libpcidsk

Contents

Chapter 1

PCIDSK SDK - Read/Write Library for the PCI PCIDSK (.pix) File Format

The **PCIDSK** (p. **??**) SDK is a C++ Library for reading and writing the **PCIDSK** (p. **??**) (.pix) geospatial file format used as the primary format of the Geomatica and related software from `PCI Geomatics`. The library is available under an `Open Source` license, and it's development is funded by PCI Geomatics. The primary author of the library is Frank Warmerdam (`warmerdam@pobox.com`).

As of November 2009 the library is still under development, but preliminary versions are now available.

1.1 Developer Oriented Documentation

- `PCIDSK Namespace`
- `Building the PCIDSK SDK on Windows and Unix`
- `PCIDSK API Tutorial`

1.2 Download

The current development version of the software is available from the `Subversion` source repository at:

`http://svn.osgeo.org/gdal/sandbox/warmerdam/pcidsk`

Packaged source distributions should be available in the `downloads` area.

Chapter 2

Building the PCIDSK SDK

As of November 2009 the **PCIDSK** (p. **??**) SDK uses a relatively simplistic set of makefiles for building on linux/unix/macos systems and on windows systems.

2.1 Building on Windows with Microsoft Visual Studio

On Windows system, building is accomplished using Microsoft Visual Studio, but via the commandline tools (NMAKE and CL). For command line builds you will normally have to have run the VCVAR32.BAT script that comes with the compiler. For Visual Studio 2003 this might be found at:

```
C:\Program Files\Microsoft Visual Studio .Net 2003\VC7\bin\VCVARS32.BAT
```

Some Visual Studios also provide a "Visual Studio Command Prompt" to be found in the start menu, which gives you a shell with the correct environment. If you plan to compile for the 64bit platform be sure to choose the correct bat / command prompt.

Once the environment is setup, you can cd to the **PCIDSK** (p. **??**) "src" subdirectory and do the following:

```
C:\PCIDSK\SRC> nmake /f makefile.vc
```

Before building as above, you may find you need to modify some commandline switches in the Makefile.vc. Generally the switches requiring adjustment are in the OPTFLAGS and CXXFLAGS macros:

```
OPTFLAGS = /Ox
CXXFLAGS = /nologo /MD /GR /EHsc $(OPTFLAGS) /W3 /I. \
/D_CRT_SECURE_NO_DEPRECATE /DLIBPCIDSK_EXPORTS /D_CRT_NONSTDC_NO_DEPRECATE
```

Adjust as required to match the needs of your application and visual studio version. The SDK should build with any version of Microsoft Visual Studio 2002 or later.

The result of a build should be a pcidsk.lib file in the pcidsk directory.

2.2 Building on Linux, Unix and MacOS X

On unix and unix-like operating systems it should be possible to build using the Makefile in the top level **PCIDSK** (p. **??**) directory. Currently the SDK does not provide a configure or similar environment. Instead the src/Makefile, and tests/Makefile can be edited to adjust the compiler switches. Key switches are in the CXXFLAGS macro. Also, in src/Makefile you can comment out, or adjust the JPEG_FLAGS switch to control whether libjpeg is used for JPEG compression support.

```
JPEG_FLAGS = -DHAVE_LIBJPEG
CXXFLAGS = -O -Wall -fPIC -I. $(JPEG_FLAGS)
```

Then "make" in the main pcidsk directory should build the SDK as well as the test programs. The "make check" command may be used to run the test suite once the test suite data is available.

By default the files src/pcidsk.a (a static library) and src/pcidsk.so (a shared library) are built.

Chapter 3

PCIDSK API Tutorial

The **PCIDSK** (p. ??) file format is a geospatial file format which includes raster channels (or bands) and auxilary segments which include information like georeferencing, color lookup tables, vector feature data and many other special data types.

The **PCIDSK** (p. ??) SDK provides C++ classes to interface to the various different components of the file format. The PCIDSKFile is used to interface with the file as a whole. The PCIDSKChannel is used to interface with a raster channel (or band). The PCIDSKSegment is the generic interface to auxilary segments. There are additional interface classes available for some of the specific segment types.

3.1 Include files, and Namespaces

The **PCIDSK** (p. ??) classes and interfaces are defined in a variety of include files, but generally it is sufficient to include just the main include file, **pcidsk.h** (p. ??).

```
#include <pcidsk.h>
```

All **PCIDSK** (p. ??) classes are part of the C++ **PCIDSK** (p. ??) namespace. It is often convenient to use this namespace explicitly making all classes and methods directly available. This can be accomplished by putting the follow declaration after the include of **pcidsk.h** (p. ??).

```
using namespace PCIDSK;
```

However, for the balance of this tutorial we will explicitly include the namespace when accessing classes to make it clearer which definitions are coming from the SDK.

3.2 Opening the File

We start with a fairly trivial program using the SDK. In this program we open the desired file (irvine.pix), and print out general information on the number of pixels, lines and channels.

```
#include "gdal_priv.h"

int main()
{
    PCIDSK::PCIDSKFile *file = PCIDSK::Open( "irvine.pix", "r", NULL );

    printf( "File: %dC x %dR x %dC (%s)\n",
            file->GetWidth(), file->GetHeight(), file->GetChannels(),
            file->GetInterleaving().c_str() );
    delete file;
}
```

Note the use of the **PCIDSK** (p. ??):: prefix for classes (like PCIDSKFile) and functions (like **Open()** (p. ??)) coming from the **PCIDSK** (p. ??) namespace. The **PCIDSK::Open** (p. ??) function is used to open the file, and various methods on the object are used to get the number of pixels, lines and channels as well as the interleaving.

The **PCIDSK** (p. ??) SDK API takes and returns strings using the std::string class, so to get a C printable string out we use the c_str() method on the returned string from the **PCIDSK::PCIDSKFile::GetInterleaving()** (p. ??) method.

3.3 Reading Image Data

The **PCIDSK::PCIDSKChannel** (p. **??**) class is used to interface with raster image channels. Normally raster access is accomplished one band at a time as demonstrated in this snippit derived from tests/pcidsk_-read.cpp.

```
for( int channel = 1; channel <= file->GetChannels(); channel++ )
{
    PCIDSK::PCIDSKChannel *channel = file->GetChannel( channel );
    printf( "Channel %d of type %s.\n",
            channel,
            PCIDSK::DataTypeName(channel->GetType()).c_str() );;

    int i_block;
    int x_block_count =
            (channel->GetWidth() + channel->GetBlockWidth()-1)
            / channel->GetBlockWidth();
    int y_block_count =
            (channel->GetHeight() + channel->GetBlockHeight()-1)
            / channel->GetBlockHeight();
    int block_size = PCIDSK::DataTypeSize(channel->GetType()) *
            channel->GetBlockWidth() *
            channel->GetBlockHeight();
    void *block_buffer = malloc( block_size );

    int block_count = x_block_count * y_block_count;

    for( i_block = 0; i_block < block_count; i_block++ )
    {
        channel->ReadBlock( i_block, block_buffer );
        /* ... do something with the imagery ... */
    }
    free( block_buffer );
}
```

Note that there is no effort to destroy the PCIDSKChannel object when no longer needed. Component objects like channels and segments are owned by the **PCIDSK::PCIDSKFile** (p. **??**) and should not be destroyed directly. The accessors like GetChannel() return a pointer that remains owned by the file.

In this case we loop through all the channels, report the channel number and type, and then loop through all the blocks in the channel reading them. For simplicity we don't actually do anything with them here.

The **PCIDSK::PCIDSKChannel::ReadBlock()** (p. **??**) method is used to read imagery. It reads exactly one block on the "natural block boundary" and it is read in the original data type of the channel (8U, 16U, 16S or 32R). The blocking depends on the organization of the data on disk. For pixel interleaved or band interleaved files this is one scanline while for tiled files this is one tile. The block size is established with the **PCIDSK::PCIDSKChannel::GetBlockWidth()** (p. **??**) and **PCIDSK::PCIDSKChannel::GetBlockHeight()** (p. **??**) methods on the channel. The data type is determined with the **PCIDSK::PCIDSKChannel::GetType()** (p. **??**) method.

The ReadBlock() method also has optional parameter to read a subwindow, but the **PCIDSK** (p. **??**) SDK does *not* provide a generic high level "read an arbitrary window of the image" method such as is provided by libraries like `GDAL` or PCI's own `GeoGateway/GDB`. This is in keeping with the **PCIDSK** (p. **??**) SDK policy of providing low level "file organization oriented" data access with a minimum of convenience services. It is intended that higher level libraries would provide caching, windowing and other convenience services.

Chapter 4

Namespace Index

4.1 Namespace List

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

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

Chapter 7

Namespace Documentation

7.1 PCIDSK Namespace Reference

Namespace for all **PCIDSK** (p. **??**) Library classes and functions.

Classes

- class **PCIDSKChannel**

  Interface to one ***PCIDSK*** *(p.* **??***) channel (band).*

- class **PCIDSKException**

  Generic SDK Exception.

- class **PCIDSKFile**

  Top interface to ***PCIDSK*** *(p.* **??***) (.pix) files.*

- class **PCIDSKGeoref**

  Interface to ***PCIDSK*** *(p.* **??***) georeferencing segment.*

- class **PCIDSKInterfaces**

  Collection of ***PCIDSK*** *(p.* **??***) hookable interfaces.*

- class **IOInterfaces**

  IO Interface class.

- class **Mutex**

  Mutex *(p.* **??***) interface class.*

- class **PCIDSK_PCT**

  Interface to ***PCIDSK*** *(p.* **??***) pseudo-color segment.*

- class **PCIDSKSegment**

  Public tnterface for the ***PCIDSK*** *(p.* **??***) Segment Type.*

- struct **ShapeVertex**

Structure for an x,y,z point.

- class **ShapeField**

  Attribute field value.

- class **PCIDSKVectorSegment**

  Interface to ***PCIDSK*** *(*p. **??***) vector segment.*

- class **ShapeIterator**

  Iterator over shapeids in a vector segment.

Typedefs

- typedef int32 **ShapeId**

  Type used for shape identifier, use constant NullShapeId as a NULL value.

Enumerations

- enum **UnitCode** { **UNIT_US_FOOT** = 1, **UNIT_METER** = 2, **UNIT_DEGREE** = 4, **UNIT_-INTL_FOOT** = 5 }
- enum **ShapeFieldType** {

  FieldTypeNone = 0, **FieldTypeFloat** = 1, **FieldTypeDouble** = 2, **FieldTypeString** = 3,

  FieldTypeInteger = 4, **FieldTypeCountedInt** = 5 }

  Attribute field types.

- enum **eChanType** {

  CHN_8U = 0, **CHN_16S** = 1, **CHN_16U** = 2, **CHN_32R** = 3,

  CHN_UNKNOWN = 99 }

  Channel pixel data types.

- enum **eSegType** {

  SEG_UNKNOWN = -1, **SEG_BIT** = 101, **SEG_VEC** = 116, **SEG_SIG** = 121,

  SEG_TEX = 140, **SEG_GEO** = 150, **SEG_ORB** = 160, **SEG_LUT** = 170,

  SEG_PCT = 171, **SEG_BLUT** = 172, **SEG_BPCT** = 173, **SEG_BIN** = 180,

  SEG_ARR = 181, **SEG_SYS** = 182, **SEG_GCPOLD** = 214, **SEG_GCP2** = 215 }

  Segment types.

Functions

- **PCIDSKFile** ∗ **Open** (std::string filename, std::string access, const **PCIDSKInterfaces** ∗interfaces)
- **PCIDSKFile** ∗ **Create** (std::string filename, int pixels, int lines, int channel_count, **eChanType** ∗channel_types, std::string options, const **PCIDSKInterfaces** ∗interfaces)
- void **ThrowPCIDSKException** (const char ∗fmt,...)

  throw a formatted exception.

- const **IOInterfaces** ∗ **GetDefaultIOInterfaces** ()
- **Mutex** ∗ **DefaultCreateMutex** (void)
- std::string **ShapeFieldTypeName** (**ShapeFieldType** type)

    Translate field type into a textual description.

- int **DataTypeSize** (**eChanType**)
- std::string **DataTypeName** (**eChanType**)
- std::string **SegmentTypeName** (**eSegType**)

7.1.1 Detailed Description

Namespace for all **PCIDSK** (p. **??**) Library classes and functions.

7.1.2 Enumeration Type Documentation

7.1.2.1 enum PCIDSK::eChanType

Channel pixel data types.

Enumerator:

CHN_8U	8 bit unsigned byte
CHN_16S	16 bit signed integer
CHN_16U	16 bit unsigned integer
CHN_32R	32 bit ieee floating point
CHN_UNKNOWN	unknown channel type

7.1.3 Function Documentation

7.1.3.1 PCIDSKFile ∗ PCIDSK::Create (std::string *filename*, int *pixels*, int *lines*, int *channel_count*, eChanType ∗ *channel_types*, std::string *options*, const PCIDSKInterfaces ∗ *interfaces*)

Create a **PCIDSK** (p. **??**) (.pix) file.

Parameters:

filename	the name of the **PCIDSK** (p. **??**) file to create.
pixels	the width of the new file in pixels.
lines	the height of the new file in scanlines.
channel_count	the number of channels to create.
channel_types	an array of types for all the channels, or NULL for all CHN_8U channels.
option	creation options (interleaving, etc)
interfaces	Either NULL to use default interfaces, or a pointer to a populated interfaces object.

Returns:

a pointer to a file object for accessing the **PCIDSK** (p. **??**) file.

References CHN_16S, CHN_16U, CHN_32R, CHN_8U, PCIDSK::PCIDSKFile::CreateSegment(), PCIDSK::PCIDSKFile::GetSegment(), PCIDSK::PCIDSKInterfaces::io, Open(), ThrowPCIDSKException(), and PCIDSK::PCIDSKGeoref::WriteSimple().

7.1.3.2 std::string PCIDSK::DataTypeName (eChanType *chan_type*)

Return name for the data type.

The returned values are suitable for display to people, and matches the portion of the name after the underscore (ie. "8U" for CHN_8U.

Parameters:

chan_type the channel type enumeration value to be translated.

Returns:

a string representing the data type.

References CHN_16S, CHN_16U, CHN_32R, and CHN_8U.

7.1.3.3 int PCIDSK::DataTypeSize (eChanType *chan_type*)

Return size of data type.

Parameters:

chan_type the channel type enumeration value.

Returns:

the size of the passed data type in bytes, or zero for unknown values.

References CHN_16S, CHN_16U, CHN_32R, and CHN_8U.

7.1.3.4 PCIDSKFile ∗ PCIDSK::Open (std::string *filename*, std::string *access*, const PCIDSKInterfaces ∗ *interfaces*)

Open a **PCIDSK** (p. **??**) (.pix) file.

This function attempts to open the named file, with the indicated access and the provided set of system interface methods.

Parameters:

filename the name of the **PCIDSK** (p. **??**) file to access.

access either "r" for read-only, or "r+" for read-write access.

interfaces Either NULL to use default interfaces, or a pointer to a populated interfaces object.

Returns:

a pointer to a file object for accessing the **PCIDSK** (p. **??**) file.

References PCIDSK::PCIDSKInterfaces::CreateMutex, PCIDSK::PCIDSKInterfaces::io, and ThrowPCIDSKException().

Referenced by Create().

7.1.3.5 std::string PCIDSK::SegmentTypeName (eSegType *type*)

Return name for segment type.

Returns a short name for the segment type code passed in. This is normally the portion of the enumeration name that comes after the underscore - ie. "BIT" for SEG_BIT.

Parameters:

type the segment type code.

Returns:

the string for the segment type.

7.1.3.6 std::string PCIDSK::ShapeFieldTypeName (ShapeFieldType *type*) `[inline]`

Translate field type into a textual description.

Parameters:

type the type enumeration value to translate.

Returns:

name for field type.

7.1.3.7 void PCIDSK::ThrowPCIDSKException (const char ∗ *fmt*, *...*)

throw a formatted exception. This function throws a **PCIDSK** (p. **??**) Exception by reference after formatting the message using the given printf style format and arguments. This function exists primarily so that throwing an exception can be done in one line of code, instead of declaring an exception and then throwing it.

Parameters:

fmt the printf style format (eg. "Illegal value:%d")

... additional arguments as required by the format string.

References PCIDSK::PCIDSKException::vPrintf().

Referenced by Create(), and Open().

Chapter 8

Class Documentation

8.1 PCIDSK::IOInterfaces Class Reference

IO Interface class.

`#include <pcidsk_io.h>`

Public Member Functions

- virtual void ∗ **Open** (std::string filename, std::string access) const =0
- virtual uint64 **Seek** (void ∗io_handle, uint64 offset, int whence) const =0
- virtual uint64 **Tell** (void ∗io_handle) const =0
- virtual uint64 **Read** (void ∗buffer, uint64 size, uint64 nmemb, void ∗io_handle) const =0
- virtual uint64 **Write** (const void ∗buffer, uint64 size, uint64 nmemb, void ∗io_handle) const =0
- virtual int **Eof** (void ∗io_handle) const =0
- virtual int **Flush** (void ∗io_handle) const =0
- virtual int **Close** (void ∗io_handle) const =0

8.1.1 Detailed Description

IO Interface class.

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

- pcidsk_io.h

8.2 PCIDSK::Mutex Class Reference

Mutex (p. **??**) interface class.

Public Member Functions

- virtual int **Acquire** ()=0

  Acquire the mutex.

- virtual int **Release** ()=0

  Release the mutex.

8.2.1 Detailed Description

Mutex (p. **??**) interface class. The **Mutex** (p. **??**) class is the standard interface for mutexes in the **PCIDSK** (p. **??**) library. A mutex provides critical section locking in multi-threaded applications. Applications may provide custom mutex implementations by passing their own CreateMutex() implementation in **PCIDSK::PCIDSKInterfaces** (p. **??**) instead of the default one provided by the library (CreateDefaultMutex()). The library will create mutexes using the **PCIDSK::PCIDSKInterfaces::CreateMutex()** (p. **??**) function.

Note that mutexes are created in the unlocked condition.

Mutexes may be destroyed with delete when no longer required.

8.2.2 Member Function Documentation

8.2.2.1 int PCIDSK::Mutex::Acquire () `[pure virtual]`

Acquire the mutex. Note that control will block in **Acquire()** (p. **??**) until such time as the mutex can be acquired for this thread.

Returns:

TRUE on success.

8.2.2.2 int PCIDSK::Mutex::Release () `[pure virtual]`

Release the mutex. Release this mutex so that it may be acquired by another thread.

Returns:

TRUE on success.

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

- pcidsk_mutex.h
- pcidskmutex.dox

8.3 PCIDSK::PCIDSK_PCT Class Reference

Interface to **PCIDSK** (p. **??**) pseudo-color segment.

`#include <pcidsk_pct.h>`

Public Member Functions

- virtual void **ReadPCT** (unsigned char pct[768])=0

  Read a PCT Segment (SEG_PCT).

- virtual void **WritePCT** (unsigned char pct[768])=0

  Write a PCT Segment.

8.3.1 Detailed Description

Interface to **PCIDSK** (p. **??**) pseudo-color segment.

8.3.2 Member Function Documentation

8.3.2.1 virtual void PCIDSK::PCIDSK_PCT::ReadPCT (unsigned char *pct*[768]) `[pure virtual]`

Read a PCT Segment (SEG_PCT).

Parameters:

pct Pseudo-Color Table buffer (768 entries) into which the pseudo-color table is read. It consists of the red gun output values (pct[0-255]), followed by the green gun output values (pct[256-511]) and ends with the blue gun output values (pct[512-767]).

8.3.2.2 virtual void PCIDSK::PCIDSK_PCT::WritePCT (unsigned char *pct*[768]) `[pure virtual]`

Write a PCT Segment.

Parameters:

pct Pseudo-Color Table buffer (768 entries) from which the pseudo-color table is written. It consists of the red gun output values (pct[0-255]), followed by the green gun output values (pct[256-511]) and ends with the blue gun output values (pct[512-767]).

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

- pcidsk_pct.h

8.4 PCIDSK::PCIDSKChannel Class Reference

Interface to one **PCIDSK** (p. **??**) channel (band).

```
#include <pcidsk_channel.h>
```

Public Member Functions

- virtual int **GetBlockWidth** ()=0

  Fetch image block width.

- virtual int **GetBlockHeight** ()=0

  Fetch image block height.

- virtual int **GetBlockCount** ()=0

  Fetch image block count.

- virtual int **GetWidth** ()=0

  Fetch image width.

- virtual int **GetHeight** ()=0

  Fetch image height.

- virtual **eChanType GetType** ()=0

  Fetch pixel data type.

- virtual int **ReadBlock** (int block_index, void ∗buffer, int win_xoff=-1, int win_yoff=-1, int win_-xsize=-1, int win_ysize=-1)=0

  read block of image data from disk.

- virtual int **WriteBlock** (int block_index, void ∗buffer)=0

  write block of image data from disk.

- virtual int **GetOverviewCount** ()=0

  Fetch number of overviews.

- virtual **PCIDSKChannel** ∗ **GetOverview** (int i)=0

  Fetch Overview.

- virtual std::string **GetMetadataValue** (std::string key)=0

  Fetch metadata value.

- virtual void **SetMetadataValue** (std::string key, std::string value)=0

  Set metadata value.

- virtual std::vector< std::string > **GetMetadataKeys** ()=0

  Fetch metadata keys.

- virtual void **Synchronize** ()=0

  Write pending information to disk.

8.4.1 Detailed Description

Interface to one **PCIDSK** (p. **??**) channel (band).

8.4.2 Member Function Documentation

8.4.2.1 int PCIDSK::PCIDSKChannel::GetBlockCount () `[pure virtual]`

Fetch image block count.

Returns:

returns the number of blocks in a channel.

8.4.2.2 int PCIDSK::PCIDSKChannel::GetBlockHeight () `[pure virtual]`

Fetch image block height. Channel (band) access is done in blocks according to the natural blocking of the data. For PIXEL, BAND and some FILE interleaved files a block is one scanline. For tiled files it is a tile.

Returns:

returns the height of a block in pixels.

8.4.2.3 int PCIDSK::PCIDSKChannel::GetBlockWidth () `[pure virtual]`

Fetch image block width. Channel (band) access is done in blocks according to the natural blocking of the data. For PIXEL, BAND and some FILE interleaved files a block is one scanline. For tiled files it is a tile.

Returns:

returns the width of a block in pixels.

8.4.2.4 int PCIDSK::PCIDSKChannel::GetHeight () `[pure virtual]`

Fetch image height.

Returns:

returns the channel height in pixels.

8.4.2.5 std::vector< std::string > PCIDSK::PCIDSKChannel::GetMetadataKeys () `[pure virtual]`

Fetch metadata keys. Returns a vector of metadata keys that occur on this object. The values associated with each may be fetched with **GetMetadataValue()** (p. **??**).

Returns:

list of keys

See also:

GetMetadataValue() (p. **??**)

8.4.2.6 std::string PCIDSK::PCIDSKChannel::GetMetadataValue (std::string *key*) `[pure virtual]`

Fetch metadata value. Note that the returned pointer is to an internal structure and it may become invalid if another thread modifies the metadata for this object.

Parameters:

key the key to fetch the value for.

Returns:

the value of the indicated metadata item, or an empty string if it does not exist on the target object.

See also:

GetMetadataKeys() (p. **??**)

8.4.2.7 PCIDSK::PCIDSKChannel ∗ PCIDSK::PCIDSKChannel::GetOverview (int *i*) `[pure virtual]`

Fetch Overview. This method fetches a pointer to the requested overview. The return **PCIDSKChannel** (p. **??**) object remains owned by the parent **PCIDSKChannel** (p. **??**) but may be otherwise accessed using the normal **PCIDSKChannel** (p. **??**) mechanisms. The size of the overview in pixels and lines will reveal it's decimation factor relative to the base image.

Parameters:

i the zero based index of the overview to fetch (from zero to **GetOverviewCount()** (p. **??**)-1)

Returns:

the overview channel object.

8.4.2.8 int PCIDSK::PCIDSKChannel::GetOverviewCount () `[pure virtual]`

Fetch number of overviews.

Returns:

the number of overviews available for this channel.

See also:

GetOverview() (p. **??**)

8.4.2.9 eChanType PCIDSK::PCIDSKChannel::GetType () `[pure virtual]`

Fetch pixel data type.

Returns:

the pixel data type for this channel.

8.4.2.10 int PCIDSK::PCIDSKChannel::GetWidth () `[pure virtual]`

Fetch image width.

Returns:

returns the channel width in pixels.

8.4.2.11 int PCIDSK::PCIDSKChannel::ReadBlock (int *block_index*, void ∗ *buffer*, int *win_xoff* = `-1`, int *win_yoff* = `-1`, int *win_xsize* = `-1`, int *win_ysize* = `-1`) `[pure virtual]`

read block of image data from disk. The buffer into which imagery is read should be preallocated large enough to hold **GetBlockWidth()** (p. **??**) ∗ **GetBlockHeight()** (p. **??**) ∗ DataTypeSize(**GetType()** (p. **??**)) bytes, or win_xsize∗win_ysize∗DataType(**GetType()** (p. **??**)) if subwindowing is being used. Image data is returned in the pixel data type reported by **GetType()** (p. **??**) and in the local systems byte order (for types larger than one byte).

For scanline oriented images the block index is the scanline index. For tiled images the block_index starts at 0 at the top left tile. The tile to the right of that is 1, and the first tile in the second row is equal to "blocks_per_row".

Partial (incomplete) blocks at the right or bottom of images that are not a multiple of the block width or height in size will be zero filled out to the block size.

The win_xoff, win_yoff, win_xsize, and win_ysize parameters may be used to select a subwindow of the desired block. By default the whole block is returned.

Parameters:

block_index zero based block index to read.

buffer the buffer into which the block will be read.

win_xoff the x (right) offset into the block to start reading.

win_yoff the y (down) offset into the block to start reading.

win_xsize the width of the window to read from the block.

win_ysize the height of the window to read from the block.

8.4.2.12 void PCIDSK::PCIDSKChannel::SetMetadataValue (std::string *key*, std::string *value*) `[pure virtual]`

Set metadata value. Assign the metadata value associated with the passed key on this object. The file needs to be open for update. Note that keys should be well formed tokens (no special characters, spaces, etc).

Parameters:

key the key to fetch the value for.

value the value to assign to the key. An empty string deletes the item.

See also:

GetMetadataValue() (p. **??**)

8.4.2.13 void PCIDSK::PCIDSKChannel::Synchronize () `[pure virtual]`

Write pending information to disk. Some write and update operations on **PCIDSK** (p. **??**) files are not written to disk immediately after write calls. This method will ensure that any pending writes are flushed through to disk.

NOTE: Currently this method does not invalidate read-cached information. At some point in the future it might be extended to do this as well.

8.4.2.14 int PCIDSK::PCIDSKChannel::WriteBlock (int *block_index*, void ∗ *buffer*) `[pure virtual]`

write block of image data from disk. The buffer from which imagery is read should be preallocated large enough to hold **GetBlockWidth()** (p. **??**) ∗ **GetBlockHeight()** (p. **??**) ∗ DataTypeSize(**GetType()** (p. **??**)) bytes. Image data is expected in the pixel data type reported by **GetType()** (p. **??**) and in the local systems byte order (for types larger than one byte).

For scanline oriented images the block index is the scanline index. For tiled images the block_index starts at 0 at the top left tile. The tile to the right of that is 1, and the first tile in the second row is equal to "blocks_per_row".

Partial (incomplete) blocks at the right or bottom of images that are not a multiple of the block width or height in size may be zero filled out to the block size.

Parameters:

block_index zero based block index to read.

buffer the buffer from which the block will be written.

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

- pcidsk_channel.h
- pcidskchannel.dox

8.5 PCIDSK::PCIDSKException Class Reference

Generic SDK Exception.

Public Member Functions

- **PCIDSKException** (const char ∗fmt,...)
- virtual ∼**PCIDSKException** () throw ()
- void **vPrintf** (const char ∗fmt, va_list list)
- virtual const char ∗ **what** () const throw ()

    fetch exception message.

8.5.1 Detailed Description

Generic SDK Exception. The **PCIDSKException** (p. **??**) class is used for all errors thrown by the **PCIDSK** (p. **??**) library. It includes a formatted message and is derived from std::exception. The **PCIDSK** (p. **??**) library throws all exceptions as pointers, and library exceptions should be caught like this:

```
try
{
     PCIDSKFile *file = PCIDSK::Open( "irvine.pix, "r", NULL );
}
catch( PCIDSK::PCIDSKException &ex )
{
    fprintf( stderr, "PCIDSKException:\n%s\n", ex.what() );
    exit( 1 );
}
```

8.5.2 Constructor & Destructor Documentation

8.5.2.1 PCIDSKException::PCIDSKException (const char ∗ *fmt*, ...)

Create exception with formatted message.

This constructor supports formatting of an exception message using printf style format and additional arguments.

Parameters:

fmt the printf style format (eg. "Illegal value:%d")

... additional arguments as required by the format string.

References vPrintf().

8.5.2.2 PCIDSKException::∼PCIDSKException () throw () `[virtual]`

Destructor.

8.5.3 Member Function Documentation

8.5.3.1 void PCIDSKException::vPrintf (const char ∗ *fmt*, va_list *args*)

Format a message.

Assigns a message to an exception using printf style formatting and va_list arguments (similar to vfprintf().

Parameters:

fmt printf style format string.

args additional arguments as required.

Referenced by PCIDSKException(), and PCIDSK::ThrowPCIDSKException().

8.5.3.2 const char ∗ PCIDSKException::what () const throw () `[inline, virtual]`

fetch exception message.

Returns:

a pointer to the internal message associated with the exception.

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

- pcidsk_exception.h
- core/pcidskexception.cpp

8.6 PCIDSK::PCIDSKFile Class Reference

Top interface to **PCIDSK** (p. **??**) (.pix) files.

```
#include <pcidsk_file.h>
```

Public Member Functions

- virtual **PCIDSKInterfaces** ∗ **GetInterfaces** ()=0

  Fetch hookable interfaces in use with this file.

- virtual **PCIDSKChannel** ∗ **GetChannel** (int band)=0

  Fetch channel interface object.

- virtual **PCIDSKSegment** ∗ **GetSegment** (int segment)=0

  Fetch segment interface object.

- virtual std::vector< **PCIDSKSegment** ∗ > **GetSegments** ()=0
- virtual **PCIDSK::PCIDSKSegment** ∗ **GetSegment** (int type, std::string name, int previous=0)=0

  Fetch segment interface object.

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

  Fetch image width.

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

  Fetch image height.

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

  Fetch channel (band) count.

- virtual std::string **GetInterleaving** () const =0

  Fetch file interleaving method.

- virtual bool **GetUpdatable** () const =0

  Check readonly/update status.

- virtual uint64 **GetFileSize** () const =0

  Fetch file size.

- virtual int **CreateSegment** (std::string name, std::string description, **eSegType** seg_type, int data_-blocks)=0

  create a new auxiliary segment

- virtual void **DeleteSegment** (int segment)=0

  delete an existing segment

- virtual void **CreateOverviews** (int chan_count, int ∗chan_list, int factor, std::string resampling)=0

  Create an overview level.

- virtual int **GetPixelGroupSize** () const =0 *fetch number of bytes per pixel*

- virtual void ∗ **ReadAndLockBlock** (int block_index, int xoff=-1, int xsize=-1)=0

  Read a block.

- virtual void **UnlockBlock** (bool mark_dirty=false)=0

  Unlock block.

- virtual void **WriteToFile** (const void ∗buffer, uint64 offset, uint64 size)=0

  Write data to file.

- virtual void **ReadFromFile** (void ∗buffer, uint64 offset, uint64 size)=0

  Read data from file.

- virtual void **GetIODetails** (void ∗∗∗io_handle_pp, **Mutex** ∗∗∗io_mutex_pp, std::string filename="")=0
- virtual std::string **GetMetadataValue** (const std::string &key)=0
- virtual void **SetMetadataValue** (const std::string &key, const std::string &value)=0
- virtual std::vector< std::string > **GetMetadataKeys** ()=0

  Fetch metadata keys.

- virtual void **Synchronize** ()=0

  Write pending information to disk.

8.6.1 Detailed Description

Top interface to **PCIDSK** (p. **??**) (.pix) files.

8.6.2 Member Function Documentation

8.6.2.1 void PCIDSK::PCIDSKFile::CreateOverviews (int *chan_count*, int ∗ *chan_list*, int *factor*, std::string *resampling*) `[pure virtual]`

Create an overview level. An overview is created on the indicated list of channels. If chan_count is zero, then it will be created for all channels on the file. The overview will have a size determined by dividing the base image level by "factor". The file needs to be open in update mode.

If the requested overview level already exists an exception will be thrown.

While this function creates the overview level, and records the resampling method in metadata, it does not actually compute and assign imagery to the overview. This must be done externally by the application. Overview computation is not a function of the **PCIDSK** (p. **??**) SDK.

Parameters:

chan_count the number of channels listed in chan_list.

chan_list the channels for which the overview level should be created.

factor the overview decimation factor.

resampling the resampling to be used for this overview - one of "NEAREST", "AVERAGE" or "MODE".

8.6.2.2 int PCIDSK::PCIDSKFile::CreateSegment (std::string *name*, std::string *description*, eSegType *seg_type*, int *data_blocks*) `[pure virtual]`

create a new auxiliary segment A new segment of the desired type is created and assigned the given name and description. The segment number is returned. The segment is created with the requested number of data blocks. If this is zero a default size may be assigned for some types with fixed sizes.

This method may fail if there are no unused segment pointers available in the file.

Parameters:

name segment name, at most eight characters.

description segment description, at most 64 characters.

seg_type the segment type.

data_blocks the number of data blocks the segment should be initially assigned. If zero a default value may be used for some fixed sized segments.

Returns:

the number of the segment created.

Referenced by PCIDSK::Create().

8.6.2.3 int PCIDSK::PCIDSKFile::DeleteSegment (int *segment*) `[pure virtual]`

delete an existing segment Delete the indicated segment number. The segment must currently exist. The internal **PCIDSKSegment** (p. **??**) object associated with this segment will also be destroyed, and any references to it from **GetSegment()** (p. **??**) or other sources should not be used by the application after this call is made.

Parameters:

segment the number of the segment to delete from the file.

8.6.2.4 PCIDSKChannel * PCIDSK::PCIDSKFile::GetChannel (int *band*) `[pure virtual]`

Fetch channel interface object. The returned channel object remains owned by the **PCIDSKFile** (p. **??**) and should not be deleted by the caller, and will become invalid after the **PCIDSKFile** (p. **??**) is closed (deleted).

Parameters:

band the band number to fetch (one based).

Returns:

pointer to internally managed channel object.

8.6.2.5 int PCIDSK::PCIDSKFile::GetChannels () const `[pure virtual]`

Fetch channel (band) count.

Returns:

the number of channels on this file.

8.6.2.6 uint64 PCIDSK::PCIDSKFile::GetFileSize () const `[pure virtual]`

Fetch file size.

Returns:

returns the size of the **PCIDSK** (p. **??**) file in bytes, as recorded in the file header.

8.6.2.7 int PCIDSK::PCIDSKFile::GetHeight () const `[pure virtual]`

Fetch image height.

Returns:

the height of the file in pixels.

8.6.2.8 const char * PCIDSK::PCIDSKFile::GetInterleaving () const `[pure virtual]`

Fetch file interleaving method.

Returns:

the interleaving name, one of "PIXEL", "BAND" or "FILE". Note that tiled files will be reported as "FILE" interleaving.

8.6.2.9 std::vector< std::string > PCIDSK::PCIDSKFile::GetMetadataKeys () `[pure virtual]`

Fetch metadata keys. Returns a vector of metadata keys that occur on this object. The values associated with each may be fetched with GetMetadataValue().

Returns:

list of keys

See also:

GetMetadataValue()

8.6.2.10 int PCIDSK::PCIDSKFile::GetPixelGroupSize () const `[pure virtual]`

fetch number of bytes per pixel Returns the number of bytes for each pixel group. Each pixel group consists of the values for all the channels in the file in order.

Note:

This method should only be called for **GetInterleaving()** (p. **??**) == "PIXEL" files.

Returns:

the size of a pixel group in bytes.

See also:

ReadAndLockBlock() (p. **??**)

8.6.2.11 PCIDSKSegment ∗ PCIDSK::PCIDSKFile::GetSegment (int *type*, std::string *name*, int *previous* = 0) `[pure virtual]`

Fetch segment interface object. If available, a segment of the specified type and name is returned. The search is started after segment "previous" if none-zero.

The returned segment object remains owned by the **PCIDSKFile** (p. **??**) and should not be deleted by the caller, and will become invalid after the **PCIDSKFile** (p. **??**) is closed (deleted).

Parameters:

type the segment type desired, or SEG_UNKNOWN for any type.

name the segment name or "" for any name.

previous segment after which the search should start or 0 (default) to start from the start.

Returns:

pointer to internally managed segment object or NULL if none correspond to the request.

8.6.2.12 PCIDSKSegment ∗ PCIDSK::PCIDSKFile::GetSegment (int *segment*) `[pure virtual]`

Fetch segment interface object. The returned segment object remains owned by the **PCIDSKFile** (p. **??**) and should not be deleted by the caller, and will become invalid after the **PCIDSKFile** (p. **??**) is closed (deleted).

Parameters:

segment the segment number to fetch (one based).

Returns:

pointer to internally managed segment object.

Referenced by PCIDSK::Create().

8.6.2.13 bool PCIDSK::PCIDSKFile::GetUpdatable () const `[pure virtual]`

Check readonly/update status.

Returns:

true if the file is open for update, or false if it is read-only.

8.6.2.14 int PCIDSK::PCIDSKFile::GetWidth () const `[pure virtual]`

Fetch image width.

Returns:

the width of the file in pixels.

8.6.2.15 uint8 ∗ PCIDSK::PCIDSKFile::ReadAndLockBlock (int *block_index*, int *win_xoff* = `-1`, int *win_xsize* = `-1`) `[pure virtual]`

Read a block. Returnst the pointer to an internal buffer for the indicated block. The buffer is owned by the **PCIDSKFile** (p. **??**) object, but will be considered locked and available for the application code to read and modify until the **UnlockBlock()** (p. **??**) method is called. The buffer will contain all the pixel values for the requested block in pixel interleaved form.

The win_xoff, and win_xsize parameters may be used to select a subregion of the scanline block to read. By default the whole scanline is read.

Note:

This method should only be called for **GetInterleaving()** (p. **??**) == "PIXEL" files. Normal imagery access should be via the **PCIDSKChannel** (p. **??**) class. This method is provided on the **PCIDSKFile** (p. **??**) for pixel interleaved files to allow optimized access for this one case.

Parameters:

block_index the zero based block(scanline) to be read.

win_xoff the offset into the scanline to start reading values.

win_xsize the number of pixels to read.

Returns:

pointer to an internal buffer with pixel data in it.

8.6.2.16 void PCIDSK::PCIDSKFile::ReadFromFile (void ∗ *buffer*, uint64 *offset*, uint64 *size*) `[pure virtual]`

Read data from file. This method is normally only used by the **PCIDSK** (p. **??**) library itself, and provides a mechanism to read directly from the **PCIDSK** (p. **??**) file. Applications should normally use the **PCIDSK::PCIDSKChannel::ReadBlock()** (p. **??**) method for imagery, or appropriate **PCIDSK::PCIDSKSegment** (p. **??**) methods for reading segment data.

Parameters:

buffer pointer to the buffer into which the data should be read.

offset the byte offset in the file (zero based) at which to read the data.

size the number of bytes from the file to read.

8.6.2.17 void PCIDSK::PCIDSKFile::Synchronize () `[pure virtual]`

Write pending information to disk. Some write and update operations on **PCIDSK** (p. **??**) files are not written to disk immediately after write calls. This method will ensure that any pending writes are flushed through to disk. This includes writing updates to tiled layer indexes, flushing out metadata, and potential caching of other information such as vector writes.

NOTE: Currently this method does not invalidate read-cached information such as segment pointer lists, segment headers, or band metadata. At some point in the future it might be extended to do this as well.

8.6.2.18 void PCIDSK::PCIDSKFile::UnlockBlock (bool *mark_dirty* = `false`) `[pure virtual]`

Unlock block. This method should be called after use of the buffer from **ReadAndLockBlock()** (p. **??**) is complete. If the buffer was modified and will need to be written to disk the argument "mark_dirty" should be passed in as true.

Note:

This method should only be called for **GetInterleaving()** (p. **??**) == "PIXEL" files.

Parameters:

mark_dirty true if the block data was modified, else false.

See also:

ReadAndLockBlock() (p. **??**)

8.6.2.19 void PCIDSK::PCIDSKFile::WriteToFile (const void ∗ *buffer*, uint64 *offset*, uint64 *size*) `[pure virtual]`

Write data to file. This method is normally only used by the **PCIDSK** (p. **??**) library itself, and provides a mechanism to write directly to the **PCIDSK** (p. **??**) file. Applications should normally use the **PCIDSK::PCIDSKChannel::ReadBlock()** (p. **??**), and **PCIDSK::PCIDSKChannel::WriteBlock()** (p. **??**) methods for imagery, or appropriate **PCIDSK::PCIDSKSegment** (p. **??**) methods for updating segment data.

Parameters:

buffer pointer to the data to write to disk.

offset the byte offset in the file (zero based) at which to write the data.

size the number of bytes from buffer to write.

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

- pcidsk_file.h
- pcidskfile.dox

8.7 PCIDSK::PCIDSKGeoref Class Reference

Interface to **PCIDSK** (p. **??**) georeferencing segment.

`#include <pcidsk_georef.h>`

Public Member Functions

- virtual void **GetTransform** (double &a1, double &a2, double &xrot, double &b1, double &yrot, double &b3)=0

  Get georeferencing transformation.

- virtual std::string **GetGeosys** ()=0

  Fetch georeferencing string.

- virtual std::vector< double > **GetParameters** ()=0

  Fetch projection parameters.

- virtual void **WriteSimple** (std::string geosys, double a1, double a2, double xrot, double b1, double yrot, double b3)=0

  Write simple georeferencing information.

- virtual void **WriteParameters** (std::vector< double > &parameters)=0

  Write complex projection parameters.

8.7.1 Detailed Description

Interface to **PCIDSK** (p. **??**) georeferencing segment.

8.7.2 Member Function Documentation

8.7.2.1 virtual std::string PCIDSK::PCIDSKGeoref::GetGeosys () `[pure virtual]`

Fetch georeferencing string. Returns the short, 16 character, georeferncing string. This string is sufficient to document the coordinate system of simple coordinate systems (like "UTM 17 S D000"), while other coordinate systems are only fully defined with additional projection parameters.

Returns:

the georeferencing string.

8.7.2.2 virtual std::vector<double> PCIDSK::PCIDSKGeoref::GetParameters () `[pure virtual]`

Fetch projection parameters. Fetches the list of detailed projection parameters used for projection methods not fully described by the Geosys string. The projection parameters are as shown below, though in the future more items might be added to the array. The first 15 are the classic USGS GCTP parameters.

- Parm[0]: diameter of earth - major axis (meters).
- Parm[1]: diameter of earth - minor axis (meters).
- Parm[2]: Reference Longitude (degrees)
- Parm[3]: Reference Latitude (degrees)
- Parm[4]: Standard Parallel 1 (degrees)
- Parm[5]: Standard Parallel 2 (degrees)
- Parm[6]: False Easting (meters?)
- Parm[7]: False Northing (meters?)
- Parm[8]: Scale (unitless)
- Parm[9]: Height (meters?)
- Parm[10]: Longitude 1 (degrees)
- Parm[11]: Latitude 1 (degrees)
- Parm[12]: Longitude 2 (degrees)
- Parm[13]: Latitude 2 (degrees)
- Parm[14]: Azimuth (degrees)
- Parm[15]: Landsat Number
- Parm[16]: Landsat Path
- Parm[17]: Unit Code (1=US Foot, 2=Meter, 4=Degree, 5=Intl Foot).

Review the **PCIDSK** (p. **??**) Database Reference Manual to understand which parameters apply to which projection methods.

Returns:

an array of values, at least 18.

8.7.2.3 virtual void PCIDSK::PCIDSKGeoref::GetTransform (double & *a1*, double & *a2*, double & *xrot*, double & *b1*, double & *yrot*, double & *b3*) `[pure virtual]`

Get georeferencing transformation. Returns the affine georeferencing transform coefficients for this image. Used to map from pixel/line coordinates to georeferenced coordinates using the transformation:

Xgeo = a1 + a2 ∗ Xpix + xrot ∗ Ypix

Ygeo = b1 + yrot ∗ Xpix + b2 ∗ Ypix

where Xpix and Ypix are pixel line locations with (0,0) being the top left corner of the top left pixel, and (0.5,0.5) being the center of the top left pixel. For an ungeoreferenced image the values will be (0.0,1.0,0.0,0.0,0.0,1.0).

Parameters:

a1 returns easting of top left corner.

a2 returns easting pixel size.

xrot returns rotational coefficient, normally zero.

b1 returns northing of the top left corner.

yrot returns rotational coefficient, normally zero.

b3 returns northing pixel size, normally negative indicating north-up.

8.7.2.4 virtual void PCIDSK::PCIDSKGeoref::WriteParameters (std::vector< double > & *parameters*) `[pure virtual]`

Write complex projection parameters. See **GetParameters()** (p. **??**) for the description of the parameters list.

Parameters:

parameters A list of at least 17 projection parameters.

8.7.2.5 virtual void PCIDSK::PCIDSKGeoref::WriteSimple (std::string *geosys*, double *a1*, double *a2*, double *xrot*, double *b1*, double *yrot*, double *b3*) `[pure virtual]`

Write simple georeferencing information. Writes out a georeferencing string and geotransform to the segment.

Parameters:

geosys 16 character coordinate system, like "UTM 17 S D000".

a1 easting of top left corner.

a2 easting pixel size.

xrot rotational coefficient, normally zero.

b1 northing of the top left corner.

yrot rotational coefficient, normally zero.

b3 northing pixel size, normally negative indicating north-up.

Referenced by PCIDSK::Create().

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

- pcidsk_georef.h

8.8 PCIDSK::PCIDSKInterfaces Class Reference

Collection of **PCIDSK** (p. **??**) hookable interfaces.

`#include <pcidsk_interfaces.h>`

Public Attributes

- const **IOInterfaces** ∗ **io**

  Pointer to IO Interfaces.

- **Mutex** ∗(∗ **CreateMutex**)(void)

  Function to create a mutex.

- void(∗ **JPEGDecompressBlock**)(uint8 ∗src_data, int src_bytes, uint8 ∗dst_data, int dst_bytes, int xsize, int ysize, **eChanType** pixel_type)

  Function to decompress a jpeg block.

- void(∗ **JPEGCompressBlock**)(uint8 ∗src_data, int src_bytes, uint8 ∗dst_data, int &dst_bytes, int xsize, int ysize, **eChanType** pixel_type, int quality)

  Function to compress a jpeg block.

8.8.1 Detailed Description

Collection of **PCIDSK** (p. **??**) hookable interfaces.

8.8.2 Member Data Documentation

8.8.2.1 void(∗ PCIDSKInterfaces::JPEGCompressBlock)(uint8 ∗src_data, int src_bytes, uint8 ∗dst_data, int &dst_bytes, int xsize, int ysize, eChanType pixel_type)

Function to compress a jpeg block. This function may be NULL if there is no jpeg interface available.

The default implementation is implemented using libjpeg.

The function encodes the image in src_data (src_bytes long) into dst_data as compressed jpeg data. The passed in value of dst_bytes is the size of the passed in dst_data array (it should be large enough to hold any compressed result0 and dst_bytes will be returned with the resulting actual number of bytes used.

Errors should be thrown as exceptions.

8.8.2.2 void(∗ PCIDSKInterfaces::JPEGDecompressBlock)(uint8 ∗src_data, int src_bytes, uint8 ∗dst_data, int dst_bytes, int xsize, int ysize, eChanType pixel_type)

Function to decompress a jpeg block. This function may be NULL if there is no jpeg interface available.

The default implementation is implemented using libjpeg.

The function decodes the jpeg compressed image in src_data (src_bytes long) into dst_data (dst_bytes long) as image data. The result should be exactly dst_bytes long, and will be an image of xsize x ysize of type pixel_type (currently on CHN_8U is allowed).

Errors should be thrown as exceptions.

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

- pcidsk_interfaces.h
- core/pcidskinterfaces.cpp

8.9 PCIDSK::PCIDSKSegment Class Reference

Public tnterface for the **PCIDSK** (p. **??**) Segment Type.

```
#include <pcidsk_segment.h>
```

Public Member Functions

- virtual void **WriteToFile** (const void ∗buffer, uint64 offset, uint64 size)=0

  Write data to segment.

- virtual void **ReadFromFile** (void ∗buffer, uint64 offset, uint64 size)=0

  Read data from segment.

- virtual **eSegType GetSegmentType** ()=0

  Fetch segment type.

- virtual std::string **GetName** ()=0

  Fetch segment name.

- virtual std::string **GetDescription** ()=0

  Fetch segment description.

- virtual int **GetSegmentNumber** ()=0

  Fetch segment number.

- virtual uint64 **GetContentSize** ()=0

  Get size of segment data.

- virtual bool **IsAtEOF** ()=0

  Is segment last in file?

- virtual std::string **GetMetadataValue** (std::string key)=0

  Fetch metadata value.

- virtual void **SetMetadataValue** (std::string key, std::string value)=0

  Set metadata value.

- virtual std::vector< std::string > **GetMetadataKeys** ()=0

  Fetch metadata keys.

- virtual void **Synchronize** ()=0

  Write pending information to disk.

8.9.1 Detailed Description

Public tnterface for the **PCIDSK** (p. **??**) Segment Type. This class interface is used for access to **PCIDSK** (p. **??**) segments and associated data. The class should never be instantiated by the application. Instead all instances are owned by the corresponding **PCIDSK::PCIDSKFile** (p. **??**) object and a pointer can be fetched using **PCIDSKFile::GetSegment()** (p. **??**) or related methods.

Some segments types such as binary (SEG_BIN) provide no custom interfaces and can only be accessed using the generic **PCIDSKSegment** (p. **??**) methods. Others, such as georeferencing segments (SEG_-GEO) offer additional segment specific interfaces via multiple inheritance. Use dynamic casts to get access to the type specific interfaces.

Example:

```
PCIDSK::PCIDSKSegment *seg = file->GetSegment(1);

if( seg->GetSegmentType() == PCIDSK::SEG_GEO )
{
    PCIDSK::PCIDSKGeoref *georef = dynamic_cast<PCIDSK::PCIDSKGeoref*>( seg );

    printf( "Geosys = %s\n", georef->GetGeosys() );
}
```

8.9.2 Member Function Documentation

8.9.2.1 uint64 PCIDSK::PCIDSKSegment::GetContentSize () `[pure virtual]`

Get size of segment data. Returns the size of the data portion of this segment (header excluded) in bytes.

Returns:

segment data size in bytes.

8.9.2.2 const char * PCIDSK::PCIDSKSegment::GetDescription () `[pure virtual]`

Fetch segment description. The returned pointer is to internally managed data of the **PCIDSKSegment** (p. **??**), and should not be modified, freed, or used after the segment object ceases to exist. The description is at most 80 characters long.

Returns:

the segment description.

8.9.2.3 std::vector< std::string > PCIDSK::PCIDSKSegment::GetMetadataKeys () `[pure virtual]`

Fetch metadata keys. Returns a vector of metadata keys that occur on this object. The values associated with each may be fetched with **GetMetadataValue()** (p. **??**).

Returns:

list of keys

See also:

GetMetadataValue() (p. **??**)

8.9.2.4 std::string PCIDSK::PCIDSKSegment::GetMetadataValue (std::string *key*) `[pure virtual]`

Fetch metadata value. Note that the returned pointer is to an internal structure and it may become invalid if another thread modifies the metadata for this object.

Parameters:

key the key to fetch the value for.

Returns:

the value of the indicated metadata item, or NULL if it does not exist on the target object.

See also:

GetMetadataKeys() (p. **??**)

8.9.2.5 const char ∗ PCIDSK::PCIDSKSegment::GetName () `[pure virtual]`

Fetch segment name. The returned pointer is to internally managed data of the **PCIDSKSegment** (p. **??**), and should not be modified, freed, or used after the segment object ceases to exist. The name is at most eight characters long.

Returns:

the segment name.

8.9.2.6 int PCIDSK::PCIDSKSegment::GetSegmentNumber () `[pure virtual]`

Fetch segment number.

Returns:

the segment number (1+).

8.9.2.7 eSegType PCIDSK::PCIDSKSegment::GetSegmentType () `[pure virtual]`

Fetch segment type.

Returns:

the type of this segment.

8.9.2.8 bool PCIDSK::PCIDSKSegment::IsAtEOF () `[pure virtual]`

Is segment last in file? Returns true if the segment is the last one in the file, and thus can be grown without having to move it. Primarily this method is used by the SDK itself.

Returns:

true if segment at EOF or false otherwise.

8.9.2.9 void PCIDSK::PCIDSKSegment::ReadFromFile (void ∗ *buffer*, uint64 *offset*, uint64 *size*) `[pure virtual]`

Read data from segment. Read from data area of this segment. Offset zero refers to the start of the data area of the segment, access to to the segment header is not available via **ReadFromFile()** (p. **??**).

Parameters:

buffer pointer to the buffer into which the data should be read.

offset the byte offset in the file (zero based) at which to read the data.

size the number of bytes from the file to read.

8.9.2.10 void PCIDSK::PCIDSKSegment::SetMetadataValue (std::string *key*, std::string *value*) `[pure virtual]`

Set metadata value. Assign the metadata value associated with the passed key on this object. The file needs to be open for update. Note that keys should be well formed tokens (no special characters, spaces, etc).

Parameters:

key the key to fetch the value for.

value the value to assign to the key. An empty string deletes the item.

See also:

GetMetadataValue() (p. **??**)

8.9.2.11 void PCIDSK::PCIDSKSegment::Synchronize () `[pure virtual]`

Write pending information to disk. Some write and update operations on **PCIDSK** (p. **??**) files are not written to disk immediately after write calls. This method will ensure that any pending writes are flushed through to disk.

NOTE: Currently this method does not invalidate read-cached information. At some point in the future it might be extended to do this as well.

8.9.2.12 void PCIDSK::PCIDSKSegment::WriteToFile (const void ∗ *buffer*, uint64 *offset*, uint64 *size*) `[pure virtual]`

Write data to segment. Write to data area of this segment. Offset zero refers to the start of the data area of the segment, access to to the segment header is not available via **WriteToFile()** (p. **??**).

Parameters:

buffer pointer to the data to write to disk.

offset the byte offset in the file (zero based) at which to write the data.

size the number of bytes from buffer to write.

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

- pcidsk_segment.h
- pcidsksegment.dox

8.10 PCIDSK::PCIDSKVectorSegment Class Reference

Interface to **PCIDSK** (p. **??**) vector segment.

`#include <pcidsk_vectorsegment.h>`

Public Member Functions

- virtual std::string **GetRst** ()=0

  Fetch RST.

- virtual int **GetFieldCount** ()=0

  Get field count.

- virtual std::string **GetFieldName** (int field_index)=0

  Get field name.

- virtual std::string **GetFieldDescription** (int field_index)=0

  Get field description.

- virtual **ShapeFieldType GetFieldType** (int field_index)=0

  Get field type.

- virtual std::string **GetFieldFormat** (int field_index)=0

  Get field format.

- virtual **ShapeField GetFieldDefault** (int field_index)=0

  Get field default.

- virtual **ShapeIterator begin** ()=0

  Get iterator to first shape.

- virtual **ShapeIterator end** ()=0

  Get iterator to end of shape lib (a wrapper for NullShapeId).

- virtual **ShapeId FindFirst** ()=0

  Fetch first shapeid in the layer.

- virtual **ShapeId FindNext** (**ShapeId** id)=0

  Fetch the next shape id after the indicated shape id.

- virtual void **GetVertices** (**ShapeId** id, std::vector< **ShapeVertex** > &list)=0

  Fetch the vertices for the indicated shape.

- virtual void **GetFields** (**ShapeId** id, std::vector< **ShapeField** > &list)=0

  Fetch the fields for the indicated shape.

8.10.1 Detailed Description

Interface to **PCIDSK** (p. **??**) vector segment. The vector segment contains a set of vector features with a common set of attribute data (fields). Each feature has a numeric identifier (ShapeId), a set of field values, and a set of geometric vertices. The layer as a whole has a description of the attribute fields, and an RST (Representation Style Table).

The geometry and attribute fields of shapes can be fetched with the **GetVertices()** (p. **??**) and **GetFields()** (p. **??**) methods by giving the ShapeId of the desired feature. The set of shapeid's can be identified using the **FindFirst()** (p. **??**), and **FindNext()** (p. **??**) methods or the STL compatible **ShapeIterator** (p. **??**) (**begin()** (p. **??**) and **end()** (p. **??**) methods).

The **PCIDSKSegment** (p. **??**) interface for the segment can be used to fetch the LAYER_TYPE metadata describing how the vertices should be interpreted as a geometry. Some layers will also have a RingStart attribute field which is used in conjunction with the LAYER_TYPE to interprete the geometry. Some vector segments may have no LAYER_TYPE metadata in which case single vertices are interpreted as points, and multiple vertices as linestrings.

More details are available in the GDB.HLP description of the GDB vector data model.

Note that there are no mechanisms for fast spatial or attribute searches in a **PCIDSK** (p. **??**) vector segment. Accessing features randomly (rather than in the order shapeids are returned by **FindFirst()** (p. **??**)/FindNext() or **ShapeIterator** (p. **??**)) may result in reduced performance, and the use of large amounts of memory for large vector segments.

8.10.2 Member Function Documentation

8.10.2.1 virtual ShapeIterator PCIDSK::PCIDSKVectorSegment::begin () `[pure virtual]`

Get iterator to first shape.

Returns:

iterator.

8.10.2.2 virtual ShapeIterator PCIDSK::PCIDSKVectorSegment::end () `[pure virtual]`

Get iterator to end of shape lib (a wrapper for NullShapeId).

Returns:

iterator.

8.10.2.3 virtual ShapeId PCIDSK::PCIDSKVectorSegment::FindFirst () `[pure virtual]`

Fetch first shapeid in the layer.

Returns:

first shape's shapeid.

8.10.2.4 virtual ShapeId PCIDSK::PCIDSKVectorSegment::FindNext (ShapeId *id*) `[pure virtual]`

Fetch the next shape id after the indicated shape id.

Parameters:

id the previous shapes id.

Returns:

next shape's shapeid.

8.10.2.5 virtual int PCIDSK::PCIDSKVectorSegment::GetFieldCount () `[pure virtual]`

Get field count. Note that this includes any system attributes, like RingStart, that would not normally be shown to the user.

Returns:

the number of attribute fields defined on this layer.

8.10.2.6 virtual ShapeField PCIDSK::PCIDSKVectorSegment::GetFieldDefault (int *field_index*) `[pure virtual]`

Get field default.

Parameters:

field_index index of the field requested from zero to **GetFieldCount()** (p. **??**)-1.

Returns:

the field default value.

8.10.2.7 virtual std::string PCIDSK::PCIDSKVectorSegment::GetFieldDescription (int *field_index*) `[pure virtual]`

Get field description.

Parameters:

field_index index of the field requested from zero to **GetFieldCount()** (p. **??**)-1.

Returns:

the field description, often empty.

8.10.2.8 virtual std::string PCIDSK::PCIDSKVectorSegment::GetFieldFormat (int *field_index*) `[pure virtual]`

Get field format.

Parameters:

field_index index of the field requested from zero to **GetFieldCount()** (p. **??**)-1.

Returns:

the field format as a C style format string suitable for use with printf.

8.10.2.9 virtual std::string PCIDSK::PCIDSKVectorSegment::GetFieldName (int *field_index*) `[pure virtual]`

Get field name.

Parameters:

field_index index of the field requested from zero to **GetFieldCount()** (p. **??**)-1.

Returns:

the field name.

8.10.2.10 virtual void PCIDSK::PCIDSKVectorSegment::GetFields (ShapeId *id*, std::vector< ShapeField > & *list*) `[pure virtual]`

Fetch the fields for the indicated shape.

Parameters:

id the shape to fetch

list the field list is updated with the field values for this shape.

8.10.2.11 virtual ShapeFieldType PCIDSK::PCIDSKVectorSegment::GetFieldType (int *field_index*) `[pure virtual]`

Get field type.

Parameters:

field_index index of the field requested from zero to **GetFieldCount()** (p. **??**)-1.

Returns:

the field type.

8.10.2.12 virtual std::string PCIDSK::PCIDSKVectorSegment::GetRst () `[pure virtual]`

Fetch RST. No attempt is made to parse the RST, it is up to the caller to decode it.

NOTE: There is some header info on RST format that may be needed to do this for older RSTs.

Returns:

RST as a string.

8.10.2.13 virtual void PCIDSK::PCIDSKVectorSegment::GetVertices (ShapeId *id*, std::vector< ShapeVertex > & *list*) `[pure virtual]`

Fetch the vertices for the indicated shape.

Parameters:

id the shape to fetch

list the list is updated with the vertices for this shape.

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

- pcidsk_vectorsegment.h

8.11 PCIDSK::ShapeField Class Reference

Attribute field value.

```
#include <pcidsk_shape.h>
```

Public Member Functions

- **ShapeField** ()

  Simple constructor.

- **ShapeField** (const **ShapeField** &src)

  Copy constructor.

- **ShapeField** & **operator=** (const **ShapeField** &src)

  Assignment operator.

- void **Clear** ()

  Clear field value.

- **ShapeFieldType** **GetType** () const

  Fetch field type.

- void **SetValue** (int32 val)

  Set integer value on field.

- void **SetValue** (const std::vector< int32 > &val)

  Set integer list value on field.

- void **SetValue** (const std::string &val)

  Set string value on field.

- void **SetValue** (double val)

  Set double precision floating point value on field.

- void **SetValue** (float val)

  Set single precision floating point value on field.

- int32 **GetValueInteger** () const

  Fetch value as integer or zero if field not of appropriate type.

- std::vector< int32 > **GetValueCountedInt** () const

  Fetch value as integer list or empty list if field not of appropriate type.

- std::string **GetValueString** () const

  Fetch value as string or "" if field not of appropriate type.

- float **GetValueFloat** () const

  Fetch value as float or 0.0 if field not of appropriate type.

- double **GetValueDouble** () const

  Fetch value as double or 0.0 if field not of appropriate type.

8.11.1 Detailed Description

Attribute field value. This class encapsulates any of the supported vector attribute field types in a convenient way that avoids memory leaks or ownership confusion. The object has a field type (initially FieldTypeNone on construction) and a value of the specified type. Note that the appropriate value accessor (ie. **GetValueInteger()** (p. **??**)) must be used that corresponds to the fields type. No attempt is made to automatically convert (ie. float to double) if the wrong accessor is used.

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

- pcidsk_shape.h

8.12 PCIDSK::ShapeIterator Class Reference

Iterator over shapeids in a vector segment.

```
#include <pcidsk_vectorsegment.h>
```

Public Member Functions

- **ShapeIterator** (**PCIDSKVectorSegment** ∗seg_in)
- **ShapeIterator** (**PCIDSKVectorSegment** ∗seg_in, **ShapeId** id_in)
- **ShapeIterator** (const **ShapeIterator** &mit)
- **ShapeIterator** & **operator++** ()
- **ShapeIterator** & **operator++** (int)
- bool **operator==** (const **ShapeIterator** &rhs)
- bool **operator!=** (const **ShapeIterator** &rhs)
- **ShapeId** & **operator∗** ()

8.12.1 Detailed Description

Iterator over shapeids in a vector segment.

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

- pcidsk_vectorsegment.h

8.13 PCIDSK::ShapeVertex Struct Reference

Structure for an x,y,z point.

```
#include <pcidsk_shape.h>
```

Public Attributes

- double **x**
- double **y**
- double **z**

8.13.1 Detailed Description

Structure for an x,y,z point.

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

- pcidsk_shape.h

Chapter 9

File Documentation

9.1 pcidsk.h File Reference

```
#include "pcidsk_config.h"
#include "pcidsk_types.h"
#include "pcidsk_file.h"
#include "pcidsk_channel.h"
#include "pcidsk_buffer.h"
#include "pcidsk_mutex.h"
#include "pcidsk_exception.h"
#include "pcidsk_interfaces.h"
#include "pcidsk_segment.h"
#include "pcidsk_io.h"
#include "pcidsk_georef.h"
#include "pcidsk_rpc.h"
```

Namespaces

- namespace **PCIDSK**

  *Namespace for all **PCIDSK** (p. **??**) Library classes and functions.*

Functions

- **PCIDSKFile** ∗ **PCIDSK::Open** (std::string filename, std::string access, const **PCIDSKInterfaces** ∗interfaces)
- **PCIDSKFile** ∗ **PCIDSK::Create** (std::string filename, int pixels, int lines, int channel_count, **eChanType** ∗channel_types, std::string options, const **PCIDSKInterfaces** ∗interfaces)

9.1.1 Detailed Description

Public **PCIDSK** (p. **??**) library classes and functions.