Exercise 4 Introduction to IDL

This exercise will give you experience of writing and using IDL. In this lab, we develop a client/server application where the client calls a number of methods on the server. These server methods use most of the common parameter passing paradigms that we will meet in COM. In particular, we will be passing arrays and structures between client and server and investigating memory allocation and de-allocation.

Step 1 - Getting Started

Open the workspace:

```
C:\COM Programming\Exercises\Introduction to IDL\Introduction to IDL.dsw
```

Observe that the workspace defines 2 projects: ***Component*** (server) and ***Test*** (client). The ***Test*** project is complete, you will be defining IDL and implementing methods in the in the ***Component*** project. Take some time to familiarise yourself with the files in each project.

You will need to modify:

File	Project	Purpose
Interface.idl	Component	IDL definitions
Component.cpp	Component	Implementation of Server methods

The other files are complete and do not require modification. In this lab we will be writing several methods in the server and calling them from the client.

Step 2 - Defining the IDL

Before the client can call any methods in the server we must define the names and parameters of these methods and we must also define the marshalling strategy. All these definitions must be placed in the IDL file (***Interface.idl***). Microsoft's MIDL compiler will process the IDL. The MIDL compiler generates several intermediate files that must be linked in to the client. We've set up the projects to handle all these configuration issues. All you have to do is write the IDL. ***Don't write any code just yet – wait until Step 3***.

Study the client code in ***Test.cpp***. Notice that the client creates a COM object with 3 interfaces and then calls 6 different methods. The IDL must exactly match these client calls. Notice that you need 2 methods on each interface.

Step 3 - Implementing the Server Methods

After you've written the IDL, you will need to implement the methods in the server. It would be a good idea if you studied the client code to get some idea of what you have to do in the server. We are going to develop the IDL and server code for several test cases. Full instructions are given below:

Test 1

Here is an easy one to start. The client calls a function in the server called ***Test1*** with a single input parameter of type ***long***:

```
long number = Test1(100);
```

The server should display this input in a message box. Write the IDL for this test in ***Interface.idl*** and then implement the server method in ***Component.cpp***. Comment out all the other tests in the client and the server and then build and test. You will need to comment out the IDL, the method prototypes and the method implementations in the server and the method calls in the client.

Test2

This time the method is called ***Test2***. The method takes a pointer to a ***date*** structure and initialises it with the last day of the second millennium. You will need to define the date structure in the IDL file:

```
struct date
{
	int day;
	int month;
	int year;
};
```

The memory for a date structure is allocated on the stack in the client and is then passed as a pointer to the server:

```
date theDate;
Test3(&theDate);
```

Since the server is not interested in the initial value of the date structure you should pass the pointer as an ***output*** only parameter. The pointer is non-null and you are not using pointer aliasing so use the correct attribute from:

```
ref
unique
ptr
```

Remove the comments from the client and server code. Build and test.

Test3

This example shows how properties are used. A property is some state information maintained by the server and access from the client by a pair of methods: ***get_Test3()*** and ***put_Test3()***. The client code looks like:

```
long x = 66;
put_Test3(x);                 // store x on the server
x = 0;                        // reset x on the client
get_Test3(&x);                // retrieve x from the server
```

In the IDL define 2 methods called ***Test3***. In the first method use the ***propget*** attribute and in the second use the ***propput*** attribute. Both methods take one parameter; the property get method uses an output only parameter and the property put method uses an input only parameter. Property get parameters must also use the retval attribute. The IDL should look something like:

```
[propget] HRESULT Test3([out,retval] ...);
[propput] HRESULT Test3([in] ...);
```

If you study the server class file (***Component.h***) you will notice a private member variable called ***property*** that you can use to hold the property value in the implementations of both functions.

Remove the comments from the client and server code. Build and test.

Test 4

Now try your hand at strings. The ***Test4*** method should take one parameter of type ***char****. When you write the IDL pass the character array as an input only parameter and use the ***string*** attribute so that MIDL knows the array is null terminated. In the method implementation you must use ***unsigned char**** as the parameter type!

The method should display the input string in a message box.
The server code looks like:

```
MessageBox(0, (char*)s, "Test4", MB_OK);
```

The client code looks like:

```
Test4((unsigned char*)"Hello");
```

Note you should use ***char**** in the IDL and ***unsigned char**** in the method implementation. Remove the comments from the client and server code. Build and test.

Test 5

In this test we are going to pass a pointer to a date structure. This time the memory for structure will be allocated in the server and deallocated in the client. The client code looks like:

```
date* p = 0;
pCCC->Test5(&p);               // server allocates memory
CoTaskMemFree(p);              // client deallocates memory
```

This time the pointer is null, but you are still not using pointer aliasing. So which attribute should you use?

```
ref
unique
ptr
```

In the server, set the date to the last day in the second millennium. Use the debugger to check that the data gets sent back to the client.

Implementing the server is a little tricky. What level of indirection should you use for the pointers? Remove the comments from the client and server code. Build and test.

Test 6

In the last test the server method is called ***Test6*** and takes 4 parameters and returns ***void***. This test demonstrates how to pass part of a variable size array from the client to the server where its elements are modified. Multiply by two the elements that are transferred to the server.

The 4 parameters are defined as follows:

Parameter	Name	Description
1	array[]	variable size array defined in the client and modified in the server
2	size	the size of the array
3	first	the first element transmitted
4	length	the total number of elements transmitted

The client code that calls this method is:

```
long array[10] = { 999, 999, 1, 3, 5, 7, 9, 999, 999, 999 };
Test6(array, 10, 2, 5);   // 10 elements, 5 transmitted, starting with element 2
```

You will need pass the array as an ***input-output*** parameter and use the ***size_is***, ***first_is*** and ***length_is*** attributes in the IDL. Use the debugger to check that the array contains:

```
{ 999, 999, 2, 6, 10, 14, 18, 999, 999, 999 }
```

after the call.

Hint: The array parameter should be declared as:

```
[in, out, size_is(size), first_is(first), length_is(length)] long array[]
```