Exercise 9 Connection Points

In this exercise you will use the ATL wizard to create a COM object that supports connection points. You will be writing some Visual Basic code to test your COM object.

The COM object will have one conventional interface

```
IDice          RollDice()
```

and one outgoing interface supporting connection points:

```
_IDiceEvents   SixThrown(),  DoubleThrown()
```

Step 1 - Getting Started

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

```
COM Programming\Exercises\Connection Points
```

Choose a DLL project with a name of ***ConnectionPoints***.

Step 2 - Creating the COM Object

In class view, right click on the mouse and select the ***New ATL Object*** option and select a ***Simple Object***. Choose a short name of ***Dice*** on the ***names*** tab and make sure ***Support Connection Points*** is selected from the ***attributes*** tab.

Step 3 - Defining the Methods

Select the ***IDice*** interface in class view and right click the mouse. Add the ***RollDice*** method to the interface:

```
RollDice( [out] short* pDice1 ,  [out] short* pDice2 )
```

Build the COM object to check everything is OK. Notice that the wizard has created the ***_IDiceEvents*** interface for the connection points.

Step 4 - Defining the Callbacks

Select the ***_IDiceEvents*** interface in class view and right click the mouse. Add the two callback events:

```
SixThrown          no parameters
DoubleThrown       no parameters
```

Before going any further, you must build your control. In the next stage we will be using the wizard to generate the callback event code. This wizard will read your type library to generate this code and if you haven't built your control this information will be unavailable and the generation will fail.

Step 5 - Implement Connection Point

Now select the ***CDice*** object in class view and click right button. Select ***Implement Connection Point ...*** and then check the ***_IDiceEvents*** interface in the ensuing dialog. After you select ***OK***, the wizard will generate the connection point code. If you want to inspect the generated code, take a look in ***CProxy_IDiceEvents*** class. Notice that two methods have been defined:

```
Fire_SixThrown
Fire_DoubleThrown
```

Build the project. Sometimes the wizard generates an incorrect entry in the connection map:

```
CONNECTION_POINT_ENTRY(IID__IDiceEvents)
```

and you get several errors in the build. If necessary, correct the entry to read:

```
CONNECTION_POINT_ENTRY(DIID__IDiceEvents)
```

and everything should now build OK.

Step 6 - Writing the RollDice Method

Create 2 member variables in the ***CDice*** class:

```
short dice1;
short dice2;
```

Then implement the ***RollDice*** method with the following code:

```
STDMETHODIMP CDice::RollDice(short* pDice1, short* pDice2)
{
	dice1 = *pDice1 = rand() % 6 + 1;
	dice2 = *pDice2 = rand() % 6 + 1;

	return S_OK;
}
```

Step 7 - Calling the Outgoing Interfaces

We need to inform all interesting parties when a six or double is thrown, through the ***_IDiceEvents*** interfaces. So add the following code to the end of the ***RollDice*** method:

```
if (dice1 == 6 || dice2 == 6) Fire_SixThrown();
if (dice1 == dice2) Fire_DoubleThrown();
```

Build the project to make sure everything is OK. Before continuing, take a moment to study the IDL file and see how the connection point interface is defined.

Step 8 - Testing with VB6

Since we intend to test our COM object using Visual Basic we must adjust the project settings accordingly. Select ***Project Settings/Debug*** and then add VB6 as the executable for the debug session (you should browse the disk to get the correct path name). Now press run (F5) and Visual Basic should fire up. Choose a ***standard exe*** project and immediately save it with the default form (***Form1.frm***) and project (***Project1.vbp***) names.

Close down Visual Basic and return to the debug settings. Add ***Project1.vbp*** to the Program Arguments box and then press run (F5) again. This time Visual Basic should fire up with your new project already loaded.

Step 9 - Writing the VB6 Code

To use our COM object in Visual Basic, select ***Project References*** and then search for Connection Points 1.0 Type Library. Check the type library.

In the form's code module declare a variable to represent the COM object's default interface:

```
Option Explicit
Public WithEvents theDice As CONNECTIONPOINTSLib.Dice
```

The ***WithEvents*** keyword informs VB that this interface is related to a connection point interface. VB will read the type library and determine the names of all callback methods.

In the ***form load*** procedure add the code to create the COM object:

```
Private Sub Form_Load()
 Set theDice = CreateObject("ConnectionPoints.Dice.1")
End Sub
```

where ***"ConnectionPoints.Dice.1"*** is the PROGID of our object.

Add a button to the form and give it the caption ***"Throw Dice"***. Add the following code to the button click procedure:

```
Private Sub Command1_Click()
 Dim dice1 As Long
 Dim dice2 As Long
 theDice.RollDice dice1, dice2
 Print dice1, dice2
End Sub
```

Now run the VB project and the results of the dice throws should be displayed on your form.

Step 10 - Adding the Events Code

The final step is to add the code for the two events. In the code window you will find two drop down boxes. In the left-hand box select ***theDice***. In the right box, two options are available, one for each event. Select each option in turn. VB will write stub procedures for you. Add code to these procedures as follows:

```
Private Sub theDice_SixThrown()
 MsgBox "Six Thrown"
End Sub
```

```
Private Sub theDice_DoubleThrown()
 MsgBox "Double Thrown"
End Sub
```

Now run the project again. The dice throws are displayed as before, but this time a message box is displayed for each event.