# Christine-GIS

# Public interface for vector data plug-ins

Specification of public interface version 1.1

Christine-GIS

| | |
|---|---|
| Released: | March 2009 |
| Revised: | April 2021 |

# Content

# 1. Introduction

Document describes interface for dynamic linked libraries, which can be used for reading and drawing spatial vector data. The interface allows you create your own library and in this way you can extent Christine's list of known vector formats. If you create your own library, it is strictly recommended to create an installation program for installing the library. See *Installation of your own library* chapter for some hints.
Document describes version 1.1 of the interface. The interface is supported by Christine-GIS version 5.1 and higher. Christine-GIS Viewer does not support this interface. The material in this document reflects the information available at the time of publication and is essentially identical to the information contained in the Christine's help file. In some cases, more up-to-date information may be available at Christine's web site (www.christine-gis.com).

# 2. Used data types

Most of used data types are common data types used by Microsoft Windows API.

**BOOL** - A boolean type (should be TRUE or FALSE), stored as signed integer

**BYTE*** - A pointer to array of bytes

**COLORREF** - A 32-bit value used to specify a RGB color. The low-order byte contains a value for the relative intensity of red; the second byte contains a value for green; and the third byte contains a value for blue. The high-order byte must be zero. The maximum value for a single byte is 0xFF.

**CRS2CRS** – A pointer to function for conversion between coordinate systems. Pointer to the function is declared in following way:

```
typedef BOOL (__cdecl *CRS2CRS)(int, int, double*, double*, double*)
```

Function returns TRUE if conversion is successful done, otherwise returns FALSE. First parameter of the function is count of points for conversion. Second parameter is offset between coordinates in array of coordinates. Third, fourth and fifth parameters are pointers to arrays of x, y and z coordinates.

**double** - A 64-bit signed floating point number

**double*** - A pointer to 64-bit signed floating point number

**DWORD** - A 32-bit unsigned integer

**EXTENT** - Structure that defines ranges of X, Y, Z coordinates and M values

```
typedef struct _EXTENT {
    double xMin;       // specifies minimum value of X coordinate
    double yMin;       // specifies minimum value of Y coordinate
    double xMax;       // specifies maximum value of X coordinate
    double yMax;       // specifies maximum value of Y coordinate
    double zMin;       // specifies minimum value of Z coordinate (ignored)
    double zMax;       // specifies maximum value of Z coordinate (ignored)
    double mMin;       // specifies minimum of M value (ignored)
    double mMax;       // specifies maximum of M value (ignored)
} EXTENT;
```

**EXTENT*** - A pointer to extent structure

**FALSE** - Boolean value, stored as signed integer with zero value

**HDC** - Handle to a device context

**HWND** - Handle to a window

**int** - A 32-bit signed integer

**LONG** - A 32-bit signed integer

**NULL** - Zero value

**RECT** - Structure that defines the coordinates of the upper-left and lower-right corners of a rectangle

```
typedef struct _RECT {
    LONG left;     // x-coordinate of the upper-left corner of the rectangle
    LONG top;      // y-coordinate of the upper-left corner of the rectangle
    LONG right;    // x-coordinate of the lower-right corner of the rectangle
    LONG bottom;   // y-coordinate of the lower-right corner of the rectangle
} RECT;
```

**RECT*** - A pointer to rectangle structure

**TCHAR*** - A pointer to array of chars (string)

**TRUE** - Boolean value, stored as non zero signed integer

**UINT** - A 32-bit unsigned integer

**VECTORFORMATSTRUCT** - structure for saving necessary informations for communication between core of Christine and dynamic linked library

```
typedef struct _VECTORFORMATSTRUCT {
    // mandatory part of structure
    DWORD    dwSize;    // size of this structure (including optional part)
    COLORREF mapBkColor;    // background color of map
    // optional part of structure
    // content of optional part depends on implementation of plug-in module
} VECTORFORMATSTRUCT;
```

**void** - Used as a function‘s return type, specifies that the function does not return a value

**void*** - A pointer to unspecified data type or structure ("universal pointer")

# 3. How it works

Christine calls functions of the public interface described in chapter *Public interface of library* when it needs. Set of called functions depends on type of operation. Library is loaded before the operation starts and it is released after operation is done. This chapter describes all supported operations.

## 3.1 Adding vector data to a view

Christine looks through Windows registry for available vector formats. Christine uses values of *FormatName*, *FormatExtensions* and *FormatID* keys in order to complete list of known formats and fills combo box in open dialog. If user chooses a vector format that uses the interface, Christine will look for value of *LibFileName* key in Windows registry in order to be able to load the library. After the library is loaded Christine calls Initialize function. The Initialize function returns pointer to VECTORFORMATSTRUCT structure. You can use the structure for storing pointers to your own objects, memory blocks, temporary files, etc. Christine allocates memory and stores the structure. The structure is used by others functions, so you can use the structure for sharing any data you need among functions.

**List of called functions during the operation:**

```
void* Initialize(TCHAR* fileName)
EXTENT* GetExtent(void* pStruct)
```

## 3.2 Removing vector data from a view

When a theme that uses this interface is about to remove from a view Christine calls Release function. In the Release function you can destroy all object you created, clean up memory, delete temporary files, etc.

**List of called functions during the operation:**

```
void Release(void* pStruct)
```

## 3.3 Drawing and printing vector data

Christine loads library to memory and fills mapBkColor member in VECTORFORMATSTRUCT structure. After this it calls Draw function. When drawing operation is done, it releases the library.

**List of called functions during the operation:**

```
void Draw(HDC hDC, EXTENT* drawExtent, RECT* clipRect, int unitSize, void*
pStruct, CRS2CRS Crs2Crs, DWORD* pId1, volatile DWORD* pId2)
```

## 3.4 Showing properties of theme based on vector data

Before properties dialog is shown Christine calls set of functions in order to get all necessary informations.

**List of called functions during the operation:**

```
EXTENT* GetExtent(void* pStruct)
TCHAR* GetFormatName(void* pStruct)
```

```
TCHAR* GetFormatVersion(void* pStruct)
DWORD GetInterfaceVersion()
BOOL HasSpatialIndex(void* pStruct)
BOOL SupportsSpatialIndex()
```

## 3.5 Setting properties of theme based on vector data

When properties dialog is shown, user can press Change Settings button in Settings page. After this Christine calls set of functions in order to allow user to change settings of display vector file in your own dialog.

**List of called functions before dialog is shown:**

```
TCHAR* GetFormatName(void* pStruct)
void* ChangeSettings(HWND hWndParent, void* pStruct)
```

**List of called functions after new settings is confirmed:**

```
BOOL HasSpatialIndex(void* pStruct)
DWORD GetInterfaceVersion()
BOOL SupportsReloading()
BOOL NeedsReloading(void* pStruct)
BOOL Reload(void* pStruct)
```

## 3.6 Converting vector data

Christine looks for value of *LibFileName* key in Windows registry and loads dynamic-linked library. After this it calls SupportsConversionVectorFormat2Shp or SupportConversionShp2VectorFormat depend on direction of conversion and releases library. If the function returns TRUE, Christine will use values of *FormatName*, *FormatExtensions* and *FormatID* keys in order to add the format into conversion dialog. If user chooses a vector format that uses the interface, Christine will look for value of *LibFileName* key in Windows registry again, loads library and calls ConvertVectorFormat2Shp or ConvertShp2VectorFormat function. After the conversion process is done, Christine will release library.

**List of called functions during the operation:**

```
BOOL ConvertVectorFormat2Shp(TCHAR* shpFileName, TCHAR* vectorFormatFileName,
HWND hWndMainFrame, void* pStruct, CRS2CRS Crs2Crs, BOOL bUseDefaultSettings)
BOOL ConvertShp2VectorFormat(TCHAR* shpFileName, TCHAR* vectorFormatFileName,
HWND hWndMainFrame, CRS2CRS Crs2Crs, BOOL bUseDefaultSettings, BYTE* selection)
DWORD GetInterfaceVersion()
BOOL SupportsConversionVectorFormat2Shp()
BOOL SupportsConversionShp2VectorFormat()
```

## 3.7 Building or removing spatial index

Christine looks into Windows registry for value of *LibFileName* key and loads dynamic-linked library. After this it calls SupportsSpatialIndex function. If the function returns TRUE, Christine calls BuildSpatialIndex or RemoveSpatialIndex function. After action is done, Christine will release library.

**List of called functions during the operation:**

```
void BuildSpatialIndex(void* pStruct)
void RemoveSpatialIndex(void* pStruct)
```

```
BOOL SupportsSpatialIndex()
```

## 3.8 Saving properties of theme based on a vector data to a project file

Christine looks into Windows registry for value of *LibFileName* key, loads library to memory and calls WriteSettingsToStringLine function. Function WriteSettingsToStringLine uses vector format structure in order to create string line with settings of theme based on a vector data. Content of the string line depends on your needs. Function WriteSettingsToStringLine returns pointer to the string line. The string line must be terminated by binary zero (it must be a null-terminated string). Christine saves the string line into project file as value of key called *vectorThemeSettings*.

**List of called functions during the operation:**

```
TCHAR* WriteSettingsToStringLine(void* pStruct)
```

## 3.9 Reading properties of theme based on a vector data from a project file

Christine looks into Windows registry for value of *LibFileName* key, loads library to memory and calls ReadSettingsFromStringLine function. Function ReadSettingsFromStringLine uses a string line with saved settings in order to set vector format settings structure. Content of the string line was created by WriteSettingsToStringLine function (see 3.8 chapter). The string line is terminated by binary zero (it is a null-terminated string). Function ReadSettingsFromStringLine will return pointer to changed settings structure or NULL if failed.

**List of called functions during the operation:**

```
void* ReadSettingsFromStringLine(TCHAR* settingsStr, void* pStruct)
```

# 4. Public interface of library

This chapter describes public interface of the library. Each function in the public interface has exactly defined behaviour and it is called in situations described in chapter *How it works*.

`void` **`BuildSpatialIndex`**`(void* pStruct)`
Builds spatial index. Parameter pStruct is a pointer to structure born during initialization process, see chapter 3.1 for details.

`void*` **`ChangeSettings`**`(HWND hWndParent, void* pStruct)`
Shows dialog with settings for vector file. Parameter hWndParent is handle of parent window for the dialog. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details. Function returns pointer to modified structure or NULL if no changes was made in the structure or user press Cancel button.

`BOOL` **`ConvertVectorFormat2Shp`**`(TCHAR* shpFileName, TCHAR* vectorFormatFileName, HWND hWndMainFrame, void* pStruct, CRS2CRS Crs2Crs, BOOL bUseDefaultSettings)`
Converts vector data to ESRI shapefile and saves it to a file. Parameter shpFileName is full file name of the shapefile (it is in fact output file). Parameter vectorFormatFileName is full file name of vector format file (it is in fact input file). Parameter hWndMainFrame is handle to Christine's main window. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details. Parameter Crs2Crs is pointer to function for conversion between coordinate systems. If no conversion is needed parameter Crs2Crs is NULL. Parameter bUseDefaultSetting indicates if you should use default settings for conversion or if you should display a dialog for modifying settings of conversion. Function returns TRUE if shapefile was written successfully, otherwise returns FALSE.

`BOOL` **`ConvertShp2VectorFormat`**`(TCHAR* shpFileName, TCHAR* vectorFormatFileName, HWND hWndMainFrame, CRS2CRS Crs2Crs, BOOL bUseDefaultSettings, BYTE* selection)`
Converts ESRI shapefile to vector data and saves it to a file. Parameter shpFileName is full file name of the shapefile (it is in fact input file). Parameter vectorFormatFileName is full file name of vector format file (it is in fact output file). Parameter hWndMainFrame is handle to Christine's main window. Parameter Crs2Crs is pointer to function for conversion between coordinate systems. If no conversion is needed parameter is NULL. Parameter bUseDefaultSetting indicates if you should use default settings for conversion or if you should display a dialog for modifying settings of conversion. Parameter selection is pointer to array of bytes indicating selection (nonzero value means that apropriate record is selected). You can use this array for converting only selected records. If all records needs to be converted, parameter selection is NULL. Returns TRUE if vector format file was written successfully, otherwise returns FALSE.

`void` **`Draw`**`(HDC hDC, EXTENT* drawExtent, RECT* clipRect, int unitSize, void* pStruct, CRS2CRS Crs2Crs, DWORD* pId1, volatile DWORD* pId2)`
Draws vector data on device context with handle hDC. Parameter drawExtent is extent for drawing in map units. Parameter clipRect is clipping rectangle in screen units. Parameter unitSize is size of one unit in pixels. You can use value of this parameter for designing size or width of symbols for drawing a feature. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details. Parameter Crs2Crs is pointer to function for conversion between coordinate systems. If no conversion is needed parameter is NULL. Parameters pId1 and pId2 are pointers to numeric identificators. If the values of identificators are the same you can continue drawing, but if pId2 is changed you should stop drawing immediately and clean up the memory.

`EXTENT*` **`GetExtent`**`(void* pStruct)`
Returns pointer to raw (unprojected) extent of vector data. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details.

`TCHAR*` **`GetFormatName`**`(void* pStruct)`
Returns pointer to string with name of vector format. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details.

`TCHAR*` **`GetFormatVersion`**`(void* pStruct)`
Returns pointer to string with version of vector data format. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details.

`DWORD` **`GetInterfaceVersion`**`()`
Returns version of the interface. In high word is major version (1) and in low word is minor version (1).

`BOOL` **`HasSpatialIndex`**`(void* pStruct)`
Returns TRUE if vector data have built spatial index. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details.

`void*` **`Initialize`**`(TCHAR* fileName)`
Returns pointer to structure VECTORFORMATSTRUCT. If function failed it should return NULL. Christine will save contents of the structure and uses it as parameter when calling other functions of this interface. Parameter fileName is full file name of vector data file.

`BOOL` **`NeedsReloading`**`(void* pStruct)`
Returns TRUE if vector data have to be reloaded. It is useful when you must reflect new settings of loaded data. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 and 3.5 for details.

`void*` **`ReadSettingsFromStringLine`**`(TCHAR* settingsStr, void* pStruct)`
Reads settings from string line settingsStr when a project file is openning. The string line is terminated by binary zero. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details. Function returns pointer to changed settings structure or NULL if failed.

`void` **`Release`**`(void* pStruct)`
Destroys all object you created, cleans up memory, deletes temporary files, etc. Parameter pStruct is a pointer to structure saved during initialization process, see chapters 3.1 and 3.2 for details.

`BOOL` **`Reload`**`(void* pStruct)`
Reloads vector data. Parameter pStruct is a pointer to structure saved during initialization process, see chapters 3.1 and 3.5 for details.

`void` **`RemoveSpatialIndex`**`(void* pStruct)`
Removes spatial index. Parameter pStruct is a pointer to structure saved during initialization process, see chapter 3.1 for details.

`BOOL` **`SupportsConversionVectorFormat2Shp`**`()`
Returns TRUE if the library supports conversion from vector format to shapefile, otherwise returns FALSE.

`BOOL` **`SupportsConversionShp2VectorFormat`**`()`
Returns TRUE if the library supports conversion from shapefile to vector format, otherwise returns FALSE.

`BOOL` **`SupportsReloading`**`()`
Returns TRUE if the library supports reloading of vector data, otherwise returns FALSE.

`BOOL` **`SupportsSpatialIndex`**`()`
Returns TRUE if the library supports spatial indexing, otherwise returns FALSE.

`TCHAR*` **`WriteSettingsToStringLine`**`(void* pStruct)`
Returns pointer to string line with settings. The string line is written into Christine's project file when you save project.

## 4.1 Minimal functional requirements

Minimal requirement is to draw vector data in a view and to print the vector data to printer. To create functional library that do this you must implement following functions with described behaviour:

`void` **`BuildSpatialIndex`**`(void* pStruct)`

Does nothing.

`void* **ChangeSettings**(HWND hWndParent, void* pStruct)`
Shows dialog with information message „There are no adjustable settings“ and returns NULL.

`BOOL **ConvertVectorFormat2Shp**(TCHAR* shpFileName, TCHAR* vectorFormatFileName, HWND hWndMainFrame, void* pStruct, CRS2CRS Crs2Crs, BOOL bUseDefaultSettings)`
Returns FALSE.

`BOOL **ConvertShp2VectorFormat**(TCHAR* shpFileName, TCHAR* vectorFormatFileName, HWND hWndMainFrame, CRS2CRS Crs2Crs, BOOL bUseDefaultSettings, BYTE* selection)`
Returns FALSE.

`void **Draw**(HDC hDC, EXTENT* drawExtent, RECT* clipRect, int unitSize, void* pStruct, CRS2CRS Crs2Crs, DWORD* pId1, volatile DWORD* pId2)`
Draws vector data on device context with handle hDC.

`EXTENT* **GetExtent**(void* pStruct)`
Returns pointer to extent of vector data.

`TCHAR* **GetFormatName**(void* pStruct)`
Returns pointer to string with name of data format.

`TCHAR* **GetFormatVersion**(void* pStruct)`
Returns version of vector data format.

`DWORD **GetInterfaceVersion**()`
Returns version of the interface. In high word is major version (1) and in low word is minor version (1).

`BOOL **HasSpatialIndex**(void* pStruct)`
Returns FALSE.

`void* **Initialize**(TCHAR* fileName)`
Creates VECTORFORMATSTRUCT object and returns pointer to it.

`BOOL **NeedsReloading**(void* pStruct)`
Returns FALSE.

`void* **ReadSettingsFromStringLine**(TCHAR* settingsStr, void* pStruct)`
Returns pointer to VECTORFORMATSTRUCT object.

`void **Release**(void* pStruct)`
Does nothing.

`BOOL **Reload**(void* pStruct)`
Returns FALSE.

`void **RemoveSpatialIndex**(void* pStruct)`
Does nothing.

`BOOL **SupportsConversionVectorFormat2Shp**()`
Returns FALSE.

`BOOL **SupportsConversionShp2VectorFormat**()`
Returns FALSE.

`BOOL **SupportsReloading**()`
Returns FALSE.

`BOOL **SupportsSpatialIndex**()`
Returns FALSE.

```
TCHAR* WriteSettingsToStringLine(void* pStruct)
```
Returns pointer to string with zero length.

# 5. Installation of your own library

It is strictly recommended to create an installation program for installing the library. Installation program must place library file to subfolder dlls in Christine's installation folder. Installation program must also write following informations to Windows registry:

**Key:**
HKEY_LOCAL_MACHINE\Software\Christine-GIS\5.x\VectorFormats\[ID of format]

**Values:**

| Name | Value | Type | Max. length (byte) | |
|---|---|---|---|---|
| LibFileName | [library.dll] | REG_SZ | 255 | Mandatory |
| FormatExtensions | [*.xxx;*.xxxx] | REG_SZ | 63 | Mandatory |
| FormatName | Name of Vector Format | REG_SZ | 255 | Mandatory |
| FormatID | [ID of format] | REG_DWORD | 4 | Mandatory |

**Example for Drawing Interchange File Format:**

**Key:**
HKEY_LOCAL_MACHINE\Software\Christine-GIS\5.x\VectorFormats\50

**Values:**

| Name | Value |
|---|---|
| LibFileName | dxf.dll |
| FormatExtensions | *.dxf |
| FormatName | Drawing Interchange File Format |
| FormatID | 50 |

Note: Value *FormatID* must be the same as name of the key.

## 5.1 Format identification number (ID)

If you want to create your own library for reading vector data, look at Christine's web site for free format IDs. After you choose a free format ID, contact us and we will publish your choosen ID on Christine's web site to avoid problems with duplicity. Format ID is used internally by Christine and by these metods in Christine's script language: GetType(Number nType) from class Theme, ExportToBmp(String sFullFileName, Number nType, Rect extent, Number nWidth, Number nHeight, Bool bCreateWorldFile) and AddTheme(String sFullFileName, Number nType) from class View, ExportToVectorFormat(String sFullFileName, Number nFormatID, Bool bUseDefaultSettings) from class FTheme, Export(String sFullFileName, Number nOutputType) from class DSCTheme, ShowBmp(String sFullFileName, Number nDelay, Bool bFrame, Number nType) from class MsgBox.

**Recommended ranges of format IDs are following:**

**0 - 1000** for raster and vector data based on files (typically stored on a file system), range 0 - 100 is reserved for Christine-GIS, 101 - 1000 is available for third-party developers

**1001 - 2000** for raster and vector data based on a web service (typically generated by a map server and sent to Christine), range 1001 - 1100 is reserved for Christine-GIS, 1101 - 2000 is available for third-party developers

**2001 - 3000** for raster and vector data stored in a geodatabase (for future use, not supported yet), range 2001 - 2100 is reserved for Christine-GIS, 2101 - 3000 is available for third-party developers

# 6. Libraries that come with Christine

Following libraries come with Christine-GIS version 5.1, that have implemented this interface. The libraries are placed in subfolder dlls in Christine's installation folder.

| | |
|---|---|
| Format name: | **Drawing Interchange File Format** |
| Library file name: | dxf.dll |
| Format ID: | 50 |
| Format extensions: | *.dxf |
| Version of interface: | 1.1 |
| Supports conversion to shapefile: | Yes |
| Supports conversion to vector format: | Yes |
| Supports reloading: | No |
| Supports spatial indexing: | Yes |

| | |
|---|---|
| Format name: | **Keyhole Markup Language** |
| Library file name: | kml.dll |
| Format ID: | 51 |
| Format extensions: | *.kml, *.kmz |
| Version of interface: | 1.1 |
| Supports conversion to shapefile: | Yes |
| Supports conversion to vector format: | Yes |
| Supports reloading: | No |
| Supports spatial indexing: | Yes |

| | |
|---|---|
| Format name: | **GPS Exchange Format** |
| Library file name: | gpx.dll |
| Format ID: | 52 |
| Format extensions: | *.gpx |
| Version of interface: | 1.1 |
| Supports conversion to shapefile: | Yes |
| Supports conversion to vector format: | Yes |
| Supports reloading: | No |
| Supports spatial indexing: | Yes |

| | |
|---|---|
| Format name: | **OGC Geography Markup Language** |
| Library file name: | gml.dll |
| Format ID: | 53 |
| Format extensions: | *.gml, *.xml |
| Version of interface: | 1.1 |
| Supports conversion to shapefile: | Yes |
| Supports conversion to vector format: | Yes |
| Supports reloading: | Yes |
| Supports spatial indexing: | Yes |

# 7. Example

In this chapter we offer you C++ source code of an empty framework of dynamic linked library that uses the interface. Code is tested in Microsoft Visual Studio 2012 enviroment.

## 7.1 How to compile the example

This short chapter describes neccessary steps to compile source code in chapter 7.2. using Microsoft Visual Studio 2012. Simply follow the instructions bellow.

- Run Microsoft Visual Studio 2012.
- From menu File select New and then choose Project. From categories of project types select Win32 in Visual C++ section. From installed templates select Win32 Project. Choose location for new project that will be created and fill project name, then click OK button. In following two steps wizard select „DLL" option and click Finish button.
- A project with dllmain.cpp file is created.
- Copy source code from chapter 7.2. to dllmain.cpp file.
- From Project menu choose projectName Properties (Alt + F7). Select Configuration Properties/General section and set „Character Set“ option to „Not Set“. Now select Configuration Properties/C/C++/Preprocesor and add to Preprocessor Definitions „_CRT_SECURE_NO_WARNINGS“.
- From Build menu choose Build Solution command (F7).

## 7.2 Source code

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

typedef BOOL (__cdecl *CRS2CRS)(int, int, double*, double*, double*);

typedef struct _EXTENT {     // extent of data
   double xMin;
   double yMin;
   double xMax;
   double yMax;
   double zMin;
   double zMax;
   double mMin;
   double mMax;
} EXTENT;

// map units
enum {
    map_units_unknown = 40244,
    map_units_kilometers = 40245,
    map_units_meters = 40246,
    map_units_decimeters = 40253,
    map_units_centimeters = 40247,
    map_units_millimeters = 40248,
    map_units_inches = 40251,
    map_units_feet = 40250,
    map_units_yards = 40254,
    map_units_statute_miles = 40249,
    map_units_nautical_miles = 40255,
    map_units_fathoms = 40256,
    map_units_chains = 40257,
    map_units_links = 40258,
    map_units_us_surveyors_inches = 40259,
    map_units_us_surveyors_feet = 40260,
    map_units_us_surveyors_yards = 40261,
    map_units_us_surveyors_chains = 40262,
    map_units_us_surveyors_statute_miles = 40263,
    map_units_indian_feet = 40264,
    map_units_indian_yards = 40265,
    map_units_indian_chains = 40266,
    map_units_decimal_degrees = 40252
};

typedef struct _VECTORFORMATSTRUCT {
    // mandatory part of structure
    DWORD dwSize;    // size of this structure (including optional part)
    COLORREF mapBkColor;    // background color of map
    // optional part of structure
    // content of optional part depends on implementation of plug-in module
    TCHAR fileName[_MAX_PATH];
} VECTORFORMATSTRUCT;

// private variables for each thread
typedef struct _THREADPARAMS {
    TCHAR buffer[sizeof(VECTORFORMATSTRUCT)];
    EXTENT extent;
} THREADPARAMS;

HINSTANCE hInstanceDll;
static DWORD dwTlsIndex; // address of shared memory


BOOL WINAPI DllMain(HINSTANCE hInstDll, DWORD fdwReason, LPVOID lpReserved)
{
    LPVOID lpvData;
    // Perform actions based on the reason for calling.
    switch(fdwReason)
    {
        case DLL_PROCESS_ATTACH:
            // Initialize once for each new process.
            // Return FALSE to fail DLL load.

            // It is always a good idea to save handle of instance of library
            hInstanceDll = hInstDll;

            // Allocate a TLS index.
            if ((dwTlsIndex = TlsAlloc()) == TLS_OUT_OF_INDEXES)
                return FALSE;
            // No break: Initialize the index for first thread.

        case DLL_THREAD_ATTACH:
            // Do thread-specific initialization.
            // Initialize the TLS index for this thread.
            lpvData = (LPVOID)new THREADPARAMS;
            if (lpvData == NULL) return FALSE;
            if (!TlsSetValue(dwTlsIndex, lpvData)) {
                delete (THREADPARAMS*)lpvData;
                return FALSE;
            }
            break;

        case DLL_THREAD_DETACH:
            // Do thread-specific cleanup.
            // Release the allocated memory for this thread.
            lpvData = TlsGetValue(dwTlsIndex);
            if (lpvData != NULL) {
                THREADPARAMS* threadParams = (THREADPARAMS*)lpvData;
                if (threadParams) delete threadParams;
            }
            break;

        case DLL_PROCESS_DETACH:
            // Perform any necessary cleanup.
            lpvData = TlsGetValue(dwTlsIndex);
            if (lpvData != NULL) {
                THREADPARAMS* threadParams = (THREADPARAMS*)lpvData;
                if (threadParams) delete threadParams;
            }
            // Release the TLS index.
            TlsFree(dwTlsIndex);
            break;
    }
    return TRUE;
```

```
}

// Function returns pointer to parameters for current thread
THREADPARAMS* GetThreadParams()
{
    THREADPARAMS* threadParams = (THREADPARAMS*)TlsGetValue(dwTlsIndex);
    if (!threadParams) {
        threadParams = new THREADPARAMS;
        if (!threadParams) return NULL;
        ZeroMemory(threadParams, sizeof(THREADPARAMS));
        if (!TlsSetValue(dwTlsIndex, (LPVOID)threadParams)) {
            delete threadParams;
            return NULL;
        }
    }
    return threadParams;
}

extern "C" __declspec( dllexport ) void __cdecl BuildSpatialIndex(void* pStruct)
{
    // Build spatial index
    // ...
    // insert your code here
    // ...
    return;
}

extern "C" __declspec( dllexport ) void* __cdecl ChangeSettings(HWND hWndParent, void* pStruct)
{
    THREADPARAMS* threadParams = GetThreadParams();
    if (!threadParams) return NULL;
    memcpy(threadParams->buffer, pStruct, ((VECTORFORMATSTRUCT*)pStruct)->dwSize);
    // Modify copy of structure pStruct and return pointer to modified structure
    // ...
    // insert your code here
    // ...
    // return threadParams->buffer;

    ::MessageBox(hWndParent, "There are no adjustable settings.", "Christine-GIS",
MB_OK|MB_ICONINFORMATION);
    return NULL;
}

extern "C" __declspec( dllexport ) BOOL __cdecl ConvertVectorFormat2Shp(TCHAR* shpFileName, TCHAR*
vectorFormatFileName, HWND hWndMainFrame, void* pStruct, CRS2CRS Crs2Crs, BOOL bUseDefaultSettings)
{
    // Convert vector format to ESRI shapefile and save it to a file
    // ...
    // insert your code here
    // ...
    return FALSE;
}

extern "C" __declspec( dllexport ) BOOL __cdecl ConvertShp2VectorFormat(TCHAR* shpFileName, TCHAR*
vectorFormatFileName, HWND hWndMainFrame, CRS2CRS Crs2Crs, BOOL bUseDefaultSettings, BYTE*
selection)
{
    // Convert ESRI shapefile to vector format and save it to a file
    // ...
    // insert your code here
    // ...
    return FALSE;
}

extern "C" __declspec( dllexport ) void __cdecl Draw(HDC hDC, EXTENT* drawExtent, RECT* clipRect,
int unitSize, void* pStruct, CRS2CRS Crs2Crs, DWORD* pId1, volatile DWORD* pId2)
{
    // Work until *pId1 is equal to *pId2
    if (*pId1 != *pId2) return;

    THREADPARAMS* threadParams = GetThreadParams();
    if (!threadParams) return;

    FILE* f = fopen(((VECTORFORMATSTRUCT*)pStruct)->fileName, "rb");
    // ...
    // insert your code here
    // ...
    // Please don't forget about value of pId2. Check the value as often as possible
    // and reasonable. If the value is changed, clean up the memory and leave
    // the function immediately.
    fclose(f);
}

extern "C" __declspec( dllexport ) EXTENT* __cdecl GetExtent(void* pStruct)
{
    THREADPARAMS* threadParams = GetThreadParams();
    if (!threadParams) return NULL;
    memset(&(threadParams->extent), 0, sizeof(EXTENT));

    FILE* f = fopen(((VECTORFORMATSTRUCT*)pStruct)->fileName, "rb");
    // Fill xMin, yMin, xMax and yMax members of threadParams->extent structure
    // ...
    // insert your code here
    // ...
    fclose(f);

    return &(threadParams->extent);
}

extern "C" __declspec( dllexport ) TCHAR* __cdecl GetFormatName(void* pStruct)
{
    THREADPARAMS* threadParams = GetThreadParams();
    if (!threadParams) return NULL;
    threadParams->buffer[0] = '\0';

    // Fill the threadParams->buffer with name of vector format
    // ...
    // insert your code here
    // ...

    return threadParams->buffer;
}

extern "C" __declspec( dllexport ) TCHAR* __cdecl GetFormatVersion(void* pStruct)
{
    THREADPARAMS* threadParams = GetThreadParams();
    if (!threadParams) return NULL;
    threadParams->buffer[0] = '\0';

    // Fill the threadParams->buffer with version of vector format
    // ...
    // insert your code here
    // ...

    return threadParams->buffer;
}

extern "C" __declspec( dllexport ) DWORD __cdecl GetInterfaceVersion()
{
    // return interface version - 1.0
    return MAKELONG(0, 1);
}

extern "C" __declspec( dllexport ) BOOL __cdecl HasSpatialIndex(void* pStruct)
{
    // ...
    // insert your code here
    // ...
    return FALSE;
}

extern "C" __declspec( dllexport ) BYTE* __cdecl Initialize(TCHAR* fileName)
{
    THREADPARAMS* threadParams = GetThreadParams();
    if (!threadParams) return NULL;
    VECTORFORMATSTRUCT* pStruct = (VECTORFORMATSTRUCT*)threadParams->buffer;
    pStruct->dwSize = sizeof(VECTORFORMATSTRUCT);
    // mapBkColor member of VECTORFORMATSTRUCT is filled by
    // Christine before Draw function is calling
    lstrcpy(pStruct->fileName, fileName);
    // ...
    // insert your code here
    // ...
    return (BYTE*)pStruct;
}
```

```
extern "C" __declspec( dllexport ) BOOL __cdecl NeedsReloading(void* pStruct)
{
    // ...
    // insert your code here, return TRUE if realoading is needed
    // ...
    return FALSE;
}

extern "C" __declspec( dllexport ) void* __cdecl ReadSettingsFromStringLine(TCHAR* settingsStr,
void* pStruct)
{
    THREADPARAMS* threadParams = GetThreadParams();
    if (!threadParams) return NULL;
    memcpy(threadParams->buffer, pStruct, ((VECTORFORMATSTRUCT*)pStruct)->dwSize);
    VECTORFORMATSTRUCT* pVectorStruct = (VECTORFORMATSTRUCT*)threadParams->buffer;
    // Read settings from string line. Save
    // settings to pVectorStruct and return
    // pointer to it.
    // ...
    // insert your code here
    // ...
    // return (BYTE*)pVectorStruct;

    return pVectorStruct;
}

extern "C" __declspec( dllexport ) void __cdecl Release(void* pStruct)
{
    // perform a neccessary clean up - destroy all object you created,
    // clean up memory, delete temporary files, etc.
    // ...
    // insert your code here
    // ...
    return;
}

extern "C" __declspec( dllexport ) BOOL __cdecl Reload(void* pStruct)
{
    // ...
    // insert your code here, return TRUE if reloading is successful
    // ...
    return FALSE;
}

extern "C" __declspec( dllexport ) void __cdecl RemoveSpatialIndex(void* pStruct)
{
    // ...
    // insert your code here
    // ...
    return;
}

extern "C" __declspec( dllexport ) BOOL __cdecl SupportsConversionVectorFormat2Shp()
{
    // If not supported return FALSE, otherwise return TRUE.
    return FALSE;
}

extern "C" __declspec( dllexport ) BOOL __cdecl SupportsConversionShp2VectorFormat()
{
    // If not supported return FALSE, otherwise return TRUE.
    return FALSE;
}

extern "C" __declspec( dllexport ) BOOL __cdecl SupportsReloading()
{
    // If not supported return FALSE, otherwise return TRUE.
    return FALSE;
}

extern "C" __declspec( dllexport ) BOOL __cdecl SupportsSpatialIndex()
{
    // If not supported return FALSE, otherwise return TRUE.
    return FALSE;
}

extern "C" __declspec( dllexport ) TCHAR* __cdecl WriteSettingsToStringLine(void* pStruct)
{
     static TCHAR stringLine[8192];    // 8 kB buffer
     stringLine[0] = '\0';
    // Write settings from pStruct to stringLine
    // ...
    // insert your code here
    // ...

    return stringLine;
}
```