn*)

Read names in a directory recursively.

This function abstracts access to directory contents and subdirectories. It returns a list of strings containing the names of files and directories in this directory and all subdirectories. The resulting string list becomes the responsibility of the application and should be freed with [CSLDestroy\(\)](#) when no longer needed.

Note that no error is issued via CPLError() if the directory path is invalid, though NULL is returned.

Parameters

<i>pszPathIn</i>	the relative, or absolute path of a directory to read. UTF-8 encoded.
------------------	---

Returns

The list of entries in the directory and subdirectories or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

Since

GDAL 1.10.0

10.11.2.27 int CPL_DLL VSIRename (const char * *oldpath*, const char * *newpath*)

Rename a file.

Renames a file object in the file system. It should be possible to rename a file onto a new filesystem, but it is safest if this function is only used to rename files that remain in the same directory.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rename() function.

Parameters

<i>oldpath</i>	the name of the file to be renamed. UTF-8 encoded.
<i>newpath</i>	the name the file should be given. UTF-8 encoded.

Returns

0 on success or -1 on an error.

10.11.2.28 int CPL_DLL VSIRmdir (const char * *pszDirname*)

Delete a directory.

Deletes a directory object from the file system. On some systems the directory must be empty before it can be deleted.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rmdir() function.

Parameters

<i>pszDirname</i>	the path of the directory to be deleted. UTF-8 encoded.
-------------------	---

Returns

0 on success or -1 on an error.

10.11.2.29 int CPL_DLL VSISatExL (const char * *pszFilename*, VSISatBufL * *psStatBuf*, int *nFlags*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSISatBufL structure. For portability only the st_size (size in bytes), and st_mode (file type). This method is similar to VSISat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function, with an extra parameter to specify which information is needed, which offers a potential for speed optimizations on specialized and potentially slow virtual filesystem objects (/vsigzip/, /vsicurl/)

Parameters

<i>pszFilename</i>	the path of the filesystem object to be queried. UTF-8 encoded.
<i>psStatBuf</i>	the structure to load with information.
<i>nFlags</i>	0 to get all information, or VSI_STAT_EXISTS_FLAG, VSI_STAT_NATURE_FLAG or VSI_STAT_SIZE_FLAG, or a combination of those to get partial info.

Returns

0 on success or -1 on an error.

Since

GDAL 1.8.0

10.11.2.30 int CPL_DLL VSISatL (const char * *pszFilename*, VSISatBufL * *psStatBuf*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSISatBufL structure. For portability only the st_size (size in bytes), and st_mode (file type). This method is similar to VSISat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function.

Parameters

<i>pszFilename</i>	the path of the filesystem object to be queried. UTF-8 encoded.
<i>psStatBuf</i>	the structure to load with information.

Returns

0 on success or -1 on an error.

10.11.2.31 int CPL_DLL VSIUnlink (const char * *pszFilename*)

Delete a file.

Deletes a file object from the file system.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX unlink() function.

Parameters

<i>pszFilename</i>	the path of the file to be deleted. UTF-8 encoded.
--------------------	--

Returns

0 on success or -1 on an error.

10.12 sdtstdataset.cpp File Reference

```
#include "sdtst_al.h"  
#include "gdal_pam.h"  
#include "ogr_spatialref.h"
```

Classes

- class [SDTSDataset](#)
- class [SDTSRasterBand](#)

10.12.1 Detailed Description

exclude