Appendix E

Creating an MFC-Based XOP With ActiveX

Appendix E — XOPs Using MFC

Overview

WebBrowser is a demonstration Igor XOP built as an "MFC regular DLL". MFC stands for "Microsoft Foundation Class" and is a Microsoft technology for creating Windows programs in C++. MFC is mostly used for creating standalone applications but we will use it to create an XOP which is a form of DLL (dynamic link library).

The motivation for creating an MFC XOP is to use MFC's support for accessing ActiveX controls. Many scientific instruments come with ActiveX controls which provide the means for controlling the instruments programmatically. Rather than using an instrument ActiveX control for demonstration purposes, this chapter uses Microsoft's web browser control because it is already installed on every Windows system and does not require any additional hardware. But the principles and techniques discussed here apply to all ActiveX controls.

Some ActiveX controls provide a visual component and support "in place activation". This means that the ActiveX control's visual component can be embedded in a window of another application. Such an ActiveX control also usually provides functions that can be called by the host application to control aspects of the visual component and related functionality. This is the type of ActiveX control that we are dealing with here.

Another type of ActiveX control provides functions that the host application can call but no visual component. This type of ActiveX control can also be wrapped in an XOP but that is not covered in this document.

An ActiveX control is an ActiveX server, usually supplied in the form of a DLL, hosted by an ActiveX container, which is usually an application. MFC provides the containment functionality for the application. It also creates wrapper functions that allow us to invoke methods and get and set properties of the ActiveX control. To implement this in straight C++ would require a very high degree of ActiveX expertise.

Using MFC, that level of expertise is not needed. But some familiarity with MFC is required. The Visual C++ help system includes a wealth of information about MFC but does not include a good overview or introduction to MFC. The Wikipedia entry for MFC provides good background information.

Visual C++ supports creating two kinds of MFC DLLs: MFC extension DLLs and MFC regular DLLs. In an MFC extension DLL, pointers to MFC objects can be passed between the DLL and the host MFC application. Igor is not an MFC application so we will not create an MFC extension DLL. In an MFC regular DLL, the DLL uses MFC and the host application may or may not use MFC, but there is no sharing of MFC objects between the two. We will create an MFC regular DLL.

Static and Dynamic Linking

An MFC program links to the MFC library which provides the core MFC functionality. The program may be statically linked or dynamically linked to the MFC library. Static linking means that the necessary MFC library code is stored in the program file rather than accessed from Microsoft DLLs through dynamic linking at runtime.

With dynamic linking, the MFC library DLLs must be installed on the end user's system. This library is not a standard part of Windows so, if you dynamically link to MFC, you must provide the MFC library DLL to the end user.

To avoid this hassle, static linking is preferable. However, static linking to MFC is not supported in the "standard" version of Visual C++ - you need the "professional" version. In this chapter we will use static linking but we have also tested dynamic linking. MFC Technical Note TN011 addresses these linking issues.

The Web Browser ActiveX Control

Typically you would create an MFC XOP to interface Igor to an instrument that comes with an ActiveX control. For the purpose of this tutorial, we wanted to use an ActiveX control that everyone would have, so we chose the Microsoft Web Browser control. This control resides in C:\Windows\System32\shdocvw.dll. You can learn more about it in the article Reusing the WebBrowser Control on Microsoft's web site.

Now we will build the WebBrowser XOP using Visual C++.NET 2003 which we call VC7 and Visual C++ 2005 which we call VC8. The free "express" edition of VC8 does not support MFC; you need the standard edition or better.

Igor Pro 6.02 Or Later Is Required

Because of a compatibility issue, you must have Igor Pro 6.02B01 or later to run the WebBrowser XOP. The compatibility issue is explained below in the Issues section.

Creating the Visual C++ Project

Now we will create the WebBrowser XOP project, including source code. You will learn the most if you go through all of the following steps. However, the finished project can is available from:

ftp://ftp.wavemetrics.net/IgorXOPToolkit/Examples/MFCXOP_VC7.zip
ftp://ftp.wavemetrics.net/IgorXOPToolkit/Examples/MFCXOP_VC8.zip

In this section, you will be asked to paste code into files that you will create in VC7 or VC8. This does not work well from a PDF file because indentation does not survive the copy/paste process.

Therefore we supply this file as a Microsoft Word document (.doc) that can be opened in Word or in WordPad.

The steps below labeled "Visual C++ 7 and 8" are intended for both Visual C++ 7 and Visual C++ 8. Other steps are different for different versions of Visual C++ and are labeled "Visual C++ 7 Only" or "Visual C++ 8 Only".

Creating the New Project In Visual C++

1-Visual C++ 7 and 8.

On the desktop, create a folder named WebBrowser inside your IgorXOPs5 folder.

WebBrowser will hold our project and source files.

2-Visual C++ 7 and 8.

In Visual C++, choose File->New->Project to display the New Project dialog.

From the Project Types list, select Visual C++ Projects.

From the Templates list, select MFC DLL.

Enter WebBrowser as the project name.

In the Location box, enter the path to your IgorXOPs5 folder.

3-Visual C++ 7 and 8.

Click the OK button.

Visual C++ 7 displays an "MFC DLL Wizard" window.

4-Visual C++ 7 and 8.

Click Application Settings.

Click the Regular DLL With MFC Statically Linked radio button.

If you do not have the professional version of Visual C++, you will not be able to use static linking. Click "Regular DLL using shared MFC DLL" instead.

Click the Finish button.

Visual C++ creates the project files and starter source files.

5-Visual C++ 7 Only.

If you are using VC7, verify that you now have the following hierarchy:

IgorXOPs5
WebBrowser
WebBrowser.sln
WebBrowser.vcproj
<Various other files created by VC7>

5-Visual C++ 8 Only.

If you are using VC8, verify that you now have the following hierarchy:

IgorXOPs5
WebBrowser
WebBrowser.sln
<Various other files created by VC8>
WebBrowser
WebBrowser.vcproj
<Various other files created by VC8>

6-Visual C++ 7 and 8.

In Visual C++, choose View->Solution Explorer.

In the Solution Explorer window, open the WebBrowser icon and all of its subfolders.

The WebBrowser.h and WebBrowser.cpp files contain starter code generated by Visual C++. This code creates a class named CWebBrowserApp which represents the application (the XOP in this case) as a whole. CWebBrowserApp is derived from MFC's CWinApp class.

Now we will add a source file to contain the main XOP code, that is, the code that interfaces to Igor.

7-Visual C++ 7 Only.

Right-click the Source Files icon in the Solution Explorer window and choose Add->New Item.

In the Add New Item dialog, select Visual C++ from the Categories section and "C++ File (.cpp)" from the Templates section.

In the name box, enter WebBrowserXOP.cpp.

In the Location section, enter the path to your WebBrowser folder.

Click Open.

A new source code file is added to the project and opened for editing.

7-Visual C++ 8 Only.

Right-click the Source Files icon in the Solution Explorer window and choose Add->New Item.

In the Add New Item dialog, select Visual C++ from the Categories section, Code from the list of types under the Visual C++ icon, and "C++ File (.cpp)" from the Templates section.

In the name box, enter WebBrowserXOP.cpp.

In the Location section, enter the path to the WebBrowser project folder.

The WebBrowser project folder is located at "IgorXOPs5\WebBrowser\WebBrowser".

Click Add.

A new source code file is added to the project and opened for editing.

8-Visual C++ 7 and 8.

Enter the following skeletal text in the new WebBrowserXOP.cpp file:

```
#include "stdafx.h"
#include "XOPStandardHeaders.h"
#include "WebBrowserXOP.h"

static int
DoFunction(void)
{
    int funcIndex;
    void *p;      // Pointer to struct containing function params and result.
    int err = 0;

    try {
        funcIndex = GetXOPItem(0);     // Which function invoked?
        p = (void*)GetXOPItem(1);      // Get pointer to params/result.
        switch (funcIndex) {
        }
    }
    catch(CException* e) {     // An exception was thrown by MFC.
        char message[256];
```

```
        char errorMessage[200];
        e->GetErrorMessage(errorMessage, sizeof(message), NULL);
        sprintf(message, "MFC Exception: %s"CR_STR, errorMessage);
        XOPNotice(message);
        e->Delete();
        err = MFC_EXCEPTION_ERROR;
    }
    catch(...) {  // An exception was thrown by MFC.
        err = MFC_EXCEPTION_ERROR;
    }

    return(err);
}

static void
XOPEntry(void)
{
    long err = 0;

    AFX_MANAGE_STATE(AfxGetStaticModuleState());   // For MFC XOPs only.

    switch (GetXOPMessage()) {
        case FUNCTION:
            err = DoFunction();
            break;
    }
    SetXOPResult(err);
}

HOST_IMPORT void
main(IORecHandle ioRecHandle)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());   // For MFC XOPs only.

    XOPInit(ioRecHandle);      // Do standard XOP initialization.
    SetXOPEntry(XOPEntry);     // Set entry point for future calls.

    if (igorVersion < 600)
        SetXOPResult(REQUIRES_IGOR_600);
    else
        SetXOPResult(0L);
}
```

main is the function initially called by Igor to initialize the XOP. XOPEntry is the function to which subsequent messages from Igor are sent. Igor calls DoFunction when one of the XOP's external functions is invoked. Soon we will add cases to the switch statement in DoFunction.

The AFX_MANAGE_STATE macro ensures that MFC will access MFC state information for our XOP and not for some other MFC XOP that might also be running. The macro must be used at all points through which an MFC DLL's code may be entered. For an XOP, this translates to the start of any function that can be called directly from Igor. In this XOP, only

the main and XOPEntry functions are called directly from Igor. Additional information on AFX_MANAGE_STATE is included below.

9-Visual C++ 7 and 8.

Save and close the WebBrowserXOP.cpp file.

Now we will add a corresponding .h file.

10-Visual C++ 7 Only.

Right-click the Header Files icon in the Solution Explorer window and choose Add->New Item.

In the Add New Item dialog, select Visual C++ from the Categories section and "Header File (.h)" from the Templates section.

In the name box, enter WebBrowserXOP.h.

In the Location section, enter the path to your WebBrowser folder.

Click Open

A new header file is added to the project and opened for editing.

10-Visual C++ 8 Only.

Right-click the Header Files icon in the Solution Explorer window and choose Add->New Item.

In the Add New Item dialog, select Visual C++ from the Categories section, Code from the list of types under the Visual C++ icon, and "Header File (.h)" from the Templates section.

In the name box, enter WebBrowserXOP.h.

In the Location section, enter the path to the WebBrowser project folder.

The WebBrowser project folder is located at "IgorXOPs5\WebBrowser\WebBrowser".

Click Add

A new header file is added to the project and opened for editing.

11-Visual C++ 7 and 8.

Enter the following text in the new WebBrowserXOP.h file:

```
// Custom error codes
#define REQUIRES_IGOR_600 1 + FIRST_XOP_ERR
#define ACTIVEX_COMPONENT_INIT_FAILED 2 + FIRST_XOP_ERR
#define ACTIVEX_COMPONENT_ERROR 3 + FIRST_XOP_ERR
#define ACTIVEX_UNEXPECTED_RESULT_TYPE 4 + FIRST_XOP_ERR
#define NULL_INPUT_STRING 5 + FIRST_XOP_ERR
#define ACTIVEX_CONTROLS_PARENT_MISSING  6 + FIRST_XOP_ERR
#define ACTIVEX_CONTROL_MISSING 7 + FIRST_XOP_ERR
#define ACTIVEX_CONTROL_UNKNOWN_ERROR 8 + FIRST_XOP_ERR
#define MFC_EXCEPTION_ERROR 9 + FIRST_XOP_ERR
```

```
// Prototypes
HOST_IMPORT void main(IORecHandle ioRecHandle);
```

12-Visual C++ 7 and 8.

Save and close the WebBrowserXOP.h file.

Now we will add a custom resource file which will contain Igor-specific resources.

13-Visual C++ 7 and 8.

In Visual C++, choose File->New->File.

Select General from the Categories section.

Select Text File from the Templates section.

Click Open.

A new empty text file is created.

Choose File->Save TextFile1 As.

In the resulting dialog, choose All Files from the Save As Type popup menu.

Enter WebBrowserWinCustom.rc as the file name.

Navigate to the WebBrowser project folder which contains WebBrowserXOP.cpp.

Click Save.

14-Visual C++ 7 and 8.

Paste the following XOP custom resource text in the new file:

```
#include "XOPResources.h"  // Defines XOP-specific symbols.

1100 STR#                  // Custom error messages.
BEGIN
   "WebBrowser requires Igor 6.0 or later.\0",
   "ActiveX component initialization failed.\0",
   "ActiveX function returned an error.\0",
   "ActiveX function returned an unexpected type of result.\0",
   "Input string parameter is NULL.\0",
   "The parent window of the ActiveX control does not exist. Call WebBrowserCreateWindo
   "The ActiveX control does not exist. Call WebBrowserCreateWindow.\0",
   "Unknown ActiveX control return code.\0",
   "An MFC (Microsoft Foundation Class) exception was thrown.\0",

   0,                          // NOTE: 0 required to terminate the resource.
END

1100 XOPI              // XOPI - Describes general XOP properties to IGOR.
BEGIN
   XOP_VERSION,               // Version number of host XOP system.
   DEV_SYS_CODE,              // Code for development system used to make XOP.
   0,                         // Obsolete - set to zero.
   0,                         // Obsolete - set to zero.
   XOP_TOOLKIT_VERSION        // XOP Toolkit version.
```

```
END
```

The STR# resource contains XOP custom error messages. The XOPI resource contains information that Igor examines before loading the XOP.

15-Visual C++ 7 and 8.

Save and close the WebBrowserWinCustom.rc file.

Now we will instruct Visual C++ to include our custom resource file.

16-Visual C++ 7 and 8.

In the Solution Explorer window, double-click the WebBrowser.rc icon to open it in Resource View.

Right-click the WebBrowser.rc folder icon in Resource View and choose Resource Includes.

In the resulting Resource Includes window, enter this text at the bottom of the Compile-time Directive section:

```
#include "WebBrowserWinCustom.rc"
```

Click OK.

Visual C++ will display a dialog saying "Directive text will be written into your resource script and may render it uncompilable."

Click OK.

Close the Resource View Window.

Now we will add the XOPSupport library and the IGOR library to the project.

17-Visual C++ 7 Only.

Choose View->Solution Explorer.

Click the WebBrowser icon in the Solution Explorer to select it.

Choose Project->Add Existing Item.

In the resulting dialog, choose All Files from the Files of Type menu.

Navigate to the IgorXOPs5\XOPSupport\VC7 folder and add XOPSupport.lib.

17-Visual C++ 8 Only.

Choose View->Solution Explorer.

Click the WebBrowser icon in the Solution Explorer to select it.

Choose Project->Add Existing Item.

In the resulting dialog, choose All Files from the Files of Type menu.

Navigate to the IgorXOPs5\XOPSupport\VC8 folder and add XOPSupport.lib.

If a dialog appears saying "A custom build rule to build files with extension '.lib' could not be found," click No.

18-Visual C++ 7 and 8.

Choose Project->Add Existing Item again.

In the resulting dialog, choose All Files from the Files of Type menu.

Navigate to the IgorXOPs5\XOPSupport folder and add IGOR.lib.

Now we will set the project properties.

19-Visual C++ 7 Only.

In Solution Explorer, right-click the WebBrowser icon and choose Properties.

In the WebBrowser Property Pages window, in the Configuration menu, choose All Configurations.

Enter the settings shown below.

Only things that need to be changed relative to the default settings are listed here.

General

Build Browser Information: Yes

C/C++

General

Additional Include Directories: ..\XOPSupport

Detect 64-bit Portability Issues: No

Code Generation

Runtime Library: Multi-Threaded (/MT)

Linker

General

Output File: WebBrowser.xop

Input

Additional Dependencies: version.lib

Resources

Additional Include Directories: $(IntDir), ..\XOPSupport

19-Visual C++ 8 Only.

In Solution Explorer, right-click the WebBrowser icon and choose Properties.

In the WebBrowser Property Pages window, in the Configuration menu, choose All Configurations.

Enter the settings shown below.

Only things that need to be changed relative to the default settings are listed here.

General

Character Set: Use Multi-Byte Character Set

C/C++

General

Additional Include Directories: ..\..\XOPSupport

Detect 64-bit Portability Issues: No

Preprocessor

Preprocessor Definitions: Add _CRT_SECURE_NO_WARNINGS

Code Generation

If you are using the static MFC library:

Runtime Library: Multi-Threaded (/MT)

If you are using the dynamic MFC library:

Runtime Library: Multi-Threaded DLL (/MD)

Browse Information

Enable Browse Information: Include All Browse Information (/FR)

Linker

General

Output File: WebBrowser.xop

Input

Additional Dependencies: version.lib

Resources

General

Additional Include Directories: $(IntDir), ..\ ..\XOPSupport

20-Visual C++ 7 and 8.

Click the OK button in the WebBrowser Property Pages window.

21-Visual C++ 7 and 8.

In Solution Explorer, right-click the WebBrowser icon and choose Properties.

In the Configuration menu, choose Debug and enter the following setting:

Debugging

Command: <Path to your Igor Pro executable>
(e.g., C:\Program Files\WaveMetrics\Igor Pro Folder\Igor.exe)

Click the OK button.

22-Visual C++ 7 and 8.

Select the WebBrowser icon in the Solution Explorer window.

Choose File->Save WebBrowser.

23-Visual C++ 7 Only.

Choose Build->Build WebBrowser.

Visual C++ 7 should build the XOP with no errors (though you may get some warnings), creating the file WebBrowser.xop in your IgorXOPs5\WebBrowser folder.

23-Visual C++ 8 Only.

Choose Build->Build WebBrowser.

Visual C++ 7 should build the XOP with no errors (though you may get some warnings), creating the file WebBrowser.xop in your IgorXOPs5\WebBrowser\WebBrowser folder.

24-Visual C++ 7 and 8.

Make a shortcut for the WebBrowser.xop file and put the shortcut in your Igor Extensions folder.

This shortcut causes Igor to load the XOP the next time Igor is launched.

Now we will add a dialog to the WebBrowser XOP. Later we will put a web browser ActiveX control in the dialog.

For our purposes, the ActiveX control must be in a modeless dialog window so that we can access its functionality without its window being the active window. We will therefore create a modeless dialog.

25-Visual C++ 7 and 8.

In Visual C++, in the Solution Explorer, open the Source Files icon and then the Resource Files icon and double-click the WebBrowser.rc icon.

This displays the Resource View.

Right-click the WebBrowser.rc icon in the Resource View and choose Add Resource.

In the Add Resource dialog, click the Dialog icon and then the New button.

This adds a new dialog named IDD_DIALOG1 to the Dialog subfolder of the WebBrowser.rc folder of the WebBrowser icon in the Resource View and then opens a Dialog Editor window for editing the dialog.

26-Visual C++ 7 and 8.

Right-click the body of the dialog in the Dialog Editor and choose Properties.

In the properties window, change the Caption to Web Browser, set the Minimize Box property to True, and change the ID property to IDD_ WEBBROWSER.

Back in the dialog editor window, make the dialog bigger, roughly 12 inches wide by 8 inches tall.

27-Visual C++ 7 and 8.

Right-click the Cancel button and choose Delete.

Drag the OK button to the bottom of the dialog, about a half-inch from the right edge of the dialog.

In the Properties window, change the caption for the button from OK to Kill.

28-Visual C++ 7 and 8.

Drag a new button from the Toolbox palette to the bottom edge of the dialog window, about a half-inch from the left edge of the dialog.

Change the caption property for the new button to Hide and change its ID property to IDC_HIDE.

Now we will add the ActiveX control.

29-Visual C++ 7 and 8.

Right-click near the top/left corner of the dialog editor window, inside the dotted line, and choose Insert ActiveX Control. In the resulting Insert ActiveX Control dialog, choose Microsoft Web Browser and click OK.

Reposition and resize the WebBrowser control so that it fills the area inside the dotted line, above the Hide and Kill buttons.

Now we will add an MFC class for the dialog.

30-Visual C++ 7 and 8.

Right click in the body of the dialog in the blank area between the buttons and choose Add Class.

For the Class Name, enter CWebBrowserDialog.

From the Base Class menu, choose CDialog.

Click the Finish button.

Visual C++ adds the WebBrowserDialog.cpp and WebBrowserDialog.h files to the project. These are starter code files for the dialog.

31-Visual C++ 7 and 8.

Using the Solution Explorer, open the WebBrowserDialog.h file and then add this after the #pragma at the top of the file:

```
#include "resource.h"
```

This is needed so that the IDD_WEBBROWSER symbol will be available in the WebBrowserDialog.cpp and WebBrowserDialog.h files.

Choose File->Save All.

Now we will add a member variable for the web browser control to the CWebBrowserDialog class.

32-Visual C++ 7 and 8.

Right click in the Web Browser ActiveX control in the dialog editor window and choose Add Variable. This displays the Add Member Variable Wizard.

In the Variable Name setting, enter m_WebBrowser.

Click the Finish button.

Visual C++ adds the explorer1.cpp and explorer1.h files to the project. They create the CExplorer1 class.

The explorer1.h file contains declarations of CExplorer1 member functions that you can call to control the web page. For example, we can call Navigate to control which web page is displayed. Later we will create XOP wrapper functions that call these CExplorer1 functions to allow them to be called from Igor.

In addition to creating the explorer1.cpp and explorer1.h files, the Add Member Variable Wizard added a member variable named m_WebBrowser to the WebBrowserDialog class. You can see it in WebBrowserDialog.h. This member variable will represent the Web Browser ActiveX control.

Save and close the source files that VC7 opened.

Close all other windows except for the Solution Explorer.

As it stands, our CWebBrowserDialog class will create a modal dialog. Now we will override some CDialog member functions to make it a modeless dialog. The technique for doing this is explained under "MODELESS Sample: Uses a CDialog Object as a Modeless Dialog Box" in VC7 MFC Samples help section.

33-Visual C++ 7 Only.

Open the WebBrowserDialog.cpp file and choose View->Properties Window.

From the strip of icons near the top of the Properties window, click the sixth one which looks like a non-descript small box rotated 45 degrees.

33-Visual C++ 8 Only.

Open the WebBrowserDialog.cpp file.

Choose View->Class View.

Open the WebBrowser icon in Class View.

Select the CWebBrowserDialog icon, right-click it, and choose Properties.

From the strip of icons near the top of the Properties window, click the sixth one which looks like a non-descript small box rotated 45 degrees.

33-Visual C++ 7 and 8.

We will now change the code for the Create member function so that it takes no parameters.

Create an override member function for the Create member function of the CDialog class by clicking in the empty cell to the right of "Create" in the left column of the Properties window, clicking the triangle icon that appears, and then choosing "<Add> Create" from the resulting popup menu.

This adds code to the WebBrowserDialog.cpp and WebBrowserDialog.h files and opens the WebBrowserDialog.cpp file.

We will now change the code for the Create member function so that it takes no parameters.

34-Visual C++ 7 and 8.

Change the WebBrowserDialog::Create definition in WebBrowserDialog.cpp to this:

```
BOOL CWebBrowserDialog::Create(void)
{
   return CDialog::Create(IDD, NULL);
}
```

Change the Create declaration in WebBrowserDialog.h to this:

```
virtual BOOL Create(void);
```

35-Visual C++ 8 Only.

In the Class View window, select the CWebBrowserDialog icon again.

35-Visual C++ 7 and 8.

Using the Properties window, create an override for the OnCancel function and edit the WebBrowserDialog.cpp file so it looks like this:

```
void CWebBrowserDialog::OnCancel()
{
    DestroyWebBrowserDialog();
}
```

We will create the DestroyWebBrowserDialog function later.

36-Visual C++ 8 Only.

In the Class View window, select the CWebBrowserDialog icon again.

36-Visual C++ 7 and 8.

Using the Properties window, create an override for the OnInitDialog function and edit the WebBrowserDialog.cpp file so it looks like this:

```
BOOL CWebBrowserDialog::OnInitDialog()
{
    CDialog::OnInitDialog();

    m_WebBrowser.Navigate("http://www.wavemetrics.com",NULL,NULL,NULL,NULL);

    return TRUE;
}
```

The Navigate function call invokes the Microsoft Web Browser ActiveX control.

37-Visual C++ 8 Only.

In the Class View window, select the CWebBrowserDialog icon again.

37-Visual C++ 7 and 8.

Using the Properties window, create an override for the PostNcDestroy function and edit the WebBrowserDialog.cpp file so it looks like this:

```
void CWebBrowserDialog::PostNcDestroy()
{
    delete this;  // See "MODELESS Sample" in VC++ MFC Samples help section.
}
```

We need to create two more overrides, for the Kill button, which is the dialog's default button, and for the Hide button.

38-Visual C++ 7 and 8.

Choose View->Resource View.

Open the icons till you see the IDD_WEBBROWSER icon.

Double-click the IDD_WEBBROWSER icon to open the dialog editor.

Right-click the Kill button and choose Properties.

In the Properties window, click the fourth icon (the lightning bolt), which is the Control Events icon.

Add an OnBnClickedOk override for the BN_CLICKED message.

In the WebBrowserDialog.cpp file, edit the OnBnClickedOk function so it looks like this:

```
void CWebBrowserDialog::OnBnClickedOk()
{
    DestroyWebBrowserDialog();
    // OnOK();       // Not for modeless dialogs.
}
```

In the Dialog Editor, right-click the Hide button and choose Properties.

In the Properties window, click the fourth icon (the lightning bolt), which is the Control Events icon.

Add an OnBnClickedHide override for the BN_CLICKED message.

In the WebBrowserDialog.cpp file, edit the OnBnClickedHide function so it looks like this:

```
void CWebBrowserDialog::OnBnClickedHide()
{
    HideWebBrowserDialog();
}
```

Close all open windows except for the Solution Explorer.

Now we will add the DestroyWebBrowserDialog and HideWebBrowserDialog functions as well as other helper functions to create, show and hide the Web Browser modeless dialog window.

39-Visual C++ 7 and 8.

Paste the following code at the bottom of the WebBrowserDialog.cpp file.

```
// Helper routine to run the dialog

CWebBrowserDialog* pWebBrowserDialog = NULL;   // Pointer to dialog object.

/* CreateWebBrowserDialog()
   Returns 0 if the dialog already existed, 1 if it was successfully
   created, 2 if an error occurred while creating the dialog.
*/
int CreateWebBrowserDialog()
{
   if (pWebBrowserDialog != NULL)    // Already exists?
      return 0;

   InitCommonControls();

   AfxEnableControlContainer();

   HWND activeHWND = GetFocus();
```

```
    // See "MODELESS Sample" in VC++ MFC Samples help section.
    pWebBrowserDialog = new CWebBrowserDialog(NULL);
    if (pWebBrowserDialog->Create() == TRUE) {
        if (activeHWND != NULL)
            SetFocus(activeHWND);
        return 1;     // Success.
    }

    return 2;         // Failure.
}

/* ShowWebBrowserDialog()

    Returns 0 if OK, 1 if dialog does not exist.
*/
int ShowWebBrowserDialog()
{
    if (pWebBrowserDialog == NULL)
        return 1;

    pWebBrowserDialog->ShowWindow(SW_SHOW);
    pWebBrowserDialog->SetActiveWindow();

    return 0;
}

/* HideWebBrowserDialog()

    Returns 0 if OK, 1 if dialog does not exist.
*/
int HideWebBrowserDialog()
{
    if (pWebBrowserDialog == NULL)
        return 1;

    pWebBrowserDialog->ShowWindow(SW_HIDE);

    return 0;
}

/* KillWebBrowserDialog()

    Returns 0 if OK, 1 if dialog does not exist.
*/
int KillWebBrowserDialog()
{
    if (pWebBrowserDialog == NULL)
        return 1;

    DestroyWebBrowserDialog();
    return 0;
}
```

```
void DestroyWebBrowserDialog(void)
{
    if (pWebBrowserDialog != NULL) {
        // See "MODELESS Sample " in VC++ MFC Samples help section.
        pWebBrowserDialog->DestroyWindow();
        pWebBrowserDialog = NULL;
    }
}
```

Now we will add the prototypes for the helper functions we just created.

40-Visual C++ 7 and 8.

Paste the following code at the bottom of the WebBrowserDialog.h file.

```
// Helper routines
int CreateWebBrowserDialog(void);
int ShowWebBrowserDialog(void);
int HideWebBrowserDialog(void);
int KillWebBrowserDialog(void);
void DestroyWebBrowserDialog(void);
```

Now we will build the project again to make sure everything is correct.

41-Visual C++ 7 and 8.

Choose File->Save All.

Choose Build->Build Solution.

The build should complete without errors. You may get some link warnings.

Now we have a Web Browser XOP that is capable of creating a modeless dialog containing a web browser, but we have no way to invoke it. Next we will add some XOP external functions so we can invoke the web browser from Igor's command line.

To add an external function we need to do four things:

1. Add an item to the XOPF resource to the XOP WinCustom.rc file. This tells Igor what function we are adding and the types of its parameters and result.
2. Add the function itself to the main XOP .cpp file.
3. Add a prototype for the function to the XOP main .h file.
4. Add an item to the DoFunction function in the main .cpp file. DoFunction is called by Igor through the XOPEntry function when an external function is invoked.

The first step is to open the WebBrowserWinCustom.rc file as a source file, not as a resource file.

42-Visual C++ 7 and 8.

Close all open windows except for the Solution Explorer.

Choose File->Open->File.

Navigate to and select WebBrowserWinCustom.rc.

If you just click the Open button now, the file will open for graphical editing as a resource file. This is not what we want. We want to edit it as a text file.

Click the triangle icon on the right side of the Open button.

From the resulting popup menu, choose Open With.

In the resulting dialog, choose "Source Code (Text) Editor" and click the Open or OK button.

The file opens in the text editor.

43-Visual C++ 7 and 8.

Paste the following text at the bottom of the file:

```
1100 XOPF           // Describes functions added by XOP to IGOR.
BEGIN
    "WebBrowserCreateWindow\0",   // Function name.
    F_UTIL | F_EXTERNAL,          // Function category,
    NT_FP64,                      // Return value type.
        0,                        // NOTE: 0 required to terminate list.

    "WebBrowserShowWindow\0",     // Function name.
    F_UTIL | F_EXTERNAL,          // Function category,
    NT_FP64,                      // Return value type.
        0,                        // NOTE: 0 required to terminate list.

    "WebBrowserHideWindow\0",     // Function name.
    F_UTIL | F_EXTERNAL,          // Function category,
    NT_FP64,                      // Return value type.
        0,                        // NOTE: 0 required to terminate list.

    "WebBrowserKillWindow\0",     // Function name.
    F_UTIL | F_EXTERNAL,          // Function category,
    NT_FP64,                      // Return value type.
        0,                        // NOTE: 0 required to terminate list.

        // More function descriptions can appear here.

    0,                            // NOTE: 0 required to terminate resource.
END
```

This tells Igor that we are adding four external functions. All of the functions return a double-precision numeric result (NT_FP64) and take no parameters.

Save the WebBrowserWinCustom.rc file.

Minimize the WebBrowserWinCustom.rc file so we can easily edit it later.

44-Visual C++ 7 and 8.

Open the WebBrowserXOP.cpp file and add the following code just below the #includes at the top of the file:

```
// *** WebBrowserShowWindow ***

// All structures passed to Igor are two-byte aligned.
#include "XOPStructureAlignmentTwoByte.h"
struct CreateWindowParams {
    // No parameters.
    double result;
};
typedef struct CreateWindowParams CreateWindowParams;
#include "XOPStructureAlignmentReset.h"

static int
WebBrowserCreateWindow(CreateWindowParams* p)
{
    int i;
    int err;

    err = 0;

    // 0=already existed, 1=success, 2=failure.
    p->result = CreateWebBrowserDialog();
    if (p->result == 2)
        err = ACTIVEX_COMPONENT_INIT_FAILED;

    return err;
}

// *** WebBrowserShowWindow ***

// All structures passed to Igor are two-byte aligned.
#include "XOPStructureAlignmentTwoByte.h"
struct ShowWindowParams {
    // No parameters.
    double result;
};
typedef struct ShowWindowParams ShowWindowParams;
#include "XOPStructureAlignmentReset.h"

static int
WebBrowserShowWindow(ShowWindowParams* p)
{
    int i;

    p->result = ShowWebBrowserDialog();  // 0 if OK, 1 if no window exists.

    return 0;
}

// All structures passed to Igor are two-byte aligned.
#include "XOPStructureAlignmentTwoByte.h"
```

```
struct HideWindowParams {
    // No parameters.
    double result;
};
typedef struct HideWindowParams HideWindowParams;
#include "XOPStructureAlignmentReset.h"

static int
WebBrowserHideWindow(HideWindowParams* p)
{
    int i;

    p->result = HideWebBrowserDialog();  // 0 if OK, 1 if no window exists.

    return 0;
}

// All structures passed to Igor are two-byte aligned.
#include "XOPStructureAlignmentTwoByte.h"
struct KillWindowParams {
    // No parameters.
    double result;
};
typedef struct KillWindowParams KillWindowParams;
#include "XOPStructureAlignmentReset.h"

static int
WebBrowserKillWindow(KillWindowParams* p)
{
    p->result = KillWebBrowserDialog();  // 0 if OK, 1 if no window exists.

    return 0;
}
```

For each function we have a structure that defines the parameters that the function receives from Igor and the result that the function returns to Igor.

Igor passes all structures using 2-byte alignment. The default alignment for Windows is 8 bytes. The #include statements ensure that the structures will be two-byte aligned. Without these #include statements, the Igor could not pass parameters to the external function nor get the function result.

To use the helper functions CreateWebBrowserDialog, ShowWebBrowserDialog, HideWebBrowserDialog and KillWebBrowserDialog, their prototypes must be included.

Add the following to the top of the WebBrowserXOP.cpp file, just after the #includes that are already there:

```
#include "WebBrowserDialog.h"
```

Choose File->Save All.

The WebBrowserCreateWindow function uses a custom error code, ACTIVEX_COMPONENT_INIT_FAILED. We defined this error code in the WebBrowserXOP.h file when we created it. We also defined a corresponding error message string in the WebBrowserWinCustom.rc file when we created it.

Next we will add an item to the DoFunction function for each of the external functions that we added.

45-Visual C++ 7 and 8.

In the WebBrowserXOP.cpp file add the four case statements shown here to change the DoFunction function to this:

```
static int
DoFunction(void)
{
    int funcIndex;
    void *p;
    int err = 0;

    try {
        funcIndex = GetXOPItem(0);     // Which function invoked?
        p = (void*)GetXOPItem(1);      // Get pointer to params/result.
        switch (funcIndex) {
            case 0:
                err = WebBrowserCreateWindow((CreateWindowParams*)p);
                break;
            case 1:
                err = WebBrowserShowWindow((ShowWindowParams*)p);
                break;
            case 2:
                err = WebBrowserHideWindow((HideWindowParams*)p);
                break;
            case 3:
                err = WebBrowserKillWindow((KillWindowParams*)p);
                break;
        }
    }
    catch(CException* e) {     // An exception was thrown by MFC.
        char message[256];
        char errorMessage[200];
        e->GetErrorMessage(errorMessage, sizeof(message), NULL);
        sprintf(message, "MFC Exception: %s"CR_STR, errorMessage);
        XOPNotice(message);
        e->Delete();
        err = MFC_EXCEPTION_ERROR;
    }
    catch(...) { // An exception was thrown by MFC.
        err = MFC_EXCEPTION_ERROR;
    }

    return(err);
}
```

There is one more thing to do in the main XOP .cpp file. We need to kill the dialog if it exists when Igor quits.

46-Visual C++ 7 and 8.

Add the following to the switch in the XOPEntry function in WebBrowserXOP.cpp:

```
case CLEANUP:
   KillWebBrowserDialog();
   break;
```

When Igor quits, it passes the CLEANUP message to the XOPEntry function. The KillWebBrowserDialog call will kill the dialog if it exists.

We will now build the XOP again.

47-Visual C++ 7 and 8.

Choose File->Save All.

Choose Build->Build Solution.

The build should complete without errors. You may get some link warnings.

Now we will test the XOP.

48-Visual C++ 7 and 8.

Choose Debug-> Start.

Igor should start up.

48-Visual C++ 8 Only.

Choose Debug-> Start Debugging.

Igor should start up.

48-Visual C++ 7 and 8.

Execute this in Igor's command line:

```
WebBrowserCreateWindow()
```

Nothing evident should happen. If things worked, a modeless dialog with a web browser ActiveX control has been created, but it is invisible.

Execute this in Igor's command line:

```
WebBrowserShowWindow()
```

The modeless dialog should appear and should display the WaveMetrics home page. Try clicking a link or two.

Notice that, although the Web Browser window is the front window, the caret is still blinking in Igor's command line. Also the Web Browser window is always on top of other Igor windows. These problems occur because the Web Browser window is not perfectly integrated into Igor's environment, as explained below.

49-Visual C++ 7 and 8.

Click the Hide button in the Web Browser window.

The web browser is hidden. To get it back, we have to execute a command from the command line.

Execute this in Igor's command line:

```
WebBrowserShowWindow()
```

The web browser is visible again.

Quit Igor.

Now we have an XOP that instantiates an ActiveX control in a modeless dialog window. We have some external functions for creating the window, showing it, hiding it and killing it.

In addition to providing a user interface that is displayed in a window, an ActiveX control provides a number of functions, called methods and properties, to control it programmatically. Methods are routines that provide access the functionality of the control. Properties are routines that get and/or set control properties. Collectively, the set of methods and properties is called an "interface". The WebBrowser ActiveX control provides an interface named IWebBrowser2.

When we inserted a member variable for the ActiveX control in our CWebBrowserDialog class, Visual C++ created the explorer1.h and explorer1.cpp source files. Explorer1.h includes "wrappers" for the methods and properties provided by the WebBrowser ActiveX control. You can see these wrapper functions by opening the explorer1.h file and searching for "IWebBrowser2".

If you were creating an XOP to control an instrument, you would want to create external functions that call the ActiveX control's methods and properties so you could programmatically control the instrument from Igor. To show how this is done, we will now add some external functions to the WebBrowser XOP. We will start by creating a WebBrowserNavigate external function that calls the Navigate method of the ActiveX control.

Since a typical instrument control XOP would add many external functions to Igor, we will put our external functions in a separate source file. This file will provide external functions that map to a subset of the interface provided by the ActiveX control.

50-Visual C++ 7 Only.

In the Solution Explorer window, right-click the Source Files icon and choose Add->New Item.

In the resulting Add New Item dialog, choose Visual C++ from the Categories section and "C++ File (.cpp)" from the Templates section.

Enter WebBrowserInterface.cpp in the Name box and the path to your WebBrowser folder in the Location box.

Click Open.

A new file, WebBrowserInterface.cpp, is created and added to the project.

50-Visual C++ 8 Only.

Right-click the Source Files icon in the Solution Explorer window and choose Add->New Item.

In the Add New Item dialog, select Visual C++ from the Categories section, Code from the list of types under the Visual C++ icon, and "C++ File (.cpp)" from the Templates section.

In the name box, enter WebBrowserInterface.cpp.

In the Location section, enter the path to the WebBrowser project folder.

The WebBrowser project folder is located at "IgorXOPs5\WebBrowser\WebBrowser".

Click Add.

A new file, WebBrowserInterface.cpp, is created and added to the project.

51-Visual C++ 7 and 8.

Open the WebBrowserXOP.cpp file, copy the #include statements at the top of the file, and paste them into the new WebBrowserInterface.cpp file.

Add the following utility function to the file:

```
extern CWebBrowserDialog* pWebBrowserDialog;   // In CWebBrowserDialog.cpp.

static int
GetWebBrowserControlObject(CExplorer1** ppWebBrowser)
{
    *ppWebBrowser = NULL;

    if (pWebBrowserDialog == NULL)
        return ACTIVEX_CONTROLS_PARENT_MISSING;

    CExplorer1* pWebBrowser = &pWebBrowserDialog->m_WebBrowser;
    if (pWebBrowser == NULL)
        return ACTIVEX_CONTROL_MISSING;

    *ppWebBrowser = pWebBrowser;
    return 0;
}
```

This utility function provides access to a pointer to the Web Browser object that we need to call its methods and properties.

Choose File->Save All.

Now we will add an item to our XOPF resource that tells Igor about the external function we are adding.

52-Visual C++ 7 and 8.

If the WebBrowserWinCustom.rc file is minimized, restore it. Otherwise open it as a text file using File->Open->File and the Open With button in the resulting dialog.

The WebBrowserWinCustom.rc file is opened for regular text editing.

Add this text to the XOPF resource, just above the comment that says "More function descriptions can appear here".

```
"WebBrowserNavigate\0",    // Function name.
F_UTIL | F_EXTERNAL,       // Function category,
NT_FP64,                   // Return value type.
    HSTRING_TYPE,          // URL.
    0,                     // NOTE: 0 required to terminate list.
```

The text we added to the XOPF resource tells Igor that our XOP will add a function named WebBrowserNavigate. It will return a double (NT_FP64) and will take one string parameter (HSTRING_TYPE).

Save and minimize the WebBrowserWinCustom.rc file.

Now we will create the function that will be called when WebBrowserNavigate is called.

53-Visual C++ 7 and 8.

Add the following text at the bottom of the WebBrowserInterface.cpp file:

```
#include "XOPStructureAlignmentTwoByte.h"
struct WebBrowserNavigateParams {
    Handle urlH;
    Handle result;
};
typedef struct WebBrowserNavigateParams WebBrowserNavigateParams;
#include "XOPStructureAlignmentReset.h"

int
WebBrowserNavigate(WebBrowserNavigateParams* p)
{
    CExplorer1* pWebBrowser;
    char url[1024];
    int err = 0;

    p->result = 0;

    if (err = GetWebBrowserControlObject(&pWebBrowser))
        return err;

    if (p->urlH == NULL) {
        err = NULL_INPUT_STRING;
        goto done;
    }
    if (GetCStringFromHandle(p->urlH, url, sizeof(url)-1)) {
        err = STR_TOO_LONG;
        goto done;
    }
```

```
    pWebBrowser->Navigate(url, NULL, NULL, NULL, NULL);
    p->result = 0;

done:
    if (p->urlH)
        DisposeHandle(p->urlH); // Dispose input string

    return err;
}
```

The Navigate method takes five parameters. Four of them can be NULL to get default behavior. For simplicity, we are using only the first parameter, the URL, and passing NULL for the other four.

Notice that there are two kinds of results from this function: the function result (err) and the result returned to the calling Igor user-defined function (p->result). If err is non-zero, this will generate an error in Igor and will cause Igor function execution to halt. Therefore we return a non-zero error code as the function result only in the case of a programmer error (e.g., passing an uninitialized string) or some other disaster (e.g., out of memory). We use the Igor function result (p->result) to return an indication of success or failure to the calling Igor user-defined function. The Navigate method of the IWebBrowser2 interface returns an error code but the MFC wrapper, in explorer1.h, does not return this error code, so we punt and always return 0 via p->result.

We have defined the function but we still need to hook it up to Igor.

54-Visual C++ 7 and 8.

Enter the following prototype to the bottom of the WebBrowserXOP.h file:

```
struct WebBrowserNavigateParams;
int WebBrowserNavigate(struct WebBrowserNavigateParams* p);
```

Add this code to the switch statement in the DoFunction function in the WebBrowserXOP.cpp file:

```
        case 4:
            err = WebBrowserNavigate((WebBrowserNavigateParams*)p);
            break;
```

Choose File->Save All.

Now we are ready to try the new WebBrowserNavigate external function.

55-Visual C++ 7 Only.

Choose Build->Build Solution.

The project should compile without error.

Choose Debug->Start.

Igor should start up.

55-Visual C++ 8 Only.

Choose Build->Build Solution.

The project should compile without error.

Choose Debug->Start Debugging.

Igor should start up.

56-Visual C++ 7 and 8.

Execute this in Igor's command line:

```
WebBrowserCreateWindow()
WebBrowserShowWindow()
```

The WaveMetrics home page should appear.

Execute this in Igor's command line:

```
WebBrowserNavigate("http://www.yahoo.com")
```

The Yahoo web page should appear.

Quit Igor.

Next we will add external functions that map to a Web Brower property. We will use the AddressBar property. This is a Get/Set property so we will have to add two external functions to Igor: WebBrowserGetAddressBar and WebBrowserSetAddressBar.

57-Visual C++ 7 and 8.

Open the WebBrowserWinCustom.rc file for editing as a text file (not for editing as a resource file).

Add this text to the XOPF resource, just above the comment that says "More function descriptions can appear here".

```
"WebBrowserGetAddressBar\0",  // Function name.
F_UTIL | F_EXTERNAL,          // Function category,
NT_FP64,                      // Return value type.
    0,                        // NOTE: 0 required to terminate list.

"WebBrowserSetAddressBar\0",  // Function name.
F_UTIL | F_EXTERNAL,          // Function category,
NT_FP64,                      // Return value type.
    NT_FP64,                  // 0 to hide tool bar, 1 to show it.
    0,                        // NOTE: 0 required to terminate list.
```

Save and minimize the WebBrowserWinCustom.rc file.

Now we will create the functions that will be called when WebBrowserGetAddressBar or WebBrowserSetAddressBar is called.

58-Visual C++ 7 and 8.

Add the following text at the bottom of the WebBrowserInterface.cpp file:

```
#include "XOPStructureAlignmentTwoByte.h"
struct WebBrowserGetAddressBarParams {
    double result;
};
typedef struct WebBrowserGetAddressBarParams WebBrowserGetAddressBarParams;
#include "XOPStructureAlignmentReset.h"

int
WebBrowserGetAddressBar(WebBrowserGetAddressBarParams* p)
{
    CExplorer1* pWebBrowser;
    int err = 0;

    p->result = 0;

    if (err = GetWebBrowserControlObject(&pWebBrowser))
        return err;

    p->result = pWebBrowser->get_AddressBar();

done:
    return err;
}

#include "XOPStructureAlignmentTwoByte.h"
struct WebBrowserSetAddressBarParams {
    double value;
    Handle result;
};
typedef struct WebBrowserSetAddressBarParams WebBrowserSetAddressBarParams;
#include "XOPStructureAlignmentReset.h"

int
WebBrowserSetAddressBar(WebBrowserSetAddressBarParams* p)
{
    CExplorer1* pWebBrowser;
    int err = 0;

    p->result = 0;

    if (err = GetWebBrowserControlObject(&pWebBrowser))
        return err;

    pWebBrowser->put_AddressBar(p->value);
    p->result = 0;

done:
    return err;
}
```

We have defined the functions but we still need to hook them up to Igor.

59-Visual C++ 7 and 8.

Add the following prototypes to the bottom of the WebBrowserXOP.h file:

```
struct WebBrowserGetAddressBarParams;
int WebBrowserGetAddressBar(struct WebBrowserGetAddressBarParams* p);

struct WebBrowserSetAddressBarParams;
int WebBrowserSetAddressBar(struct WebBrowserSetAddressBarParams* p);
```

Add this code to the switch statement in the DoFunction function in the WebBrowserXOP.cpp file:

```
        case 5:
            err = WebBrowserGetAddressBar((WebBrowserGetAddressBarParams*)p);
            break;
        case 6:
            err = WebBrowserSetAddressBar((WebBrowserSetAddressBarParams*)p);
            break;
```

Choose File->Save All.

Now we are ready to try the new external functions.

60-Visual C++ 7 Only.

Choose Build->Build Solution.

The project should compile without error.

Choose Debug->Start.

Igor should start up.

60-Visual C++ 8 Only.

Choose Build->Build Solution.

The project should compile without error.

Choose Debug->Start Debugging.

Igor should start up.

61-Visual C++ 7 and 8.

Execute this in Igor's command line:

```
WebBrowserCreateWindow()
WebBrowserShowWindow()
```

The WaveMetrics home page should appear.

Execute this in Igor's command line:

```
Print WebBrowserGetAddressBar()
```

"1" is printed.

Execute this in Igor's command line:

```
WebBrowserSetAddressBar(0); Print WebBrowserGetAddressBar()
```

"0" is printed. This indicates that we can get and set the AddressBar property. However, the address bar does not appear. This is a mystery until you read the documentation for the AddressBar property of the IWebBrowser2 interface. It says:

"The WebBrowser object saves the value of this property, but otherwise ignores it."

Quit Igor.

62-Visual C++ 7 and 8.

If you have persevered through all of these steps, you now have what should be a functional XOP!

Is anybody still reading this? If you are, send a note to support@wavemetrics.com.

Creating Your Own MFC XOP

Now you have seen how to create an MFC XOP that instantiates an ActiveX control. To create your own XOP to use a different ActiveX control, follow the steps listed above but use a name appropriate to your project instead of "WebBrowser".

The WebBrowser XOP sample code includes a help file that is intended to serve as the starting point for your help file.

If you succeed in creating your own XOP, let us know at support@wavemetrics.com so we can tell if it was worth all the time and effort to create this tutorial.

Issues

This section discusses problems and other issues in using MFC to write an IGOR XOP.

MFC Versus MDI

Igor is an MDI ("Multiple Document Interface") application. This means that Igor has a main frame window and all of the other windows are children of the frame window.

MFC is perfectly capable of creating an MDI application, but we were unable to figure out how to create a modeless dialog as an MDI child window in an MFC DLL. Consequently, the WebBrowser modeless dialog window does not behave quite right.

The problems caused by the fact that the modeless dialog is not a proper child window include:

The modeless dialog is always on top of Igor windows.

Igor windows do not properly deactivate when the modeless dialog is activated.

There are probably other problems as well.

At the start of this chapter, we noted that you need Igor Pro 6.02 or later to use the WebBrowser XOP. This is an upshot of the fact that the modeless dialog is not a proper MDI child. When running with earlier versions of Igor, when you activateed and then closed an Igor dialog, Igor became unresponsive. This is because the presence of the non-MDI-child dialog interfered with Igor's normal deactivation and reactivation of the MDI frame window. We worked around this problem in Igor Pro 6.02B01.

Modeless Dialogs

The Web Browser window is a modeless dialog. Adding a Windows modeless dialog to Igor from an XOP is not really supported. The reason for this is that Windows modeless dialogs require special processing of Windows messages in the "message pump" part of the host program. The special processing (calling IsDialogMessage) is responsible for handling certain keystrokes in a special way for a modeless dialog. For example, it causes the tab key to move from one control in the dialog to the next.

Igor does not do this special processing. Consequently, some modeless dialog features may not work as expected.

This issue is explained in Microsoft Knowledge Base article 187988.

The AFX_MANAGE_STATE Macro

MFC associates certain private "state" information with each MFC module. When code in an MFC module is called and in turn calls MFC routines, the module's MFC state must be set as the "current" MFC state. This is explained in MFC Tech Note TN011.

Normally the management of the MFC state is handled entirely by the MFC library. However this is not the case with an MFC DLL. It is up to the DLL author to make sure that the state information is correct. This is achieved by putting the AFX_MANAGE_STATE macro at the start of any routine through which the DLL can be entered.

An XOP can be entered through the following four places:

Through the main function which is called by Igor to initialize the XOP

Through the XOPEntry function, which is what Igor, calls for all messages after initialization

Through a "direct" external function

Through an Operation Manager external operation

In WebBrowserXOP.cpp, you will notice the AFX_MANAGE_STATE macro appears at the start of the main and XOPEntry functions.

The WebBrowser XOP does not add any direct external functions or any external operations, so these two places are the only places where AFX_MANAGE_STATE is required.

Microsoft's documentation on AFX_MANAGE_STATE is somewhat confusing. It appears that the AFX_MANAGE_STATE macro does nothing if the MFC library is statically linked. It is needed if it is dynamically linked.

MFC Exceptions

When you call any MFC routine, it can throw an exception. If you do not catch the exception, this will call a crash. In the DoFunction routine in WebBrowserXOP.cpp, you will see that all of the external functions added by the WebBrowser XOP are called in a try-catch block. This insures that any thrown exceptions will be caught.

There are two catch statements in DoFunction. The first catches thrown CException objects. CException is an MFC class designed to communicate exception information to the catching routine. The second catch statement (catch(…)), catches any other thrown exception data type. This guarantees that a crash will not occur if some routine throws an exception that we are not expecting.

FUNCTION Message Versus Direct-Call External Functions

As noted in the Adding Functions chapter, Igor can call an XOP in two ways. In the WebBrowser XOP, we use the FUNCTION message method. This means that, when one of the XOP's functions is invoked, Igor sends a FUNCTION message to the XOP's XOPEntry routine which in turn calls the DoFunction routine which in turn calls the external function's execute routine.

In the direct-call method, Igor does not pass the FUNCTION message to the XOPEntry routine. Instead, it calls the external function's execute routine directly, using an address previously returned from the XOP to Igor in response to the FUNCADDRESS message.

The direct-call method is faster than the FUNCTION message method. However, in the case of an MFC XOP, we would need to invoke the AFX_MANAGE_STATE macro and handle MFC exceptions in each and every external function execute routine. This would be cumbersome and also would introduce clutter into the WebBrowser example.

For most purposes, the overhead of the FUNCTION message method is inconsequential. Therefore, most MFC XOPs should use that method. If you feel that you need the utmost speed, you can use the direct-call method but you need to remember to use the AFX_MANAGE_STATE macro and handle MFC exceptions in each direct-call external function that you add.

Error Codes

An external function's C++ execute function returns two types of results. The first is the execute function's direct result. The second is the external function result. For example:

```
int
Example(ExampleParams* p)
{
    int err = 0;

    p->result = 0;

    if (p->inputStringParameter == NULL) {
        err = NULL_INPUT_STRING;
        goto done;
    }

    // p->result is the external function result which goes
    // to the calling user-defined function.
    p->result = Instrument(...);

done:
    if (p-> inputStringParameter)
        DisposeHandle(p-> inputStringParameter);    // Dispose input string

    // err is the execute function's direct result. It goes to Igor.
    return err;
}
```

Example is the external function's execute function. That is, it is the C++ function that is called when the user invokes the external function from within Igor.

err is the direct function result. It goes back to Igor. If it is non-zero, Igor stops execution of user procedures and displays an error message. All error codes returned as the direct function result must be either a built-in Igor error code, as defined in IgorXOP.h, or a custom XOP error code, as defined in the XOP's STR#,1100 resource.

p->result is the result that goes back to the calling Igor user-defined function. Often it represents an error code that is zero for no error or non-zero in the event of an error. If p->result is non-zero, the calling function can choose to do a retry, try a different call, or display an error message.

In real life you would most likely create an MFC XOP to interface Igor to an instrument that is supplied by the manufacturer with an ActiveX control. Typically the instrument control methods provided by the ActiveX control will return their own error codes. You therefore need to return two kinds of error information to the caller: Igor error codes (via the direct function result) and instrument error codes (via p->result). An error code returned via p->result to the calling user-defined function can have any meaning you want. For example, it might be an error code defined by the instrument manufacturer.

The recommended approach is to return programming errors or catastrophic errors (e.g., out-of-memory) as the direct function result but return errors from the instrument as the external function result. The error codes returned as the external function result will be exactly the codes returned to you by the instrument control software. If you want the Igor procedure code to be able to display a meaningful message for instrument errors, you will need to provide an external function that returns an error string for each of the instrument control errors.

References

The completed XOP projects:
<ftp://ftp.wavemetrics.net/IgorXOPToolkit/Examples/MFCXOP_VC7.zip>
<ftp://ftp.wavemetrics.net/IgorXOPToolkit/Examples/MFCXOP_VC8.zip>

Overview of MFC
<http://en.wikipedia.org/wiki/Microsoft_Foundation_Classes>

TN011: Using MFC as Part of a DLL
<http://msdn2.microsoft.com/en-us/library/zfz4xb9a(VS.80).aspx>

Reusing the WebBrowser Control
<http://msdn2.microsoft.com/en-us/library/Aa752044.aspx>

Using MFC to Host a WebBrowser Control
<http://msdn2.microsoft.com/en-us/library/aa752046.aspx>

TN058: MFC Module State Implementation
<http://msdn2.microsoft.com/en-us/library/ft1t4bbc(VS.80).aspx>

Modeless Sample: Uses a CDialog Object as a Modeless Dialog Box
<http://msdn2.microsoft.com/en-us/library/zhk0y9cw(VS.80).aspx>

Modeless Dialog Issues
<http://support.microsoft.com/kb/q187988/>

Handling Events Sent by an ActiveX control
HOW TO: Create a Sink Interface in MFC-Based COM Client (KB 181845)
<http://support.microsoft.com/default.aspx?scid=kb;en-us;181845>