Exercise 10 COM EXE Servers

This exercise uses a simple ATL control to illustrate marshalling. In this lab, we will

- create an EXE based COM object with an interface that needs to be marshalled
- use the Automation marshaller to marshal the interface
- use the MIDL marshalling code to marshal the interface

Step 1 - Setting up the COM Object

Fire up VC6 and create a new ATL COM Wizard project in the directory:

```
COM Programming\Exercises\COM EXE Servers
```

Choose an EXE project with a name of ***Marshalling***

Add a new ATL object from class view. Choose ***Object, Simple Object*** and in the next dialog name the object ***Simple.*** Don't change any of the attributes for the object, but note that the object will be created with as a dual object. Thus the object will support ***IDispatch*** and can therefore use the Universal Marshaller to marshal its interface.

Build the project.

Step 2 -Adding Methods to the ISimple Interface

Add the following methods to the ***ISimple*** interface:

```
DoubleIt([in, out] long* pNumber);
DisplayMessage([in] BSTR theMessage);
```

and then provide the following implementations:

```
STDMETHODIMP CSimple::DoubleIt(long *pNumber)
{
	*pNumber = (*pNumber) * 2;
	return S_OK;
}
```

```
STDMETHODIMP CSimple::DisplayMessage(BSTR theMessage)
{
	MessageBoxW(0, theMessage, L"ISimple", MB_OK);
	return S_OK;
}
```

Step 3 - Testing with a Console Based Application

We intend to test our COM object using some raw C++ code, so add a console based application to the current workspace: in ***File View*** right click on the workspace icon and select ***add new project to workspace.*** Choose ***Win32 Console Application*** with a project name of ***Test.***

Create a new file in this project called ***Main.cpp*** (use ***File/New/Files/C C++ Source File***) and then add the code below:

```
#include <windows.h>
#include <assert.h>
#import "..\Marshalling\Marshalling.tlb"
#include <iostream>
using namespace std;

int main()
{
        return 0;
}
```

Now build the application. Pay special attention to the ***import*** directive. This directive de-compiles the type library of our COM object and creates two source files with code defining the COM object and its interfaces and a set of wrapper methods. This allows us to use all the symbols defined in the ATL project in our test harness.

Take a look in the ***Debug*** directory and you will see the two files:

```
Marshalling.tlh
Marshalling.tli
```

If you examine the first file you will find symbols defined for the COM object and all its interfaces. Note that everything is defined in the namespace ***MARSHALLINGLib***, so to refer to the ***ISimple*** interface you must write:

```
MARSHALLINGLib:: ISimple* p;
```

or use the ***using namespace*** construct:

```
using namespace MARSHALLINGLib;
ISimple* p;
```

The second file contains wrapper methods for your interface. These methods are essentially the same as those provided by scripting languages such as VB and Java.

Note: the ***using namespace std*** statement must not precede the import directive, or your code will not compile. This is a bug in the compiler.

Step 4 - Adding Code to the Test Application

Testing is along the lines we followed earlier. Add the following code to the ***Main.cpp*** file:

```
// 1. Declare a interface pointer to the object
        HRESULT hr;
        ISimple* pISimple = 0;

// 2. Initialise COM
        hr = CoInitialize(0);
        assert(SUCCEEDED(hr));

// 3. Create Object
        hr = CoCreateInstance(__uuidof(Simple),
                              0,
                              CLSCTX_LOCAL_SERVER,
                              __uuidof(ISimple),
                              (void**)&pISimple);
        assert(SUCCEEDED(hr));

// 4. Call methods
        BSTR hello = SysAllocString(L"Hello");
        pISimple->DisplayMessage(hello);

        long x = 100;
        pISimple->DoubleIt(&x);
        cout << "x = " << x << endl;

// 5. Release pointers
        pISimple->Release();

// 6. Unload COM
        CoUninitialize();
```

Note the use of the ***__uuidof*** construct to regenerate GUIDs and UUIDs. Test out the COM object with the above code. Although the COM object is a separate executable the marshalling code connects the test harness to the EXE. The marshalling code is of course the Automation marshaller, as can be verified by inspecting the registry.

Look up the ***ISimple*** interface to find the classid of its ***ProxyStubClsid32***. You should find the classid is:

```
{00020424-0000-0000-C000-000000000046}
```

Use this classid to find the Marshalling object and look at the ***InprocServer32*** entry. There you should find

```
oleaut32.dll
```

which is the Universal Marshaller.

Step 5 - MIDL Marshalling

Repeat the above exercise, but this time you will create a COM object that is not a dual object (select Custom on the attributes tab). If you use the test harness above you will find the program asserts when you attempt to create the COM object. If you investigate the ***HRESULT*** for this error you will find:

```
No such interface supported
```

This is referring to the marshalling code. If you look up the ***ProxyStubClsid32*** entry you will find that you are using the MIDL generated marshalling code. The only trouble is that this code is not registered!

The MIDL compiler has placed all the marshalling code in the files:

```
dlldata.obj Marshalling_p.obj Marshalling_i.obj
```

and created a make file to build the marshalling proxy/stub DLL.

```
Marshallingps.mk
```

You can build and register the proxy/stub COM object on the command line:

```
nmake Marshallingps.mk
regsvr32 Marshallingps.dll
```

Now test the object once again. This time everything should be OK.