Christine-GIS

Public interface for map service plug-ins

Specification of public interface version 1.0

Christine-GIS

Released: October 2005
Revised: October 2018

Content

1. Introduction

Document describes interface for dynamic linked libraries, which can be used for reading and drawing maps provided by a web map service. The interface allows you create your own library and in this way you can enable communication between Christine and a map service. 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 1.0 of the interface. The interface is supported by Christine-GIS version 2.0 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

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.

double - A 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

HWND - Handle to a window

NULL - Zero value

POINT - Structure that defines the x- and y- coordinates of a point

```
typedef struct tagPOINT {
    LONG x;            // specifies the x-coordinate of the point
    LONG y;            // specifies the y-coordinate of the point
} POINT;
```

POINT* - A pointer to point structure

SIZE - The SIZE structure specifies the width and height of a rectangle

```
typedef struct tagSIZE {
    LONG cx;     // Specifies the rectangle's width
    LONG cy;     // Specifies the rectangle's height
} SIZE;
```

SIZE* - A pointer to size structure

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

TRUE - Boolean value, stored as non zero signed integer

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")

WMSCONNECCTION - connection structure for saving connection parameters and any other data you need

```
typedef struct _WMSCONNECTION {
```

```
      // mandatory part of structure
      DWORD dwSize;     // size of this structure (including optional part)
      TCHAR recommendedThemeName[128];    // recommended name of theme
      int   outputRasterFormat;           // ID of output raster format generated
                                          // by the mmap service
      COLORREF    bgColor;                // background color of map
      // optional part of structure
      // content of optional part depends on implementation of plug-in module
} WMSCONNECTION;
```

WORD - A 16-bit unsigned integer

3. How it works

Christine calls functions of the public interface described in chapter *Public interface of library* when it needs. Library is released after each function call, except these cases:

- calling `BOOL` **`Connect`**`(HWND hWndParent, TCHAR** formatNames, int* formatIDs, void* pSettingsStruct)` function before calling function `void*` **`GetConnectionSettingsStruct`**`()`. It is happen when user adds theme based on a map service or changes properties of the theme.
- calling `BOOL` **`ReadSettingsFromStringLine`**`(TCHAR* settingsStr)` function before calling function `void*` **`GetConnectionSettingsStruct`**`()`. It is happen when user opens a project that contains a theme based on a map service.

3.1 Adding layer based on a map service to a view

Christine looks through Windows registry for available types of known map services. Christine uses values of *WMSName* and *FormatID* keys in order to complete list of supported types of map services and fills combo box in Add Map Service dialog. In this dialog user choose type of map service. When user chooses type of map service, Christine will look for value of *LibFileName* key in Windows registry in order to be able to load the library. If user click on Connect To Map Service button, Christine will load the library and call Connect function. The Connect function should show dialog to allow user to connect a map service. Parameter pSettingsStruct in Connect function is NULL. If connect operation was successful the Connect function will return TRUE. After this Christine calls GetConnectionSettingsStruct function, allocates memory and stores connection structure. The connection structure is used by GetMap and GetDescriptionString functions. So you can use the structure for sharing any data you need among functions.

List of called functions during the operation:

```
BOOL Connect(HWND hWndParent, TCHAR** formatNames, TCHAR** formatMINEs, int*
formatIDs, void* pSettingsStruct)
void* GetConnectionSettingsStruct()
EXTENT* GetExtent(void* pSettingsStruct)
BOOL IsQueryable(void* pSettingsStruct)
```

3.2 Drawing and printing map provided by a map service

Christine loads library to memory and calls GetMap function. Function GetMap uses connection settings structure in order to connect map service, transfers image data and saves it as image file using path and file name specified in fileName parameter. After this releases the library and draws saved image file.

List of called functions during the operation:

```
void GetMap(void* pSettingsStruct, TCHAR* fileName, HWND hWndMainFrame, EXTENT*
mapExtent, SIZE* imgSize, DWORD* pId1, volatile DWORD* pId2)
```

3.3 Getting information about map feature

After user clicks into view which has first active theme based on a map service by Info tool  , Christine loads library to memory and calls GetFeatureInfo function. The way how Christine shows informations depend on returned value of GetFeatureInfo function. For detailed informations about possible returned values see chapter 4.

List of called functions during the operation:

```
TCHAR* GetFeatureInfo(void* pSettingsStruct, TCHAR* fileName, HWND
hWndMainFrame, EXTENT* locality)
```

3.4 Showing properties of theme based on map service

Before properties dialog is shown Christine calls GetExtent, GetServerURL and GetServiceName functions in order to be able to show description of map service. There are property sheet caled Map Service in dialog Theme Properties. The property sheet contains Change Settings button. Users can use the button for changing settings of connected map service. Clicking on the button calls Connect and GetConnectionSettingsStruct functions. Parameter pSettingsStruct in Connect function contains pointer to structure that describes current settings of connection. Prevent users from changing basic parameters of connection like URL or map service, but users should be able to change some others settings like visibility of layers, output raster format and so on.

List of called functions during the operation:

```
BOOL Connect(HWND hWndParent, TCHAR** formatNames, TCHAR** formatMINEs, int*
formatIDs, void* pSettingsStruct)
void* GetConnectionSettingsStruct()
EXTENT* GetExtent(void* pSettingsStruct)
TCHAR* GetServerURL(void* pSettingsStruct)
TCHAR* GetServiceName(void* pSettingsStruct)
BOOL IsQueryable(void* pSettingsStruct)
```

3.5 Saving properties of theme based on a map service to a project file

Christine loads library to memory and calls WriteSettingsToStringLine function. Function WriteSettingsToStringLine uses connection settings structure in order to create string line with settings of theme based on a map service. 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 mapServiceSettings.

List of called functions during the operation:

```
TCHAR* WriteSettingsToStringLine(void* pSettingsStruct)
```

3.6 Reading properties of theme based on a map service from a project file

Christine loads library to memory and calls ReadSettingsFromStringLine function. Function ReadSettingsFromStringLine uses a string line with saved settings of theme based on a map service in order to create connection settings structure. Content of the string line was created by WriteSettingsToStringLine function (see 3.5 chapter). The string line is terminated by binary zero (it is a null-terminated string). Function ReadSettingsFromStringLine will return TRUE if connection settings structure is created successfully, otherwise will return FALSE. If connection settings structure is created successfully, Christine calls GetConnectionSettingsStruct function, creates theme and releases library.

List of called functions during the operation:

```
void* GetConnectionSettingsStruct()
EXTENT* GetExtent(void* pSettingsStruct)
BOOL IsQueryable(void* pSettingsStruct)
BOOL ReadSettingsFromStringLine(TCHAR* settingsStr)
```

4. Public interface of library

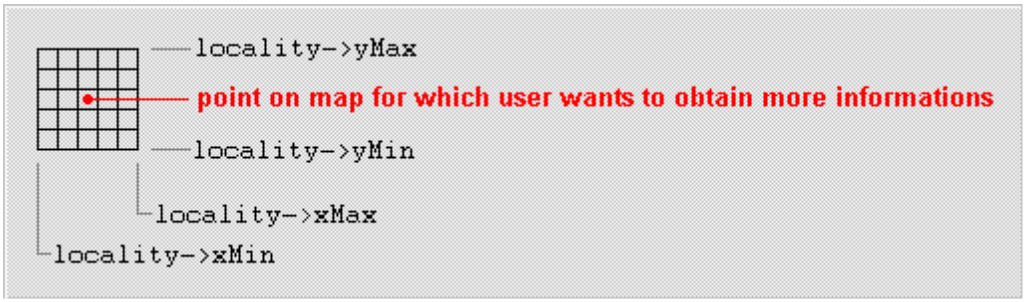
This chapter describes public interface of the library. Each function in the public interface has exactly defined behaviour and it is called in situations described in chapter *How it works*.

`BOOL` **`Connect`**`(HWND hWndParent, TCHAR** formatNames, TCHAR** formatMINEs, int* formatIDs, void* pSettingsStruct)`
Shows dialog where user can specify parameters of connection and connect map service. Parameter hWndParent is handle of parent window for dialog. Parameter formatNames is pointer to array of names of raster formats. Array is terminated by NULL pointer. Use the list to allow user specify raster format generated by map service. Parameter formatMINEs is array of MINE strings of raster formats that are listed in formatNames array. Order of format names and MINE strings in these arrays is the same, so first raster format has name formatNames[0] and MINE string MINEs[0]. Parameter formatIDs is array of identification numbers of raster formats that are listed in formatNames array. Order of format names and format IDs in these arrays is the same, so first raster format has name formatNames[0] and indentification number formatIDs[0]. Parameter pSettingsStruct is NULL or contains pointer to structure that describes current settings of connection (see 3.1 and 3.4 chapters). This function has to fill or modify connection structure in order to allow Christine to save the structure. If connection was successful function will return TRUE, otherwise will return FALSE.

`void*` **`GetConnectionSettingsStruct`**`()`
Returns pointer to structure that describes settings of connection and possibly stores any other informations you need.

`EXTENT*` **`GetExtent`**`(void* pSettingsStruct)`
Returns pointer to extent of all data from which is map created in map units. Parameter pSettingsStruct is a structure saved during connection process, see chapter 3.1 for details.

`TCHAR*` **`GetFeatureInfo`**`(void* pSettingsStruct, TCHAR* fileName, HWND hWndMainFrame, EXTENT* locality)`
The response to GetFeatureInfo request is always a computer file that is transferred over the internet from the server to the client. The file contains information about a map feature. Format of the file is indicated by return value, see bellow. Parameter fileName is full file name of the file. Use the file name for saving transferred file. Parameter pSettingsStruct is pointer to a structure saved during connection process, see chapter 3.1 for details. Parameter hWndMainFrame is handle to Christine's main window. Parameter locality defines neighbourhood of point on map for which user wants to obtain more informations. Parameter locality is in map coordinates and represents square of five pixels width and five pixel height on user's screen. In the center of the square is point for which user wants to obtain more informations.



Function will return NULL if failed, otherwise will return pointer to MINE string that specifies format of file with informations. Christine-GIS version 5.x accepts these values:

- "text/plain" for plain text. These informations will be shown using Report metod from class MsgBox (see to Christine's script language manual).
- "text/richtext" for RTF text. These informations will be shown using Report metod from class MsgBox (see to Christine's script language manual).
- "text/html" for HTML text. These informations will be shown using ReportHTML metod from class MsgBox (see to Christine's script language manual).
- an other MINE string for a known raster format. These informations will be shown using ShowBmp metod from class MsgBox (see to Christine's script language manual).

`DWORD` **`GetInterfaceVersion`**`()`
Returns version of the interface. In high word is major version (1) and in low word is minor version (0).

`void` **`GetMap`**`(void* pSettingsStruct, TCHAR* fileName, HWND hWndMainFrame, EXTENT* mapExtent, SIZE* imgSize, DWORD* pId1, volatile DWORD* pId2)`
The response to GetMap request is always a computer file that is transferred over the internet from the server to the client. The file represents a map image in ouput raster format accepted by Connect function. Parameter fileName is full file name of the raster file. Use the file name for saving transferred raster file. Parameter pSettingsStruct is a structure saved during connection process, see chapter 3.1 for details. Parameter hWndMainFrame is handle to Christine's main window. Parameter mapExtent is required extent for map in map units. Parameter imgSize is required size of image file in pixels. Parameters pId1 and pId2 are pointers to numeric identificators. If the values of identificators are the same you can continue transmission of data, but if pId2 is changed you should stop transmission immediately, clean up the memory and leave the function.

`TCHAR*` **`GetServerURL`**`(void* pSettingsStruct)`
Returns pointer to string that contains URL of the map service.

`TCHAR*` **`GetServiceName`**`(void* pSettingsStruct)`
Returns pointer to string that contains name of the map service.

`BOOL` **`IsQueryable`**`(void* pSettingsStruct)`
Returns TRUE if map service is queryable. It means that map service is able to handle GetFeatureInfo request.

`BOOL` **`ReadSettingsFromStringLine`**`(TCHAR* settingsStr)`
Reads settings from string line when a project file is openning.

`TCHAR*` **`WriteSettingsToStringLine`**`(void* pSettingsStruct)`
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 map in a view and to print the map to printer. To create functional library that do this you must implement following functions with described behaviour:

`BOOL` **`Connect`**`(HWND hWndParent, TCHAR** formatNames, TCHAR** formatMINEs, int* formatIDs, void* pSettingsStruct)`
Allows users to connect map service and configure it. Creates and fills connection structure.

`void*` **`GetConnectionSettingsStruct`**`()`
Returns pointer to connection structure.

`EXTENT*` **`GetExtent`**`(void* pSettingsStruct)`
Returns pointer to extent of all data from which is map created.

`TCHAR*` **`GetFeatureInfo`**`(void* pSettingsStruct, TCHAR* fileName, HWND hWndMainFrame, EXTENT* locality)`
Returns NULL.

`DWORD` **`GetInterfaceVersion`**`()`
Returns version of the interface. In high word is major version (1) and in low word is minor version (0).

`void` **`GetMap`**`(void* pSettingsStruct, TCHAR* fileName, HWND hWndMainFrame, EXTENT* mapExtent, SIZE* imgSize, DWORD* pId1, volatile DWORD* pId2)`
Saves map to file.

`TCHAR*` **`GetServerURL`**`(void* pSettingsStruct)`
Returns pointer to string that contains URL of the map service.

`TCHAR* **GetServiceName**(void* pSettingsStruct)`
Returns pointer to string that contains name of the map service.

`BOOL **IsQueryable**(void* pSettingsStruct)`
Returns FALSE.

`BOOL **ReadSettingsFromStringLine**(TCHAR* settingsStr)`
Reads settings from string line when a project file is openning.

`TCHAR* **WriteSettingsToStringLine**(void* pSettingsStruct)`
Writes settings to string line.

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 information to Windows registry:

Key:
HKEY_LOCAL_MACHINE\Software\Christine-GIS\5.x\WMS\[ID of type of map service]

Values:

Name	Value	Type	Max. length (byte)	
LibFileName	[library.dll]	REG_SZ	255	Mandatory
WMSName	Name of type of map service	REG_SZ	255	Mandatory
FormatID	[ID of map service format]	REG_DWORD	4	Mandatory

Example for OGC WMS:

Key:
HKEY_LOCAL_MACHINE\Software\Christine-GIS\5.x\WMS\1002

Values:

Name	Value
LibFileName	ogcwms.dll
WMSName	OGC Web Map Service
FormatID	1002

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

5.1 Identification number of type of map service (ID)

If you want to create your own library for showing maps provided by a web map service, look at Christine's web site for free IDs. After you choose a free 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

One library comes with Christine-GIS version 5.x, that has implemented this interface. It is placed in subfolder dlls in Christine's installation folder.

Type of map service:	**OGC Web Map Service**
Library file name:	ogcwms.dll
ID:	1002
Version of interface:	1.0

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 2005 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 2005. Simply follow the instructions bellow.

- Run Microsoft Visual Studio 2005.
- 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 basic main source file projectName.cpp is created.
- Copy source code from chapter 7.2. to created 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 struct _EXTENT { // extent of data
   double xMin;
   double yMin;
   double xMax;
   double yMax;
   double zMin;
   double zMax;
   double mMin;
   double mMax;
} EXTENT;

// connection structure for saving connection
// parameters and any other data you need
typedef struct _WMSCONNECTION {
        // mandatory part of structure
        DWORD   dwSize;        // size of this structure (including optional part)
        TCHAR   recommendedThemeName[128];    // recommended name of theme
        int             outputRasterFormat;   // ID of output raster
                                                        // format generated by the mmap service
        COLORREF        bgColor;                        // background color of map
        // optional part of structure - depend on your needs
        // ...
        // declare your own members of connection structure here
        // ...
        EXTENT  extent;
        TCHAR   buffer[1024];
} WMSCONNECTION;

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 BYTE[sizeof(WMSCONNECTION)];
                     if (lpvData == NULL) return FALSE;
                     if (!TlsSetValue(dwTlsIndex, lpvData)) {
                            delete[] 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) {
                            BYTE* wmsConnection = (BYTE*)lpvData;
                            if (wmsConnection) delete[] wmsConnection;
                     }
                     break;

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

extern "C" __declspec( dllexport ) BOOL __cdecl Connect(HWND hWndParent, TCHAR** formatNames,
TCHAR** formatMINEs, int* formatIDs, void* pSettingsStruct)
{
       BYTE* lpvData = (BYTE*)TlsGetValue(dwTlsIndex);
       if (!lpvData) {
              lpvData = new BYTE[sizeof(WMSCONNECTION)];
              if (!lpvData) return FALSE;
              if (!TlsSetValue(dwTlsIndex, lpvData)) {
                     delete[] lpvData;
                     return FALSE;
              }
       }
       // copy connection structure
       if (pSettingsStruct) {
              delete[] lpvData;
              lpvData = new BYTE[((DWORD*)pSettingsStruct)[0]];
              if (!lpvData) return FALSE;
              memcpy(lpvData, pSettingsStruct, ((DWORD*)pSettingsStruct)[0]);
              if (!TlsSetValue(dwTlsIndex, lpvData)) {
                     delete[] lpvData;
                     return FALSE;
              }
       }
       // Initialize connection structure
       WMSCONNECTION* wmsConnection = (WMSCONNECTION*)lpvData;
       wmsConnection->dwSize = sizeof(WMSCONNECTION);
```

```
	lstrcpy(wmsConnection->recommendedThemeName, "aThemeName");
	wmsConnection->outputRasterFormat = 7;              // PNG
	wmsConnection->bgColor = RGB(255,255,255);  // white color

	// Show a dialog to allow user to connect a map service
	// or modify connection and settings of map service.
	// If connection is successful, disconnect the map service,
	// save connection parameters and any other data you need to
	// connection structure and return TRUE. If connection is not successful
	// return FALSE;
	// ...
	// insert your code here
	// ...

	return TRUE;
}

extern "C" __declspec( dllexport ) void* __cdecl GetConnectionSettingsStruct()
{
	// return poiner to connection structure
	// Christine saves it for you
	return TlsGetValue(dwTlsIndex);
}

extern "C" __declspec( dllexport ) EXTENT* __cdecl GetExtent(void* pSettingsStruct)
{
	// pointer to connection structure
	WMSCONNECTION* wmsConnection = (WMSCONNECTION*)pSettingsStruct;

	memset(&(wmsConnection->extent), 0, sizeof(EXTENT));

	// Fill xMin, yMin, xMax and yMax members of extent structure
	// ...
	// insert your code here
	// ...

	return &(wmsConnection->extent);
}

extern "C" __declspec( dllexport ) TCHAR* __cdecl GetFeatureInfo(void* pSettingsStruct, TCHAR*
fileName, HWND hWndMainFrame, EXTENT* locality)
{
	BYTE* lpvData = (BYTE*)TlsGetValue(dwTlsIndex);
	if (!lpvData) delete[] lpvData;
	// copy connection structure
	lpvData = new BYTE[((DWORD*)pSettingsStruct)[0]];
	if (!lpvData) return NULL;
	memcpy(lpvData, pSettingsStruct, ((DWORD*)pSettingsStruct)[0]);
	if (!TlsSetValue(dwTlsIndex, lpvData)) {
		delete[] lpvData;
		return NULL;
	}
	WMSCONNECTION* wmsConnection = (WMSCONNECTION*)lpvData;

	// Coordinate of point for which user wants to obtain more informations
	double x = locality->xMin + ((locality->xMax - locality->xMin) / 2);
	double y = locality->yMin + ((locality->yMax - locality->yMin) / 2);

	// Get information about a map feature and save it to file named fileName
	// ...
	// insert your code here
	// ...

	// if all is successfully done, return pointer to MINE string
	// that indicates output format, otherwise return NULL

	// return wmsConnection->buffer;
	return NULL;
}

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

```
extern "C" __declspec( dllexport ) void __cdecl GetMap(void* pSettingsStruct, TCHAR* fileName, HWND
hWndMainFrame, EXTENT* mapExtent, SIZE* imgSize, DWORD* pId1, volatile DWORD* pId2)
{
	BYTE* lpvData = (BYTE*)TlsGetValue(dwTlsIndex);
	if (!lpvData) delete[] lpvData;
	// copy connection structure
	lpvData = new BYTE[((DWORD*)pSettingsStruct)[0]];
	if (!lpvData) return;
	memcpy(lpvData, pSettingsStruct, ((DWORD*)pSettingsStruct)[0]);
	if (!TlsSetValue(dwTlsIndex, lpvData)) {
		delete[] lpvData;
		return;
	}
	WMSCONNECTION* wmsConnection = (WMSCONNECTION*)lpvData;

	// Work until *pId1 is equal to *pId2
	if (*pId1 != *pId2) return;

	// ...
	// 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,
	// cleanup the memory and leave the function immediately.
	// ...

}

extern "C" __declspec( dllexport ) TCHAR* __cdecl GetServiceName(void* pSettingsStruct)
{
	BYTE* lpvData = (BYTE*)TlsGetValue(dwTlsIndex);
	if (!lpvData) delete[] lpvData;
	// copy connection structure
	lpvData = new BYTE[((DWORD*)pSettingsStruct)[0]];
	if (!lpvData) return NULL;
	memcpy(lpvData, pSettingsStruct, ((DWORD*)pSettingsStruct)[0]);
	if (!TlsSetValue(dwTlsIndex, lpvData)) {
		delete[] lpvData;
		return NULL;
	}
	WMSCONNECTION* wmsConnection = (WMSCONNECTION*)lpvData;
	wmsConnection->buffer[0] = '\0';

	// Fill the wmsConnection->buffer with name of map service
	// ...
	// insert your code here
	// ...

	return wmsConnection->buffer;
}

extern "C" __declspec( dllexport ) TCHAR* __cdecl GetServerURL(void* pSettingsStruct)
{
	BYTE* lpvData = (BYTE*)TlsGetValue(dwTlsIndex);
	if (!lpvData) delete[] lpvData;
	// copy connection structure
	lpvData = new BYTE[((DWORD*)pSettingsStruct)[0]];
	if (!lpvData) return NULL;
	memcpy(lpvData, pSettingsStruct, ((DWORD*)pSettingsStruct)[0]);
	if (!TlsSetValue(dwTlsIndex, lpvData)) {
		delete[] lpvData;
		return NULL;
	}
	WMSCONNECTION* wmsConnection = (WMSCONNECTION*)lpvData;
	wmsConnection->buffer[0] = '\0';

	// Fill wmsConnection->buffer with URL of map service
	// ...
	// insert your code here
	// ...

	return wmsConnection->buffer;
}
```

```
extern "C" __declspec( dllexport ) BOOL __cdecl IsQueryable(void* pSettingsStruct)
{
	BYTE* lpvData = (BYTE*)TlsGetValue(dwTlsIndex);
	if (!lpvData) delete[] lpvData;
	// copy connection structure
	lpvData = new BYTE[((DWORD*)pSettingsStruct)[0]];
	if (!lpvData) return FALSE;
	memcpy(lpvData, pSettingsStruct, ((DWORD*)pSettingsStruct)[0]);
	if (!TlsSetValue(dwTlsIndex, lpvData)) {
		delete[] lpvData;
		return FALSE;
	}
	WMSCONNECTION* wmsConnection = (WMSCONNECTION*)lpvData;

	// Check if map service is queryable
	// ...
	// insert your code here
	// ...

	// return TRUE if map service is queryable, otherwise return FALSE
	return FALSE;
}

extern "C" __declspec( dllexport ) BOOL __cdecl ReadSettingsFromStringLine(TCHAR* settingsStr)
{
	BYTE* lpvData = (BYTE*)TlsGetValue(dwTlsIndex);
	if (!lpvData) delete[] lpvData;
	// create connection structure
	lpvData = new BYTE[sizeof(WMSCONNECTION)];
	if (!lpvData) return FALSE;
	if (!TlsSetValue(dwTlsIndex, lpvData)) {
		delete[] lpvData;
		return FALSE;
	}
	// Initialize connection structure
	WMSCONNECTION* wmsConnection = (WMSCONNECTION*)lpvData;
	wmsConnection->dwSize = sizeof(WMSCONNECTION);
	lstrcpy(wmsConnection->recommendedThemeName, "aThemeName");
	wmsConnection->outputRasterFormat = 7;              // PNG
	wmsConnection->bgColor = RGB(255,255,255);  // white color

	// fill connection structure
	// ...
	// insert your code here
	// ...

	// if connection structure is successfully filled return TRUE
	return TRUE;
}

extern "C" __declspec( dllexport ) TCHAR* __cdecl WriteSettingsToStringLine(void* pSettingsStruct)
{
	BYTE* lpvData = (BYTE*)TlsGetValue(dwTlsIndex);
	if (!lpvData) delete[] lpvData;
	// copy connection structure
	lpvData = new BYTE[((DWORD*)pSettingsStruct)[0]];
	if (!lpvData) return NULL;
	memcpy(lpvData, pSettingsStruct, ((DWORD*)pSettingsStruct)[0]);
	if (!TlsSetValue(dwTlsIndex, lpvData)) {
		delete[] lpvData;
		return NULL;
	}
	WMSCONNECTION* wmsConnection = (WMSCONNECTION*)lpvData;
	wmsConnection->buffer[0] = '\0';

	// fill wmsConnection->buffer
	// ...
	// insert your code here
	// ...

	return wmsConnection->buffer;
}
```