Exercise 7 Automation

In this exercise you will be working with a pre-written COM object that has a dual interface. You will instantiate the object in a web page and then invoke its methods using HTML and VB Script. The COM object is a graphical control called ***Fire***. The control is quite good fun to play with, so you'll have a chance to host it in a Visual Basic form before you write your web page.

This exercise illustrates various aspects of web programming:

- creating an HTML page
- adding object tags
- adding form tags
- adding buttons and labels to a web page
- writing VB script

Step 1 - Getting familiar with the Fire Control Object

Open the workspace:

```
C:\COM Programming\Exercises\Automation\Automation.dsw
```

This workspace has 2 projects defined. The ***Fire*** project is already complete, but you will modify the Test project later. For now build and run the ***Fire*** project to get familiar with the control. When you have finished playing with the control in Visual Basic, investigate the properties and methods of the control in Visual Basic's object browser (View/Object Browser).

Step 2 - Creating the Web Page

Swap projects to the ***Test*** project and then add a new HTML page to the project using ***File/New File/HTML Page***. Call your web page ***Page1.htm***. Before you can use the page you will have to add various HTML tags and some VB script.

Step 3 - Adding the Fire Control to the Web Page

In order to add the ***Fire Control*** to the Web Page you must define an appropriate object tag. The easiest way to do this is to copy the object tag from the ***Tools/OLE COM Object Viewer*** product that is bundled with the Visual C++ development suite.

On the Tools menu select:

```
Tools/OLE COM Object Viewer
```

and then expand the ***All Objects*** branch. Scroll down to the ***Fire Control*** object. Make sure you have selected the correct object by checking the ***Implementation*** tab. Note that objects are categorised according to the name given to the CLSID at the time the control was registered. You can determine this name by looking up the CLSID in the registry.

With the ***Fire Control*** selected, hit the right mouse button. Choose the last option:

```
Copy HTML <object> Tag to Clipboard
```

Go the HTML page and then paste in the object tag into ***Page1.htm*** where it says

```
"Insert HTML here"
```

Save the file.

Step 4 - Using Internet Explorer

Since we intend to use Internet Explorer as our test harness, we need to modify the project settings accordingly. Choose ***Project/Settings/Debug*** and select the default web browser as the executable for the debug session. We still need to specify the web page as a parameter to IE and you must specify the full path of web page as program arguments (quote the pathname because it contains spaces). A quick method to obtain the path is to use the Window Explorer and hit right button on ***Page1.htm*** to get its properties; this will display the full path. Copy and paste the pathname to the project settings tab.

Run to test (F5). This should bring up the ***Fire Control*** in a web page. You won't see too much at present - just a black screen.

Step 5 - Writing the test harness Q1

We need to add some buttons to the web page to drive the ***Fire Control***. In order for these buttons to reference the control we must give the control a name. So modify the object tag as follows:

```
<object
        ID="FireControl" HEIGHT=300 WIDTH=300
        classid="clsid: ... "
>
</object>
```

To add the buttons themselves, first add a form to the web page. You need to add a form tag after COM object tag:

```
<Form Name = "Form1">
</Form>
```

Then add 3 buttons to the form. Use one button to create the fire, one to turn it up and one to turn it down. You can add a button with code like the following:

```
<Form Name = "Form1">
        <Input Type ="Button" Name="Up" Value="Up"
                onClick="Up_Click" Language="VBScript">
</Form>
```

The onClick entry specifies the name of a subroutine in the VB script that you have still to write. Repeat for the other buttons.

Add a label to display the temperature of the control:

```
<Input Type="Label" Name="Temperature" Value="0"
        Language="VBScript">
```

Run up IE and check that the buttons and the label appear on the web page. The buttons won't work of course - not until you write the VB script to handle the click events.

Step 8 - Adding the VB Script

Add script code immediately after the form tag. Enclose your script between

```
<SCRIPT LANGUAGE="VBScript">
</SCRIPT>
```

tags.

Add routines to control the fire:

```
Sub Up_Click
        FireControl.Up
End Sub

Sub Down_Click
        FireControl.Down
End Sub

Sub CreateFire_Click
        FireControl.CreateFire
        Form1.CreateFire.Disabled = True
        Form1.Up.Disabled = False
        Form1.Down.Disabled = False
End Sub
```

Step 9 - Displaying the Temperature of the Fire

Run to test (F5). This should bring up the ***Fire Control*** in a web page. Everything is working except the ***Temperature*** label. Getting this to work is a little tricky.

We need to update the label on a regular basis and therefore we need to enlist the services of a timer object that we have provided for you. Register (with ***REGSRV32***) the ***QATimer.DLL*** that is provided in the ***COM Programming\Exercises\Automation*** directory.

Now, look up the ***Timer Control*** in the ***OLE/COM Viewer*** and then add an object tag for this control before other the object tag but after the ***BODY*** tag. Give the control an id as follows:

```
ID="TimerControl"
```

To initialise the control add

```
TimerControl.Enabled = True
TimerControl.Interval = 1000
```

to the ***CreateFire_Click*** subroutine.

Update the label display periodically by defining a ***Timer*** event for the ***Timer Control***:

```
Sub TimerControl_Timer
        Form1.Temperature.Value = FireControl.Temperature
End Sub
```

where ***Form1.Temperature*** identifies the label that display the temperature of the fire.

Run to test (F5) one last time. Everything should now be working correctly.

<<SubExercise>>Automation (Optional)

In this exercise you will be using the ATL to create a COM object with a dual interface. You also have to write a test harness that calls a method in the object using early and late binding.

Step 1 - Getting Started

Create a workspace in the directory:

```
COM Programming\Exercises\Automation (Optional)
```

called ***Automation (Optional)***. Within this workspace create an ATL DLL project called ***Server***.

Step 2 - Creating the COM Server

Working in the server project, create a new ***Simple*** object using the ATL Object wizard. Call the object ***Automation*** and make sure you leave the ***Dual*** option checked in the attributes dialog.

Add the following method to the ***IAutomation*** interface:

```
HRESULT Hello([in] BSTR message);
```

Step 3 - Implementing the Server

Keep the implementation of the ***Hello*** method as simple as possible. We suggest you use:

```
STDMETHODIMP CAutomation::Hello(BSTR message)
{
        MessageBoxW(0, message, L"Hello", MB_OK);
        return S_OK;
}
```

Step 4 - Creating the Test Harness

Add a console-based application to your workspace called ***Test***. This should be a simple project, which will have a single C++ source file with ***main()*** already provided. The test harness should create the server and then call the ***Hello*** method using both early and late binding.

Coding the test harness is lengthy but straightforward. If you prefer you can follow the instructions below:

1. Accessing the Server's Type Information

Include the files ***Server.h*** and ***Server_i.c*** at the top of you client

2. Initialise COM

Don't forget to call ***CoInitialize*** before you call any COM functions.

3. Create Class Factory

Call ***CoGetClassObject*** to create the class object and the ***CreateInstance*** method to create the server.

4. Query for IDispatch

Use ***QueryInterface*** to obtain a pointer to the server's ***IDispatch*** interface ,

5. Check if Type Information is Present

Use ***GetTypeInfoCount*** to check the type information is available. This call should always succeed if the server has been built with the ATL. If it fails you've got problems!

6. Use Late Binding

Use the IDispatch pointer to make a late binding call. You will need to call ***GetIDsOfNames*** first. This method requires an ***LCID*** parameter; you can use

```
LCID English = MAKELCID(LANG_ENGLISH, SUBLANG_ENGLISH_UK);
```

Set up the array of names as an array of OLECHAR:

```
OLECHAR* ArrayOfNames[1];
```

and fill in the method name:

```
ArrayOfNames[0] = L"Hello";
```

You will get back an array of dispatch IDs. Use the dispatch IDs to call ***Invoke***. Note that ***Invoke*** requires a ***DISPPARAMS*** structure to describe the parameters for each method. Set up this structure with:

```
VARIANT v;
v.vt = VT_BSTR;
v.bstrVal = message;
DISPPARAMS NoArgs = {&v, 0, 1, 0 };
```

where ***message*** is the parameter passed.

7. Use Early Binding

Use ***QueryInterface*** to get a pointer to the ***IAutomation*** interface and then call the ***Hello*** method using early binding.

8. Release Pointers and Shutdown

Now release all interface pointers and call ***CoUninitialize***.

Step 5 - Testing Performance

Once you get your test harness working you might like to compare the performance of early and late binding. We suggest that you call the methods of your COM in loop (about 1,000,000 iterations should do the trick). Obviously, you will need to comment out the code to display the message box in your method calls before running the harness. What performance gains do you get with early binding?