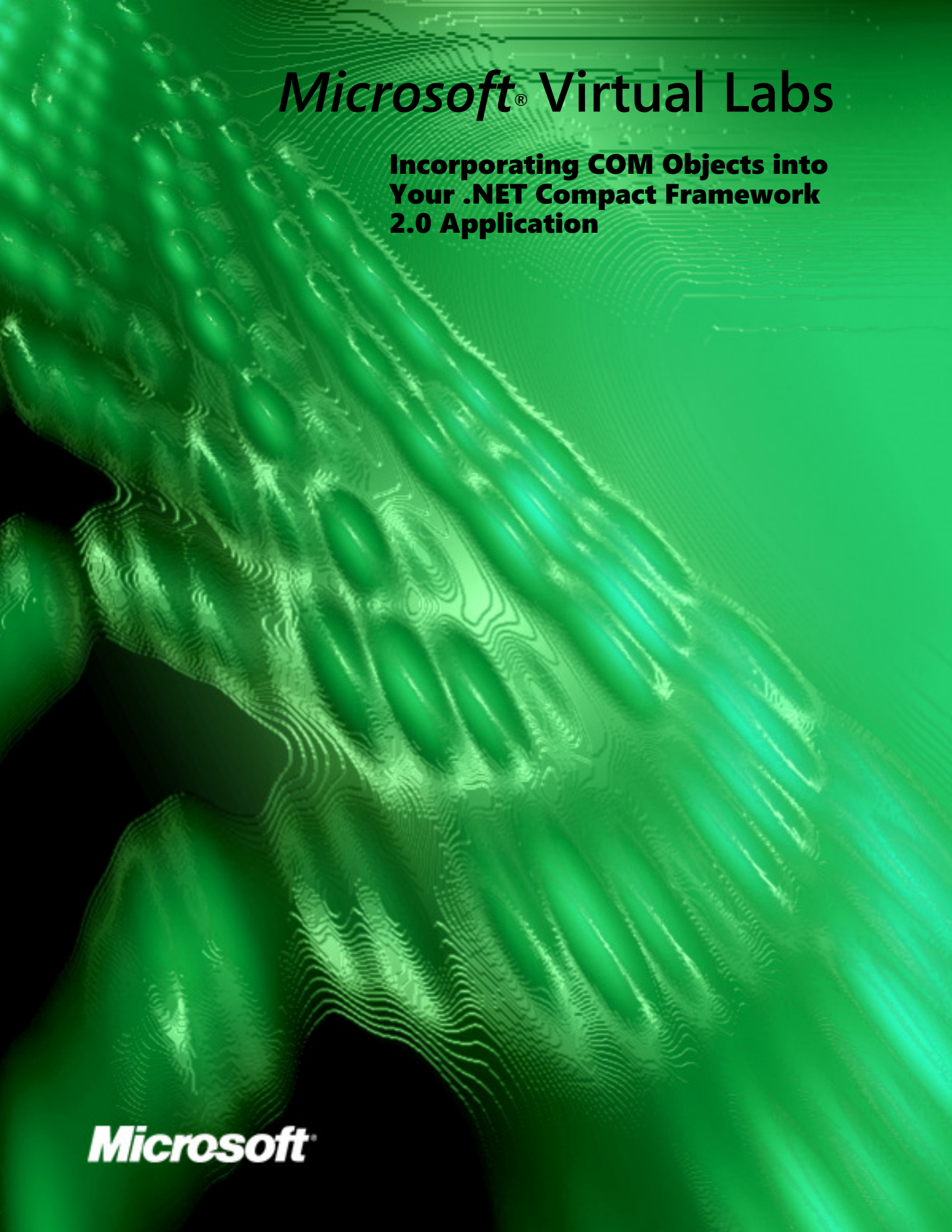


The background is a vibrant green with a complex, abstract pattern. On the left side, there is a dark, shadowed shape that resembles a hand or a series of fingers reaching out. The rest of the background is filled with intricate, wavy, and concentric lines that create a sense of depth and movement, similar to a topographical map or a digital signal visualization.

Microsoft® Virtual Labs

**Incorporating COM Objects into
Your .NET Compact Framework
2.0 Application**

Microsoft®

Table of Contents

Incorporating COM Objects into Your .NET Compact Framework 2.0 Application	1
Exercise 1 Creating and Using a COM Object by Using Visual Studio 2005	2
Exercise 2 Accessing Pocket Outlook Inside a Managed Application	20
Exercise 3 Accessing Pocket Outlook by Using Windows Mobile 5.0 Managed APIs.....	33
Summary.....	42

Incorporating COM Objects into Your .NET Compact Framework 2.0 Application

Objectives

After completing this lab, you will be better able to:

- Creating and using a COM object by using Visual Studio 2005
- Accessing Pocket Outlook inside a managed application
- Accessing Pocket Outlook by using Windows Mobile 5.0 managed APIs

Scenario

In this lab, you will learn how you can use the .NET Compact Framework version 2.0 to incorporate your existing native COM objects into managed applications. Through a series of examples, you will learn how to prepare your COM objects, incorporate them into your .NET Compact Framework projects, and call them from managed code. Upon completion of this lab, you will know how to avoid rewriting all of your legacy COM objects by using them directly in your .NET Compact Framework applications.

Because completing all exercises in the lab may take more time than you have available, you may select only those exercises that are most useful for you. Each exercise starts with a completely new project, and there is no explicit order in which you have to do the exercises.

Estimated Time to Complete This Lab

75 Minutes

Computer used in this Lab



MEDC

Exercise 1

Creating and Using a COM Object by Using Visual Studio 2005

Scenario

In this exercise, you will create a COM object that you will use from within a .NET Compact Framework 2.0 application. This exercise is the starting point for this section of the HOL, in which you will build a complete Pocket PC application through a number of exercises. As you will find out, you can use Visual Studio 2005 to create managed .NET Compact Framework applications and native C++ applications for devices—even with ATL or MFC support.

Tasks	Detailed Steps
1. To open an existing solution and add a new project to it	a. On the desktop computer, click Start All Programs Microsoft Visual Studio 2005 Microsoft Visual Studio 2005. b. In Visual Studio 2005, click File Open Project/Solution. c. In the Open Project dialog box, browse to C:\Program Files\Windows Mobile Developer Samples\HOLs\HOL301_Incorp_COM_Obj_NETCF2_Apps\HOL301Exercise1. d. Select HOL301Exercise1.sln, and then click Open. The solution opens. e. In Solution Explorer, right-click Solution 'HOL301Exercise1' (1 project), and then select Add New Project to add a new project to the solution. f. In the Add New Project dialog box under Project types, ensure that Other Languages Visual C++ Smart Device is selected. <i>Note: Depending on your Visual Studio configuration, the Visual C++ project type may be located at the outer-most outline level instead of under Other Languages.</i> g. Under Templates, ensure ATL Smart Device Project is selected. h. In the Name box, change the name to MyCOMObject, as shown in Figure 1.

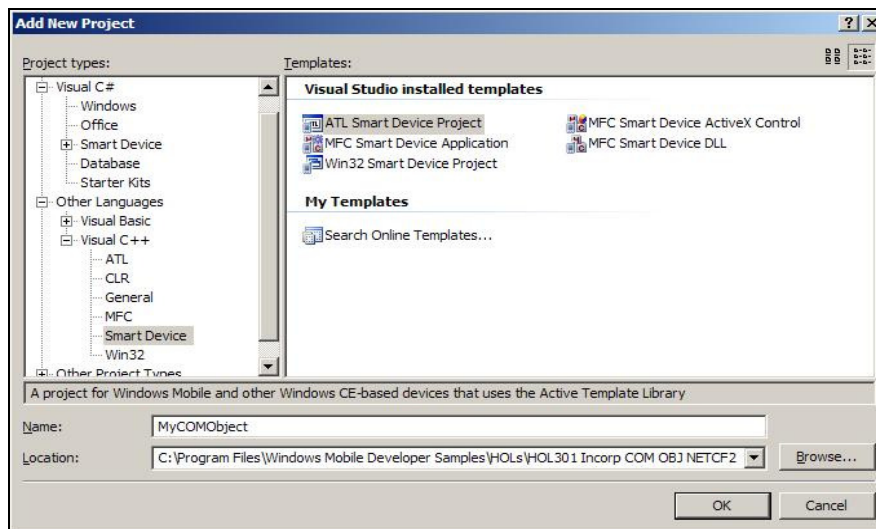
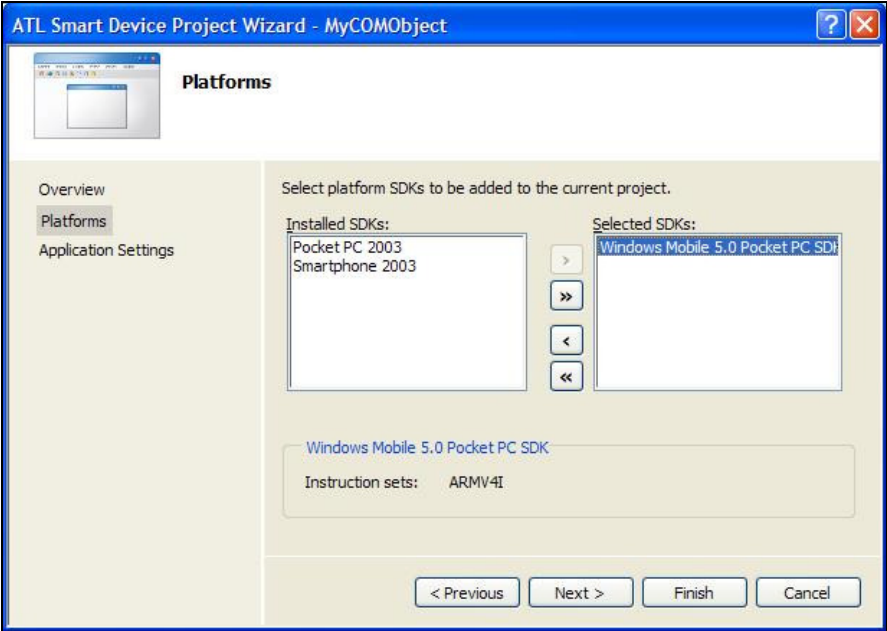
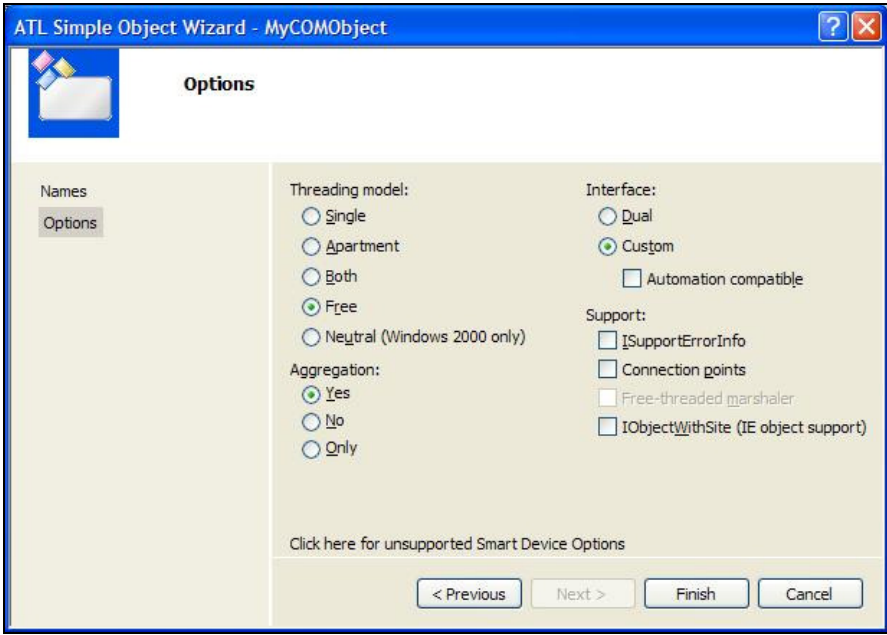
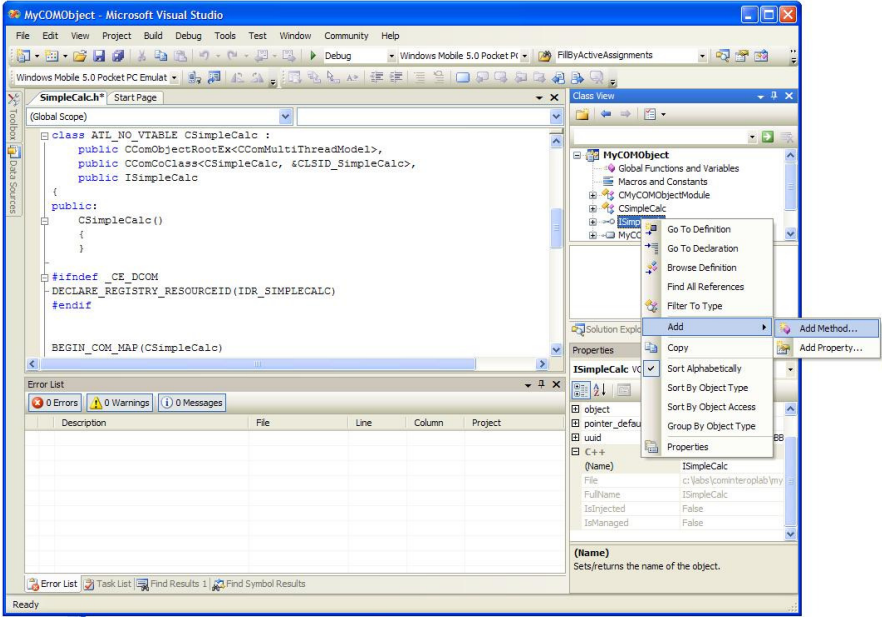


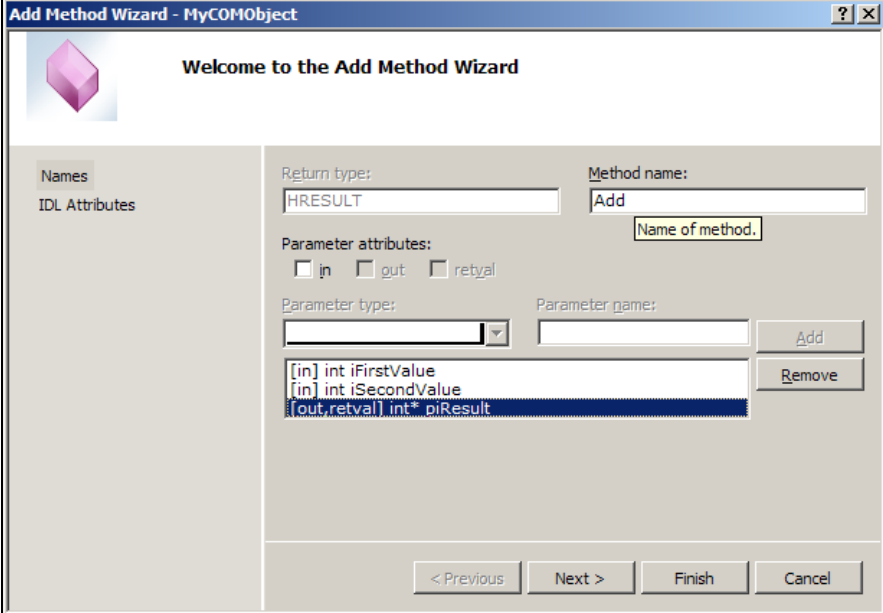
Figure 1. Add New Project dialog box

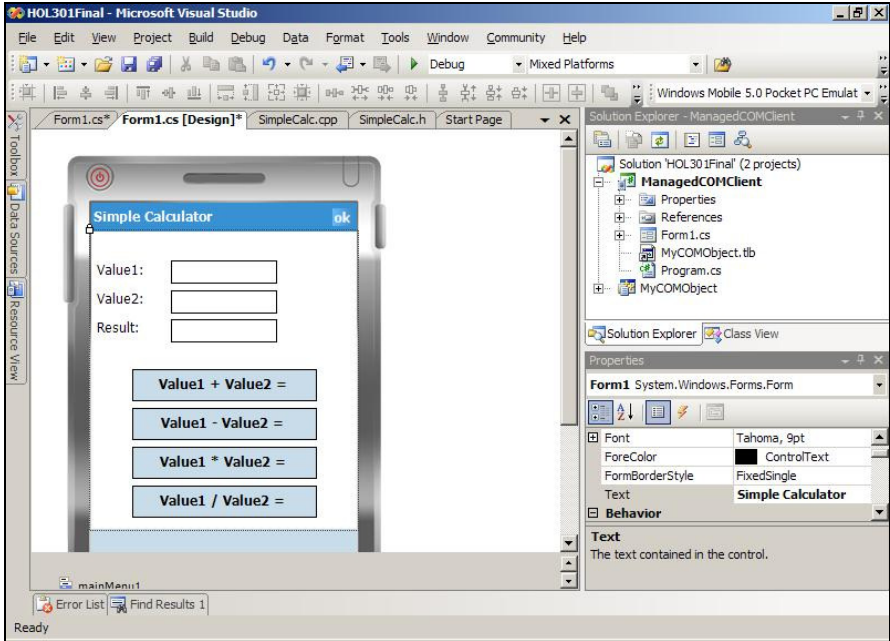
Tasks	Detailed Steps
	i. Click OK to start the ATL Smart Device Project Wizard. <i>Note: The wizard is going to assist you in creating your COM object. You need to provide some basic information, but you can leave most of the settings as they are.</i> j. On the welcome page of the wizard, click Next. k. On the Platforms page of the ATL Smart Device Project Wizard, as shown in Figure 2, be sure that only Windows Mobile 5.0 Pocket PC SDK is listed under Selected SDKs. Use the add and remove arrow buttons to move the appropriate SDKs.  Figure 2. Platforms page of the ATL Smart Device Project Wizard l. Click Next. m. On the Application Settings page of the ATL Smart Device Project Wizard, be sure that Dynamic-link library (DLL) is selected under Server type, and be sure that the additional options boxes are not selected. n. Click Finish to exit the wizard. <i>Note: The wizard has created a new project called MyCOMObject that contains all of the code to act as a COM component. To turn this project into a meaningful COM object, you have to add functionality for your specific COM object.</i>
2. To add a new class to the COM object	a. In Solution Explorer, right-click the MyCOMObject project, point to Add, and then click Class. <i>Note: (Alternatively, you can click Project Add Class from the Visual Studio 2005 menu.)</i> b. In the Add Class dialog box, in the Categories tree, select ATL, and be sure that ATL Simple Object is selected in the list of available templates, as shown in Figure 3.

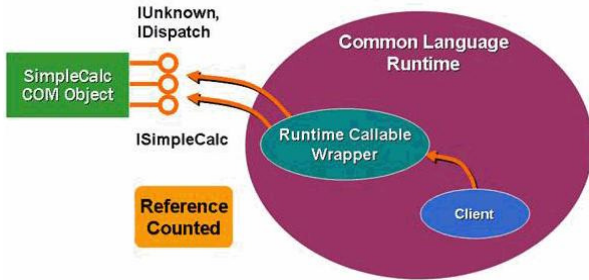
Tasks	Detailed Steps
	Figure 3. Add Class dialog box c. Click the Add button. <i>Note: The ATL Simple Object Wizard starts. In this wizard, you will give your new object a name and set some specific object settings.</i> d. On the left side of the Welcome to the ATL Simple Object Wizard page, select Names. e. In the C++ Short name box, type SimpleCalc. Leave all of the other values as default, as shown in Figure 4. Figure 4. Names page of the ATL Simple Object Wizard f. Click Next. g. On the Options page of the ATL Simple Object Wizard, under Threading model, select Free.

Tasks	Detailed Steps
	h. Under Interface, select Custom, as shown in Figure 5.  Figure 5. Options page of the ATL Simple Object Wizard i. Click Finish to close the wizard and create the new class. <i>Note: The ATL Simple Object Wizard has created a number of new files for you, in which you will add methods and interfaces that bring functionality to your COM object. The COM object that you are about to implement contains a few basic math functions, like add, subtract, multiply, and divide. The functionality is straightforward, but it should help you understand how a managed .NET Compact Framework application can interoperate with a COM object.</i> <i>Note: You will now work on the implementation of SimpleCalc. All methods that you are going to write will be exposed through COM interfaces. A COM client can access the methods and, thanks to COM Interop, a managed client application can access SimpleCalc's methods (as you will see in this HOL).</i>
3. To implement the functionality of SimpleCalc	a. In Visual Studio 2005, change to Class View by clicking View Class View or by using the keyboard shortcut CTRL+W+C. b. In the Class View pane, expand MyCOMObject, right-click the interface ISimpleCalc, and then select Add Add Method, as shown in Figure 6. <i>Note: The Add Method Wizard appears. In this wizard, you will specify the signature of the method that you are going to add.</i>

Tasks	Detailed Steps
	 Figure 6. Adding a method to ISimpleCalc - In the Add Method Wizard, type Add in the Method name box. - Type int in the Parameter type box. - Type iFirstValue in the Parameter name box. - Select the in check box under Parameter attributes. - Click the Add button. - Type int in the Parameter type box. - Type iSecondValue in the Parameter name box. - Select the in check box under Parameter attributes. - Click the Add button. - Type int* in the Parameter type box. - Type piResult in the Parameter name box. - Select the retval check box under Parameter attributes. - Click the Add button. Note: The Add Method Wizard's Names page should now look like Figure 7.

Tasks	Detailed Steps
	 Figure 7. Names page of the Add Method Wizard p. Click Finish. Note: You have just created a new empty method with a COM interface. You can find the method's interface in the header file <i>SimpleCalc.h</i>, the (still empty) implementation in the source file <i>SimpleCalc.cpp</i>. Next, you will add a few more methods to the component. You will also automatically see the new methods appear in the Class View pane. q. In the Class View pane, right-click the interface ISimpleCalc, and then click Add Add Method. r. Type Subtract in the Method name box. s. Repeat Steps 4 through 16 to add parameters to the method. You must enter exactly the same parameters. t. Repeating the same steps, add a new method called Multiply. u. Again, repeating the same steps, add a new method called Divide. Note: The only thing left to do to finish your COM component is add functionality. v. In Solution Explorer, (View Solution Explorer View), open the file SimpleCalc.cpp by double-clicking it. You can find this file under Source Files. w. Find the method CSimpleCalc::Add, and then add the following statement to the method (prior to the statement <code>return S_OK</code>). <pre>*piResult = iFirstValue + iSecondValue;</pre> x. Find the method CSimpleCalc::Subtract, and then add the following statement to the method (prior to the statement <code>return S_OK</code>). <pre>*piResult = iFirstValue - iSecondValue;</pre> y. Find the method CSimpleCalc::Multiply, and then add the following statement to the method (prior to the statement <code>return S_OK</code>).

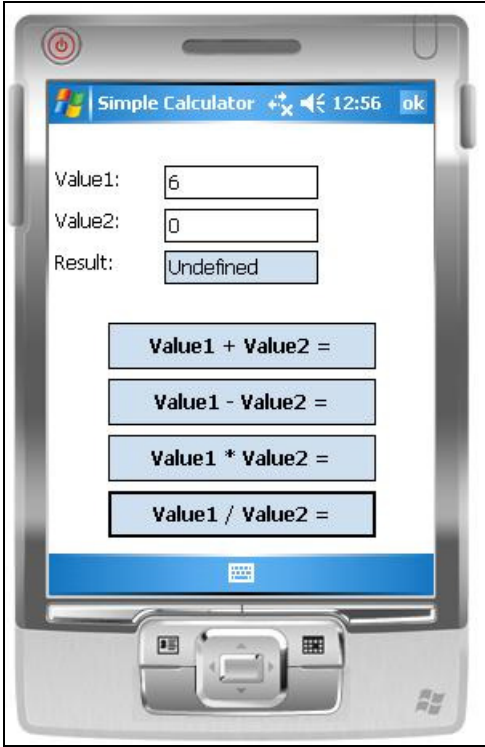
Tasks	Detailed Steps
	<pre>*piResult = iFirstValue * iSecondValue;</pre> z. Find the method CSimpleCalc::Divide, and then replace all of its code with the following code example. <pre>if (iSecondValue != 0) { *piResult = iFirstValue / iSecondValue; return S_OK; } else { *piResult = 0; return E_FAIL; }</pre> aa. Compile the COM object by clicking Build Build Solution or by pressing F6. <i>Note: Assuming you didn't get any compilation errors, you are now going to use the COM object inside a managed application.</i>
4. To use the COM object inside a managed application	a. In Solution Explorer, browse to the project ManagedCOMClient, and then expand it (if it is not expanded already). b. Open the file Form1.cs in the form designer by double-clicking it in Solution Explorer. <i>Note: To save time, the user interface is already created for you, which should look like Figure 8.</i>  Figure 8. User interface for ManagedCOMClient <i>Note: The next thing you will do is add functionality to the application to make use of MyCOMObject. Before you can use the COM component, you have to add a reference to it to make it available in the managed application. When you are adding a</i>

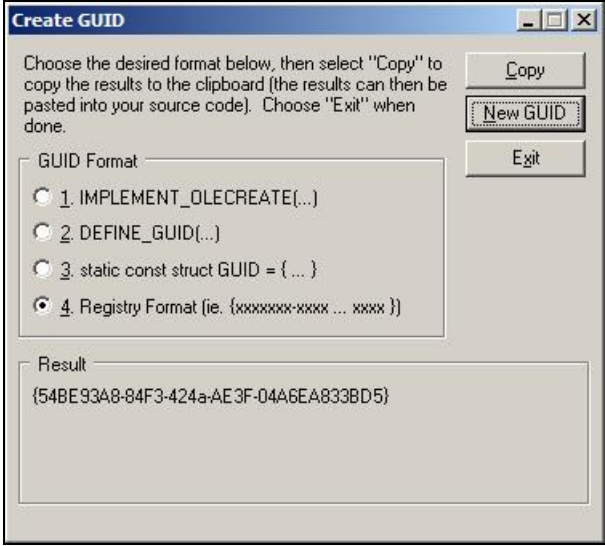
Tasks	Detailed Steps
	reference to a COM object, code is automatically generated by Visual Studio 2005 to be able to use your COM object inside a managed application. Note: You need a so-called Interop Assembly to be able to call into a COM object. The Interop Assembly contains managed definitions of COM interfaces and types. It can be generated manually by the Type Library Importer (tlbimp.exe), or it can be generated automatically by Visual Studio 2005 when you add a reference to the COM object's Type Library (*.tlb) file, which you will see in a moment. To a managed client application, the COM object looks like a managed type, as shown in Figure 9. The Runtime Callable Wrapper hides all COM details for you.  Figure 9. Calling a COM component from a managed client Note: To generate an Interop Assembly to be able to access the COM component from within managed code, you need to add a reference to <i>MyCOMObject.tlb</i>. - In Solution Explorer, right-click References, and then click Add Reference to add a reference to the ManagedCOMClient project. - In the Add Reference dialog box, click the Browse tab, and then browse to MyCOMObject.tlb in C:\Program Files\Windows Mobile Developer Samples\HOLs\HOL301_Incorp_COM_Obj_NETCF2_Apps\HOL301Exercise1\MyCOMObject\Windows Mobile 5.0 Pocket PC SDK (ARMV4I)\Debug. Note: You can easily browse to the type library file by clicking the Up One Level button on the toolbar of the Add Reference dialog box, after which you can browse through the MyCOMObject project tree to reach the MyCOMObject.tlb file. - Click OK to add the selected reference to the project. Note: You can now use the methods from <i>MyCOMObject</i> as if it were a managed object, as you will see when you are adding functionality to the button click event handlers of the <i>ManagedCOMClient</i> application. - In Solution Explorer, right-click Form1.cs, and then click View Code. - Create an alias for the "System.Runtime.InteropServices" namespace by adding the following statement immediately under the other <code>using</code> statements at the beginning of the Form1.cs file (this is the file into which all other code will be added as well). <pre>using System.Runtime.InteropServices;</pre> - Create an alias for your COM component by adding the following statement. <pre>using MyCOMObjectLib;</pre> - Add the following instance variable to the Form1 class.

Tasks	Detailed Steps
	<pre data-bbox="545 195 1398 226">private ISimpleCalc calculator = new SimpleCalc();</pre> Note: You now have declared an interface variable of type ISimpleCalc and instantiated it with a SimpleCalc object. This is, in fact, an instantiation of your COM component. j. Add the following code to the btnAdd_Click event handler. <pre data-bbox="545 459 1398 583">int firstValue = Convert.ToInt32(tbValue1.Text); int secondValue = Convert.ToInt32(tbValue2.Text); int result = calculator.Add(firstValue, secondValue); tbResult.Text = result.ToString();</pre> Note: As you saw when entering the Add method for the calculator, it looks like a managed object and even has full Microsoft IntelliSense® support. If you compare the managed call to the Add method as defined in the COM component, you will see a remarkable difference. The signature of the managed Add method looks like the following. <pre data-bbox="545 848 1398 879">int Add (int iFirstValue, int iSecondValue);</pre> Note: The signature of the COM Add method looks like the following. <pre data-bbox="545 1001 1398 1157">HRESULT Add ([in]int iFirstValue, [in]int iSecondValue, [out, retval]int* piResult);</pre> Note: The translation of the COM method to the managed method happened automatically when you added a reference to the Type Library file. k. Add the following code to the btnSubtract_Click event handler. <pre data-bbox="545 1352 1398 1476">int firstValue = Convert.ToInt32(tbValue1.Text); int secondValue = Convert.ToInt32(tbValue2.Text); int result = calculator.Subtract(firstValue, secondValue); tbResult.Text = result.ToString();</pre> l. Add the following code to the btnMultiply_Click event handler. <pre data-bbox="545 1608 1398 1732">int firstValue = Convert.ToInt32(tbValue1.Text); int secondValue = Convert.ToInt32(tbValue2.Text); int result = calculator.Multiply(firstValue, secondValue); tbResult.Text = result.ToString();</pre> m. Add the following code to the btnDivide_Click event handler. <pre data-bbox="545 1864 1398 1896">int firstValue = Convert.ToInt32(tbValue1.Text);</pre>

Tasks	Detailed Steps
	<pre data-bbox="548 195 1398 289">int secondValue = Convert.ToInt32(tbValue2.Text); int result = calculator.Divide(firstValue, secondValue); tbResult.Text = result.ToString();</pre> n. Compile the entire solution, including the code that you just entered by clicking Build Build Solution or by pressing F6. <i>Note: Assuming you didn't get compilation errors, you can now test the ManagedCOMClient application in the Windows Mobile 5.0 Pocket PC emulator.</i>
5. To test the ManagedCOMClient application in the emulator	a. In Visual Studio 2005, press F5 or select Debug Start Debugging to start the application in debugging mode. b. In the Deploy ManagedCOMClient dialog box, be sure that the Windows Mobile 5.0 Pocket PC Emulator is selected, and then click the Deploy button. <i>Note: If you receive an error during deployment that indicates that the process or file is in use, this means that the program is still running on the emulator and must be closed before a new copy can be deployed and run. This error may appear anytime in the lab that you deploy the emulator. See Appendix A in this HOL for instructions about exiting a running application.</i> c. Be sure that the emulator is visible by clicking its button in the Windows taskbar. <i>Note: After a while, you will see the application running inside the Windows Mobile 5.0 Pocket PC emulator. Now it is time to test the application. It may take some time to deploy the application for the first time. The first time you are using the emulator, prior to deploying the application, the .NET Compact Framework 2.0 will be deployed to the emulator as well. When the .NET Compact Framework 2.0 is already found on the emulator, this step is skipped.</i> d. Enter different numbers in the text boxes for Value1 and for Value2. e. Click all four buttons one after another, and examine the result. Every time a button is clicked, a method in the COM object is called. The running application should look like Figure 10.

Tasks	Detailed Steps
	Figure 10. ManagedCOMClient application running in the emulator f. Enter the number 0 in the Value2 box, and then click the divide button. You will see an exception being thrown, as shown in Figure 11. Figure 11. Exception that is thrown from the divide method call Note: This is the default behavior when a method call of a COM object returns a value other than S_OK as HRESULT. Therefore, it is always advisable to add exception handling when calling into COM objects.

Tasks	Detailed Steps
	g. Stop the debugger by clicking Debug Stop Debugging. h. Change the code for the btnDivide_Click event handler in source file Form1.cs to the following. <pre data-bbox="544 342 1398 722"> int firstValue = Convert.ToInt32(tbValue1.Text); int secondValue = Convert.ToInt32(tbValue2.Text); try { int result = calculator.Divide(firstValue, secondValue); tbResult.Text = result.ToString(); } catch { tbResult.Text = "Undefined"; } </pre> i. In Visual Studio, press F5 or click Debug Start Debugging to start the application in debugging mode. j. In the Deploy ManagedCOMClient dialog box, be sure that the Windows Mobile 5.0 Pocket PC Emulator is selected, and then click the Deploy button. k. Watch what happens if you enter the number 0 in the Value2 box and then click the divide button (at least the behavior is now much more acceptable, simply by adding an exception handler), as shown in Figure 12.  Figure 12. The application continues to run when attempting to divide by zero l. Close the application by clicking the ok button in the upper-right corner of the application.

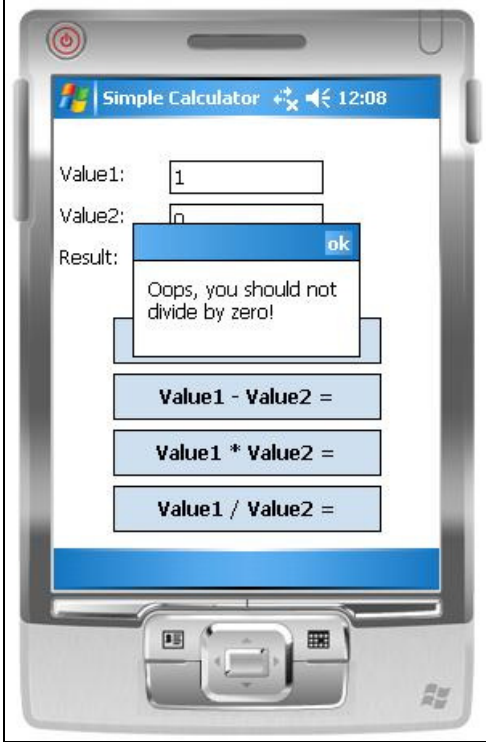
Tasks	Detailed Steps
	Note: In the .NET Compact Framework 2.0, it is not only possible to call methods on COM components without the need to write your own wrapper. It is also possible to call back from the COM object into the managed client, thanks to the fact that the .NET Compact Framework 2.0 also allows calling from native to managed code. You will now extend <i>MyCOMObject</i> by adding a new interface and implementing callback functionality. The callback functionality that you are implementing simply displays a message passed from the COM component to a managed message box. Note that even though callbacks are supported, activation of .NET types from COM is not supported.
6. To callback from COM into the managed client	a. In Solution Explorer, double-click the MyCOMObject.idl to open it. You can find the file in the tree view by clicking MyComObject Source Files. b. In the MyCOMObject.idl file, look for the definition of the line import atliface.idl, and then manually add a new interface immediately below it. Enter the following code for the new interface. <pre data-bbox="544 703 1398 1018">[object, uuid(), helpstring("ISimpleCalcCallback Interface"), pointer_default(unique)] interface ISimpleCalcCallback : IUnknown{ [, helpstring("method ShowMsg")] HRESULT ShowMsg([in] LPTSTR pMsg); };</pre> c. Add a new universally unique identifier (UUID) to the interface to distinguish it from other interfaces. (You can generate a UUID in Visual Studio 2005 by clicking Tools Create GUID.) d. In the Create GUID dialog box, be sure that the Registry Format option is selected, as shown in Figure 13.  Figure 13. Generating a new GUID to identify the newly created interface

Tasks	Detailed Steps
	e. Click New GUID, and then immediately click Copy. f. Click Exit to close the Create GUID dialog box. g. In MyCOMObject.idl, place the cursor between the brackets behind the UUID attribute in the code that you just entered, and then paste the created GUID in there. Be sure to remove the braces that were copied as part of the GUID. h. Verify that the interface you added in the MyCOMObject.idl file, including the GUID you pasted in it, looks like the following. <pre data-bbox="545 485 1398 804">[object, uuid(54BE93A8-84F3-424a-AE3F-04A6EA833BD5), helpstring("ISimpleCalcCallback Interface"), pointer_default(unique)] interface ISimpleCalcCallback : IUnknown{ [, helpstring("method ShowMsg")] HRESULT ShowMsg([in] LPTSTR pMsg); };</pre> Note: The value of the GUID that you pasted in differs from the value that you see in this example. You have just created a new interface for your COM component, and this interface contains only one method called ShowMsg. This is the only method that the ISimpleCalcCallback interface exposes. This method acts as the callback from managed code, so you will write the implementation for this particular interface later in managed code. First, however, you need to do something else. To store the callback function in the COM object, you have to pass a reference to it from managed code. Therefore, you will add another method to the ISimpleCalc interface. i. In the Class View pane, right-click the interface ISimpleCalc, and then select Add Add Method. Note: This step displays the Add Method Wizard, in which you will specify the signature of the method you are going to add. j. Type Initialize in the Method name box. k. Type ISimpleCalcCallback* as the Parameter type. l. Type callbackInterface in the Parameter name box. m. Select the ☒ in check box under Parameter attributes. n. Click the Add button, and then click the Finish button. Note: You have now created all interfaces, but to be able to use the callback function, you also have to implement the Initialize method. o. In Visual Studio 2005, in Solution Explorer, double-click SimpleCalc.cpp to open it. You can find this file under Source Files. p. Find the CSimpleCalc::Initialize method, and then add the following code to the method (prior to the statement <code>return S_OK</code>). <pre data-bbox="545 1703 1398 1892">pClientCallback = callbackInterface; // Because you are using a new interface for the first // time, be // sure to increment its reference count. pClientCallback->AddRef();</pre>

Tasks	Detailed Steps
	<pre data-bbox="545 226 1398 317">// and callback into managed code for the first time pClientCallback->ShowMsg(_T("Managed Callback installed!"));</pre> q. Find the CSimpleCalc::Divide method, and then add the following code to the method to give the user an error message. Add the code prior to the statement <code>return E_FAIL</code>. <pre data-bbox="545 506 1398 596">if (pClientCallback) pClientCallback->ShowMsg(_T("Oops, you should not divide by zero!"));</pre> Note: As you can see in the code for the Initialize method, the callback method is stored locally. However, you still need to declare a variable to store the callback method into. Finally, you are also providing some code to release the interface after you finish with the object. r. In Solution Explorer, double-click SimpleCalc.h to open it. You can find this file under Header Files. s. Add the following code toward the end of the CSimpleCalc class. <pre data-bbox="545 938 1398 997">private: ISimpleCalcCallback* pClientCallback;</pre> t. In the SimpleCalc.h header file, look for the FinalRelease method inside the CSimpleCalc class, and then add the following code to the method. <pre data-bbox="545 1165 1398 1224">if (pClientCallback) pClientCallback->Release();</pre> u. Compile the COM component that you just extended by clicking Build Build Solution, or by pressing F6. Note: Before you can use the callback functionality that you just have implemented, you need to provide the COM component with a managed function that can act as a callback. You will create the callback method as part of a new class that you will now add to the ManagedCOMClient project. v. In Solution Explorer, right-click the project ManagedCOMClient, and then select Add Class. w. In the Add New Item dialog box, be sure that Class is selected in the Templates list. x. Change the name of the class to NewMsgNotification.cs, and then click the Add button. y. Replace all of the code in the NewMsgNotification.cs source file that you just created with the following code. <pre data-bbox="545 1812 1398 1902">using System; using System.Collections.Generic; using System.Text;</pre>

Tasks	Detailed Steps
	<pre data-bbox="544 195 1372 636">using System.Windows.Forms; using System.Runtime.InteropServices; using MyCOMObjectLib; namespace ManagedCOMClient { public class NewMsgNotification : ISimpleCalcCallback { public void ShowMsg(string message) { MessageBox.Show(message); } } }</pre> Note: As you can see, the NewMsgNotification class derives from the ISimpleCalcCallback interface, which is the interface that you created in the COM component. The interface expects one function, called ShowMsg, that is defined in the class as well. Finally, you need to connect the callback function to the COM component. z. In Code view, open the Form1.cs source file in Visual Studio 2005. aa. Add an instance variable for the new class that you just created by adding the following code. <pre data-bbox="544 1003 1107 1066">private NewMsgNotification newMsg = new NewMsgNotification();</pre> bb. Find the constructor of the Form1 class, and then add a call to the Initialize method of the calculator COM object to tell the COM object what callback function to use (add the following statement immediately under <pre data-bbox="544 1264 1278 1327">InitializeComponent() : calculator.Initialize((ISimpleCalcCallback)newMsg);</pre> cc. In Visual Studio, press F5 or click Debug Start Debugging to start the application in debugging mode. dd. In the Deploy ManagedCOMClient dialog box, be sure that the Windows Mobile 5.0 Pocket PC Emulator is selected, and then click the Deploy button. Note: After a while, you will see the application running inside the Windows Mobile 5.0 Pocket PC emulator. Immediately, you will notice that a message appears and informs you that the managed callback is installed, as shown in Figure 14.

Tasks	Detailed Steps
	Figure 14. Managed callback installation dialog box <i>Note: This message box resides inside managed code (in the newly created class), but it is called from inside the COM component. Now, watch what happens if you divide by zero.</i> ee. Click ok to close the message. ff. Type 1 in the Value1 box, and then type 0 for Value2. gg. Click the divide button, and verify that another message appears, as shown in Figure 15.

Tasks	Detailed Steps
	 Figure 15. The divide by zero message hh. Click ok to close the message, and then close the application by clicking the ok button in the upper-right corner of the application. <i>Note: This step concludes the first exercise. You have created a COM component by using ATL and used that component inside a managed client application. Not only is it possible to call methods in the COM component from inside managed code, but it is also possible to register a managed callback function in the COM component. As you just have seen, it is very simple to access COM components from within managed code—without the need to write your own wrapper—as was the case in the .NET Compact Framework version 1.0. In the next exercise, you will use a popular, existing COM component inside a managed application.</i>

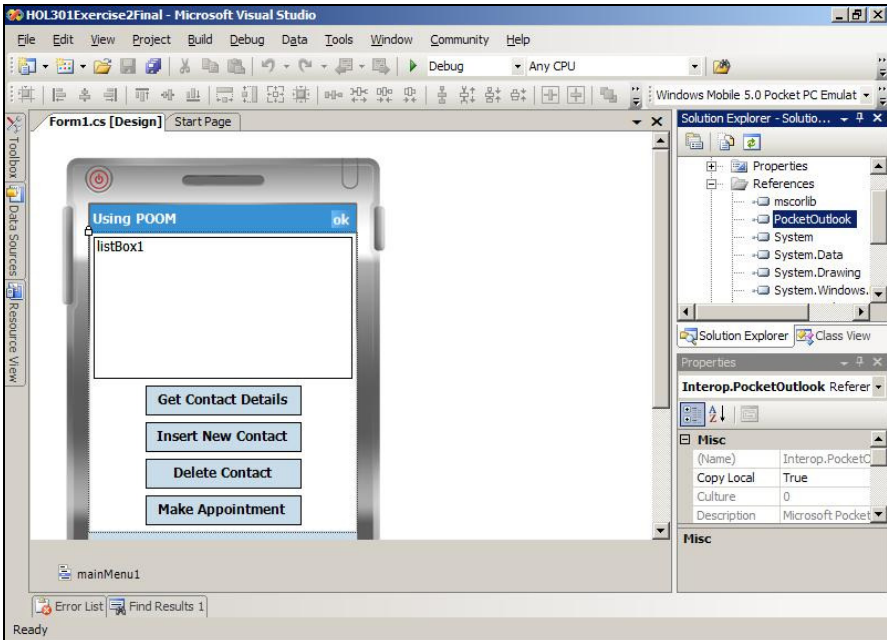
Exercise 2

Accessing Pocket Outlook Inside a Managed Application

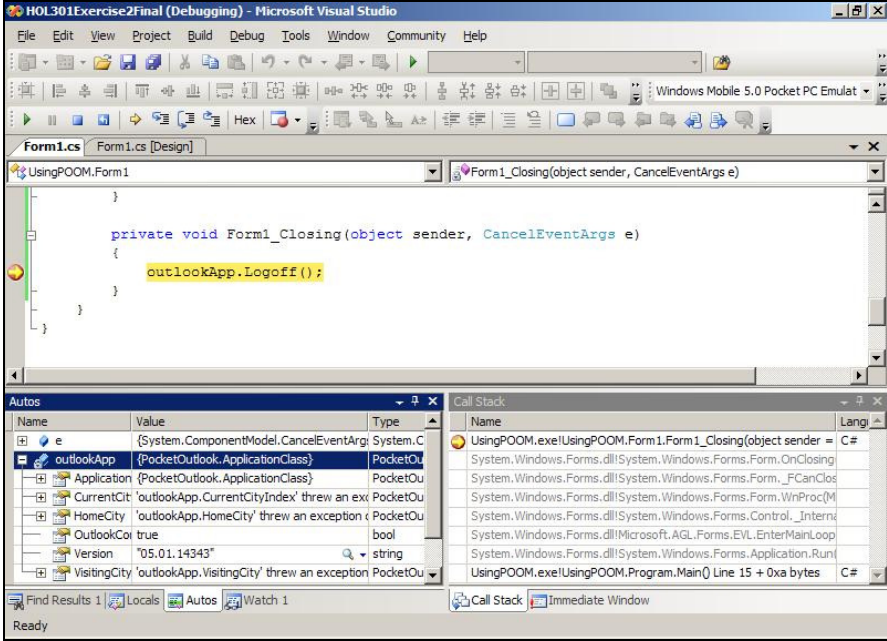
Scenario

In this exercise, you will create a new Windows Mobile 5.0–based Pocket PC application that makes use of Microsoft Office Outlook® Mobile by means of the Pocket Outlook Object Model (POOM). POOM is a COM component that exposes functionality of Pocket Outlook, so you can use that functionality in your own (managed) application. In the .NET Compact Framework 1.0, using POOM was not straightforward. You had to either create a wrapper DLL in native code that flattens out the POOM component or make use of a third-party library that did exactly this for you. With the .NET Compact Framework 2.0, it is much easier to use POOM inside a managed application, thanks to COM Interop. In this exercise, you are not going to create a fancy user interface; instead, you will concentrate on the possibilities that POOM offers.

Tasks	Detailed Steps
1. To open an existing solution and add functionality to it	a. If it is not already running, open Visual Studio 2005. b. In Visual Studio 2005, click File Open Project/Solution. c. In the Open Project dialog box, browse to C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL301\HOL301Exercise2. d. Select the HOL301Exercise2.sln file, and then click Open. The solution opens in Visual Studio 2005. <i>Note: The entire user interface is already created for you to save some time. In this exercise, you will add functionality to the application to access Pocket Outlook functionality.</i> <i>Note: To use the POOM in your managed application, you don't have to write numerous platform invoke wrappers or define your own managed object model. Instead, you can generate an assembly by using the Type Library Importer (tlbimp.exe) SDK tool, so you can call POOM directly. You will now create a POOM type library. From that, you will create an Interop Assembly, and you will reference that Interop Assembly in your Visual Studio 2005 project.</i> <i>The easiest way to call a COM component from managed code is to start with a type library that defines the component's interfaces, coclasses, and so on. Unfortunately, the type library for POOM is not included in the Pocket PC or Smartphone SDKs, so you'll have to build your own. Fortunately, building the type library is easy.</i>
2. To make POOM work with your Pocket PC application	a. On the desktop computer, click Start All Programs Microsoft Visual Studio 2005 Visual Studio Tools Visual Studio 2005 Command Prompt to open a Visual Studio 2005 command prompt. <i>Note: It is important to use this particular command prompt because it sets a number of environment variables, so you can use particular command-line tools that come with Visual Studio 2005.</i> b. In the Command Prompt window, type cd "C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL301\HOL301Exercise2" to change the working directory, and then press ENTER. <i>Note: Now you will create a type library for POOM by using midl.exe. You typically need to execute this step for all COM components that you want to access from inside a managed application and that don't have a type library available. You need to have the interface definition file for that particular COM component.</i> <i>Note: Pimstore.idl does not ship with the Windows Mobile 5.0 Pocket PC SDK. The file is included in the source code for this HOL in the folder specified in step 2.</i>

Tasks	Detailed Steps
	c. In the Command Prompt window, type midl pimstore.idl, and then press ENTER, which creates a type library for POOM. <i>Note: Executing this command creates a type library file for you with the name pimstore.tlb. You can simply ignore the warnings that the tool may generate. The next step is to reference pimstore.tlb from within Visual Studio.</i> d. Close the Command Prompt window. <i>Note: To generate an Interop Assembly to be able to access POOM from within managed code, you now need to add a reference to pimstore.tlb.</i> e. In Solution Explorer under the project UsingPOOM, right-click References, and then click Add Reference. f. In the Add Reference dialog box, click the Browse tab, and then browse to C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL301. g. Select pimstore.tlb. <i>Note: You can easily browse to the type library file by clicking the Up One Level button on the toolbar of the Add Reference dialog box.</i> h. Click OK to generate the Interop Assembly. i. If you expand the References tree in Solution Explorer, you will find a newly added reference to PocketOutlook, the just-created Interop Assembly that is your managed interface to POOM, as shown in Figure 16.  Figure 16. PocketOutlook reference to pimstore.tlb <i>Note: To be able to use POOM inside your managed application, you need to add a number of namespaces. You also need to create an instance to the Pocket Outlook application and log on to it. Because you are going to use POOM all through your application, you will create the instance to the application and log on to it when loading the main form. You will log off from the Pocket Outlook application when the main form is closed. You can use POOM functionality throughout the lifetime of the application.</i>
3. To access POOM inside your	a. In Solution Explorer , right-click Form1.cs , and then click View Code .

Tasks	Detailed Steps
Pocket PC application	b. Create an alias for the "System.Runtime.InteropServices" namespace by adding the following statement immediately under the other <code>using</code> statements at the beginning of the Form1.cs file (this is the file into which all other code will be added as well). <pre>using System.Runtime.InteropServices;</pre> c. Create an alias for POOM by adding the following statement. <pre>using PocketOutlook;</pre> d. Add the following instance variable to the Form1 class. <pre>private ApplicationClass outlookApp;</pre> e. In Solution Explorer, double-click Form1.cs to open Form1.cs [Design]. f. Click somewhere on the form outside the controls, and then in the Properties pane, click the Events button on the toolbar. g. Look for the Load event and double-click it. You now have created a Form1_Load event handler. h. In the Form1_Load event handler, add the following code. <pre>// Create an instance of the application object and log on outlookApp = new PocketOutlook.ApplicationClass(); outlookApp.Logon(0);</pre> i. Open Form1.cs [Design] again, and then click somewhere on the form outside the controls. In the Properties pane, click the Events button on the toolbar. j. Look for the Closing event and double-click it. You now have created a Form1_Closing event handler. k. In the Form1_Closing event handler, add the following code. <pre>outlookApp.Logoff();</pre> <i>Note: Now you should be able to use Pocket Outlook functionality in your own application, so this is a great time to test whether it is possible to create an instance of the Pocket Outlook application object and to log on to it and log off from it.</i> l. In Visual Studio 2005, press F5 or click Debug Start Debugging to start the application in debugging mode. m. In the Deploy Using Poom dialog box, be sure that the Windows Mobile 5.0 Pocket PC Emulator device is selected, and then click the Deploy button. <i>Note: After a while, you will see the application running inside the Windows Mobile 5.0 Pocket PC emulator.</i> n. While the application is running, find the Form1_Closing method in the Form1.cs file, and then set a breakpoint on the statement <code>outlookApp.Logoff()</code>. (Right-click the statement, point to Breakpoint, and then click Insert

Tasks	Detailed Steps
	Breakpoint.) o. Click ok in the upper-right corner of the application inside the emulator. <i>Note: The debugger stops the application on the breakpoint, and inside Visual Studio 2005, you can examine variables.</i> p. In Visual Studio 2005, expand the outlookApp object in the Autos pane. The contents of outlookApp should look similar to Figure 17.  Figure 17. Examining the contents of outlookApp in the Visual Studio 2005 debugger q. Right-click the red circular breakpoint symbol to the left of the line of code with the breakpoint, and then click Delete Breakpoint. r. Continue running the application by pressing F5 to allow the application to close properly. <i>Note: Before you can do anything useful with the application, you have to add a few contacts in Pocket Outlook on the emulator.</i> s. Be sure the Windows Mobile 5.0 Pocket PC emulator is visible, and then click Start Contacts in the emulator. t. In the emulator, click New, and then add a few new contacts. Enter only names for contacts, and then click ok, as shown in Figure 18.

Tasks	Detailed Steps
	Figure 18. Adding a new contact on the emulator Note: Now, you will write some code for the <i>UsingPOOM</i> application to show the contacts that you have just added in <i>Pocket Outlook</i> in the list box inside your application. You are going to fill the list box with contacts from <i>Pocket Outlook</i> in the Form1_Load event handler. u. Find the Form1_Load method inside the Form1.cs file, and then add the following code to it, immediately under the statement <code>outlookApp.Logon(0)</code>. <pre>ShowContacts();</pre> v. Add a new method called ShowContacts to the Form1.cs file with the following code. <pre>private void ShowContacts() { // Fill the list box with contact information from Pocket Outlook Folder contactsFolder = outlookApp.GetDefaultFolder(01DefaultFolders.olFolderConta cts); PocketOutlook.Items contacts = (PocketOutlook.Items)contactsFolder.Items; listBox1.BeginUpdate(); listBox1.Items.Clear(); foreach (ContactItem contact in contacts) { string name = contact.FirstName + " " +</pre>

Tasks	Detailed Steps
	<pre data-bbox="544 195 1019 346"> contact.LastName; listBox1.Items.Add(name); } listBox1.EndUpdate(); } </pre> Note: The code you've just entered retrieves the folder in which Pocket Outlook stores contacts and walks through the contact's <i>Items</i> collection to add contact names to the list box. If you have added some contacts in Pocket Outlook and you run the <i>UsingPOOM</i> application, you will see those contacts appear in the list box. w. In Visual Studio, press F5 or click Debug Start Debugging to start the application in debugging mode. x. In the Deploy UsingPoom dialog box, be sure that the Windows Mobile 5.0 Pocket PC Emulator is selected, and then click the Deploy button. y. After verifying that the contacts are visible in the list box, as shown in Figure 19, you can close the application again by clicking the ok button on the upper-right side of the form.  Figure 19. Viewing Pocket Outlook contact names in the UsingPOOM application
4. To extend the application's functionality	a. In Visual Studio's form designer, double-click each of the buttons on Form1 to add Click event handlers. Note: You have to switch back and forth between <i>Code view</i> and <i>form designer</i> to add click event handlers for all buttons. b. In the form designer, open the ContactDetails.cs file. Note: The <i>ContactDetails.cs</i> form already contains all user interface controls that you need, but the form does not yet contain any functionality. c. Add TextChanged event handlers for each of the text boxes. (To add

Tasks	Detailed Steps
	TextChanged event handlers, click the text box, find the TextChanged event in the Properties pane (you have to click the Events button to see all of the available events), and then double-click it.) <i>Note: You have to switch back and forth between Code view and the form designer to add click event handlers for all text boxes.</i> d. In ContactDetails.cs [Design], click somewhere on the form outside the controls, and then in the Properties pane, click the Events button on the toolbar. e. Look for the Load event and double-click it. f. In the form designer, add a Form_Closing event. Look for the Closing event and double-click it. g. In the ContactDetails.cs source file, select all of the text by pressing CTRL+A, and then replace all of the code with the following code. <pre> using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Text; using System.Windows.Forms; using System.Runtime.InteropServices; using PocketOutlook; namespace UsingPoom { public partial class ContactDetails : Form { private ContactItem thisContact; private bool isDirty; private bool newEntry; public ContactDetails(ContactItem contact, bool newEntryRequested) { InitializeComponent(); thisContact = contact; isDirty = false; newEntry = newEntryRequested; } private void ContactDetails_Load(object sender, EventArgs e) { if (thisContact != null && !newEntry) { tbName.Text = thisContact.FirstName + " " + thisContact.LastName; tbCompany.Text = thisContact.CompanyName; tbPhone.Text = thisContact.BusinessTelephoneNumber; tbEmail.Text = thisContact.Email1Address; } } } } </pre>

Tasks	Detailed Steps
	<pre> this.Text = "ContactDetails"; tbName.ReadOnly = true; } else if (newEntry) { this.Text = "Add new Contact"; tbName.ReadOnly = false; } } private void tbName_TextChanged(object sender, EventArgs e) { isDirty = true; } private void tbCompany_TextChanged(object sender, EventArgs e) { isDirty = true; } private void tbPhone_TextChanged(object sender, EventArgs e) { isDirty = true; } private void tbEmail_TextChanged(object sender, EventArgs e) { isDirty = true; } private void ContactDetails_Closing(object sender, CancelEventArgs e) { if (isDirty) { // The detailed information is changed. // Update it in the contact list. if (newEntry) { string[] nameParts; nameParts = tbName.Text.Split(' '); if (nameParts[0].Length == 0) { MessageBox.Show("Name field must be filled"); e.Cancel = true; } else { thisContact.FirstName = nameParts[0]; </pre>

Tasks	Detailed Steps
	<pre data-bbox="544 195 1388 1081"> if (nameParts.Length == 2) { thisContact.LastName = nameParts[1]; } else if (nameParts.Length > 2) { // Assuming that a middle name was // entered as well, // if more data was entered it will be // ignored! thisContact.MiddleName = nameParts[1]; thisContact.LastName = nameParts[2]; } } } thisContact.CompanyName = tbCompany.Text; thisContact.BusinessTelephoneNumber = tbPhone.Text; thisContact.EmailAddress = tbEmail.Text; thisContact.Save(); } } } } </pre> Note: The code you have just added shows some detailed information for a selected Pocket Outlook contact. Most of the code consists of overhead to get a well-functioning user interface. The contact is passed in the constructor of the form, and the information is displayed on the form during execution of the ContactDetails_Load event. Whenever the user is changing some text in one of the text boxes, the isDirty flag is set to true. When the form is closed again, the modified information in the text boxes is stored to the contact database of Pocket Outlook. Of course, this application is very simple. It automatically stores modified data without asking the user whether modifications need to be stored. Rather than creating a fancy user interface and providing a large amount of code for that, this exercise concentrates on using POOM. The ContactDetails dialog is also reused to enter new contact information. For that purpose, the constructor also contains a Boolean variable newEntry. The biggest difference between updating and inserting a new contact can be seen in the ContactDetails_Closing event handler because, in the case of a new contact, a valid name must be entered. In the case of updating details, the name cannot be changed. Take some time studying the code in the ContactDetails.cs file, and especially note how data is stored in the Pocket Outlook contact database. The next thing you need to do is make a modification to the Form1.cs file so it displays the ContactDetails dialog when the Get Contact Details button is clicked. Prior to displaying the ContactDetails dialog, the currently selected contact in the list box is retrieved from the contact database and passed to the constructor of the ContactDetails dialog. h. Locate the btnDetails_Click event handler in the Form1.cs file, and then add the following code to it.

Tasks	Detailed Steps
	<pre> Folder contactsFolder = outlookApp.GetDefaultFolder(01DefaultFolders.olFolderConta cts); PocketOutlook.Items contacts = (PocketOutlook.Items)contactsFolder.Items; if (listBox1.SelectedIndex == -1) listBox1.SelectedIndex = 0; ContactDetails cd = new ContactDetails((ContactItem) contacts.Item(listBox1.SelectedIndex+1), false); cd.ShowDialog(); </pre> Note: You have now finished adding functionality to display and modifying contact details. The next thing you will do is add code to the event handler for the Add new Contact button that, as you might remember, shares the ContactDetails dialog. i. Locate the btnNewContact_Click event handler in the Form1.cs file, and then add the following code to it. <pre> Folder contactsFolder = outlookApp.GetDefaultFolder(01DefaultFolders.olFolderConta cts); PocketOutlook.Items contacts = (PocketOutlook.Items)contactsFolder.Items; ContactItem contact = (ContactItem)contacts.Add(); ContactDetails cd = new ContactDetails(contact, true); cd.ShowDialog(); ShowContacts(); </pre> Note: The code for the btnNewContact_Click event handler and the code for the btnDetails_Click event handler are almost identical, with the exception of the way the constructor of the ContactDetails dialog is called and an additional update of the list box in case a new contact has been added. Next, you need to add some functionality to delete contact entries from the contact list. Again, the user interface is kept very simple. For instance, no confirmation is asked when you decide to delete a contact. j. Locate the btnDelContact_Click event handler in the Form1.cs file, and then add the following code to it. <pre> Folder contactsFolder = outlookApp.GetDefaultFolder(01DefaultFolders.olFolderConta cts); PocketOutlook.Items contacts = (PocketOutlook.Items)contactsFolder.Items; if (listBox1.SelectedIndex == -1) listBox1.SelectedIndex = 0; </pre>

Tasks	Detailed Steps
	<pre> ContactItem ci = (ContactItem)contacts.Item(listBox1.SelectedIndex + 1); ci.Delete(); ShowContacts(); </pre> <i>Note: Like updating and adding entries to the contact list, deleting entries from them is very straightforward when you are using POOM. It is simply a matter of selecting the correct entry in the PocketOutlook.Items list and calling its Delete method. The last thing to do before testing the application is add functionality to be able to enter appointments for a contact that you selected on the contact list.</i> k. In the form designer, open the file AddAppt.cs. <i>Note: The AddAppt.cs form already contains all of the user interface controls you will need, but the form does not yet contain any functionality.</i> l. In AddAppt.cs [Design], click somewhere on the form outside the controls, and then in the Properties pane, click the Events button on the toolbar. m. Look for the Load event, and then double-click it. n. In the form designer, add a Form_Closing event. Look for the Closing event and double-click it. o. In the AddAppt.cs source file, select all of the text by pressing CTRL+A, and then replace all of the code with the following code. <pre> using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Text; using System.Windows.Forms; using System.Runtime.InteropServices; using PocketOutlook; namespace UsingPOOM { public partial class AddAppt : Form { private ContactItem thisContact; private AppointmentItem thisAppt; public AddAppt(ContactItem contact, AppointmentItem appt) { InitializeComponent(); thisContact = contact; thisAppt = appt; } private void AddAppt_Load(object sender, EventArgs e) { tbName.Text = thisContact.FirstName + " " + </pre>

Tasks	Detailed Steps
	<pre> thisContact.LastName; dtpDate.Value = DateTime.Now; } private void AddAppt_Closing(object sender, CancelEventArgs e) { if (tbSubject.Text.Length == 0) { MessageBox.Show("You have to add a valid subject"); e.Cancel = true; } else { thisAppt.Subject = tbName.Text + " - " + tbSubject.Text; thisAppt.Start = dtpDate.Value; thisAppt.Save(); } } } </pre> Note: The code you have just added takes a name from the contact list and adds a new appointment for that particular contact. Adding contacts or adding appointments is more or less the same action. This time, to add a new appointment, you are collecting the appointment details and storing the appointment in the Pocket Outlook database by using the Add method. Just like the contacts, you have to be sure that you are using the correct folder where all appointments are stored. That is something you will set up in the event handler of the New Appointment button. The last modification that you need to make to the Form1.cs file is to display the AddAppts dialog when the New Appointment button is clicked. Prior to displaying the AddAppts dialog, the currently selected contact in the list box is retrieved from the contact database and passed to the constructor of the AddAppts dialog. A new, empty appointment is also passed to the dialog. p. Locate the btnNewAppt_Click event handler in the Form1.cs file, and then add the following code to it. <pre> Folder contactsFolder = outlookApp.GetDefaultFolder(OlDefaultFolders.olFolderConta cts); PocketOutlook.Items contacts = (PocketOutlook.Items)contactsFolder.Items; Folder apptsFolder = outlookApp.GetDefaultFolder(OlDefaultFolders.olFolderCalen dar); PocketOutlook.Items appts = (PocketOutlook.Items)apptsFolder.Items; AppointmentItem appt = (AppointmentItem)appts.Add(); </pre>

Tasks	Detailed Steps
	<pre data-bbox="544 195 1398 478"> if (listBox1.SelectedIndex == -1) listBox1.SelectedIndex = 0; AddAppt aa = new AddAppt((ContactItem)contacts.Item(listBox1.SelectedIndex + 1), appt); aa.ShowDialog(); </pre> Note: The only action that remains is for you to test the application. If you want to verify that the information entered in this application is also available in Pocket Outlook, you can simply change to either Contacts or Calendar and compare the information shown in those applications to the information that you entered in the UsingPOOM application. q. In Visual Studio, press F5 or select Debug Start Debugging to start the application in debugging mode. r. Verify the correct operation of the application by testing all options, and regularly compare the results with Contacts and Calendar. You should feel free to enter, modify, and delete contacts as well as enter new appointments, as shown in Figure 20.  Figure 20. Entering a new appointment s. When you are finished, close the application by clicking the ok button on the upper-right corner of the form.

Exercise 3

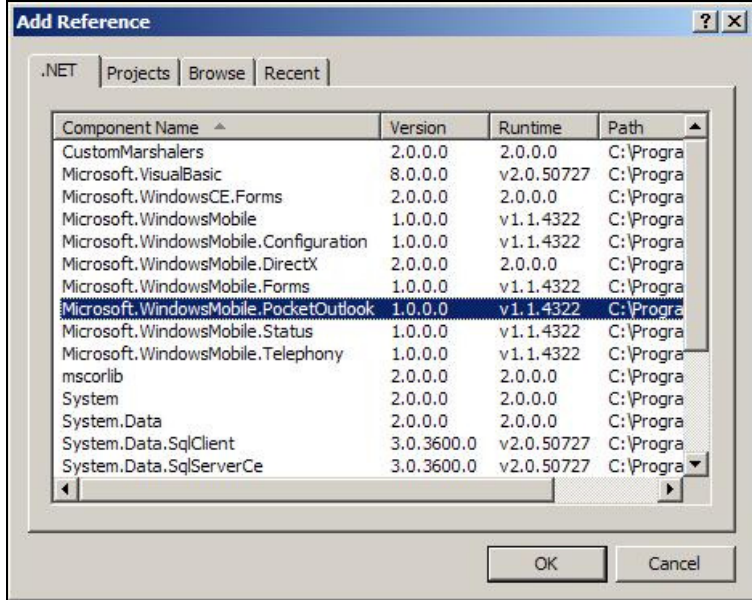
Accessing Pocket Outlook by Using Windows Mobile 5.0 Managed APIs

Scenario

The .NET Compact Framework 2.0 contains great functionality to make interoperability with native code easier and more complete. In the previous exercises, you have seen that it is relatively easy to use COM components from inside a managed application. Using Windows Mobile 5.0 devices, it will even be easier to access some native components by means of managed APIs that will ship as part of the SDKs for those devices. Particularly, using the Telephony API and Pocket Outlook API will be very easy.

In this exercise, you will create a managed application for a Windows Mobile 5.0 Pocket PC that makes use of Pocket Outlook, just as you did in the previous exercise. There is one important difference, however. This time, you will make use of the Windows Mobile 5.0 Managed APIs, so there is no need to interoperate with COM components yourself. To compare the amount of code that you have to provide yourself, you will create an application with the same functionality as that in Exercise 2.

Tasks	Detailed Steps
1. To open an existing solution and add functionality to it	a. If it is not already running, open Visual Studio 2005. b. In Visual Studio 2005, click File Open Project/Solution. c. In the Open Project dialog box, browse to C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL301\HOL301Exercise3. d. Select the HOL301Exercise3.sln file, and then click Open. The solution opens in Visual Studio 2005. <i>Note: The entire user interface is already created for you to save some time. In this exercise, you will add functionality to the application to access Pocket Outlook functionality, making use of the Windows Mobile 5.0 managed APIs.</i> e. In the form designer, double-click each button on Form1 to add Click event handlers. <i>Note: You have to switch back and forth between the Code view and the form designer to add click event handlers for all of the buttons.</i> <i>Note: In Windows Mobile 5.0-based devices, a lot of the device functionality is exposed by managed APIs. You can simply add a reference to the managed API that you need and start using it. All components starting with Microsoft.WindowsMobile contain managed wrappers around particular functionality.</i>
2. To make Pocket Outlook work with your application	a. In Solution Explorer under project ManagedPOOM, right-click References, and then click Add Reference. b. In the Add Reference dialog box, click the .NET tab, select Microsoft.WindowsMobile.PocketOutlook, and then click OK to add the reference to your project, as shown in Figure 21.

Tasks	Detailed Steps
	 Figure 21. Add Reference dialog box Note: To be able to use Pocket Outlook inside your managed application, you need to add a namespace. You also need to create an instance to the Pocket Outlook application and log on to it by means of instantiating an OutlookSession object. Because you are going to use Pocket Outlook all through your application, OutlookSession will be available in an instance variable. c. Create an alias for the "Microsoft.WindowsMobile.PocketOutlook" namespace by adding the following statement immediately under the other <code>using</code> statements at the beginning of the Form1.cs file (this is the file into which all other code will be added as well). <pre>using Microsoft.WindowsMobile.PocketOutlook;</pre> Note: Because the "Microsoft.WindowsMobile.PocketOutlook" namespace contains true managed interfaces to Pocket Outlook, there is no need to set up COM Interop as you had to do in Exercise 2. d. Add the following instance variable to the Form1 class. <pre>private OutlookSession oSession;</pre> e. Open Form1.cs [Design] in Visual Studio 2005, and then click somewhere on the form outside the controls. In the Properties pane, click the Events button on the toolbar. f. Look for the Load event, and then double-click it. You now have created a Form1_Load event handler. g. In the Form1_Load event handler, add the following code. <pre>oSession = new OutlookSession(); ShowContacts();</pre>

Tasks	Detailed Steps
	h. In Form1.cs [Design] again, click somewhere on the form outside the controls. In the Properties pane, click the Events button on the toolbar. i. Look for the Closing event, and then double-click it. You now have created a Form1_Closing event handler. j. In the Form1_Closing event handler, add the following code. <pre>oSession.Dispose();</pre> k. Add a new method called ShowContacts to the Form1.cs file with the following code. <pre>private void ShowContacts() { // Fill the list box with contact information from Pocket Outlook ContactFolder cFolder = oSession.Contacts; listBox1.BeginUpdate(); listBox1.Items.Clear(); foreach (Contact contact in cFolder.Items) { string name = contact.FirstName + " " + contact.LastName; listBox1.Items.Add(name); } listBox1.EndUpdate(); }</pre> <i>Note: If you compare the code for ShowContacts with ShowContacts in the previous exercise, you will see that it is extremely simple to access the contact database with Windows Mobile 5.0–based devices in combination with managed code. This is a great time to have a first test run of the application.</i> l. On the Device toolbar, verify that the Windows Mobile 5.0 Pocket PC Emulator is selected as the target device. m. On the Device toolbar, click the Connect to Device button to set up a connection to the emulator. <i>Note: You will see the Windows Mobile 5.0 Pocket PC emulator starting when it has not been started already. When a connection with Visual Studio 2005 is established, a confirmation message appears in the connecting dialog; click Close to dismiss this dialog. To test your application, you first have to add one or two contacts in the contact database. If there are already contacts in the contact database, you can omit the following two steps.</i> n. Be sure the Windows Mobile 5.0 Pocket PC emulator is visible. o. On the emulator, click Start Contacts. p. Add a few new contacts (entering only names for contacts is sufficient). <i>Note: Now you are ready to deploy and run your application on the Windows Mobile</i>

Tasks	Detailed Steps
	<i>5.0 Pocket PC emulator.</i> q. In Visual Studio 2005, press F5 or select Debug Start Debugging to start the application in debugging mode. r. In the Deploy ManagedPoom dialog box, be sure that the Windows Mobile 5.0 Pocket PC Emulator is selected, and then click the Deploy button. <i>Note: After a while, you will see your application running inside the emulator. Immediately when the main form of the application is shown, you will see the contacts that you just entered in the contact database appear in the list box of your application. Those contacts are, of course, retrieved from the Outlook contact database.</i> s. After verifying that the contacts are visible in the list box, you can close the application again by clicking the ok button on the upper-right side of the form.
3. To extend the application's functionality	a. In the form designer, open the ContactDetails.cs file. <i>Note: The ContactDetails.cs form already contains all of the user interface controls that you need, but the form does not yet contain any functionality.</i> b. Add TextChanged event handlers for each of the text boxes. To add TextChanged event handlers, click the text box, find the TextChanged event in the Properties pane (you have to click the Events button to see all available events), and then double-click it. c. In ContactDetails.cs [Design], click somewhere on the form outside the controls. In the Properties pane, click the Events button on the toolbar. d. Look for the Load event, and then double-click it. e. In the form designer, add a Form_Closing event. Look for the Closing event, and then double-click it. f. In the ContactDetails.cs source file, select all of the text by typing CTRL+A, and then replace all of the code with the following code. <pre> using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Text; using System.Windows.Forms; using Microsoft.WindowsMobile.PocketOutlook; namespace ManagedPoom { public partial class ContactDetails : Form { private Contact thisContact; private bool isDirty; private bool newEntry; public ContactDetails(Contact contact, bool newEntryRequested) { InitializeComponent(); thisContact = contact; isDirty = false; newEntry = newEntryRequested; } } </pre>

Tasks	Detailed Steps
	<pre> private void ContactDetails_Load(object sender, EventArgs e) { if (thisContact != null && !newEntry) { tbName.Text = thisContact.FirstName + " " + thisContact.LastName; tbCompany.Text = thisContact.CompanyName; tbPhone.Text = thisContact.BusinessTelephoneNumber; tbEmail.Text = thisContact.EmailAddress; this.Text = "ContactDetails"; tbName.ReadOnly = true; } else if (newEntry) { this.Text = "Add new Contact"; tbName.ReadOnly = false; } } private void tbName_TextChanged(object sender, EventArgs e) { isDirty = true; } private void tbCompany_TextChanged(object sender, EventArgs e) { isDirty = true; } private void tbPhone_TextChanged(object sender, EventArgs e) { isDirty = true; } private void tbEmail_TextChanged(object sender, EventArgs e) { isDirty = true; } private void ContactDetails_Closing(object sender, CancelEventArgs e) { if (isDirty) { // The detailed information is changed. // Update it in the contact list. } } </pre>

Tasks	Detailed Steps
	<pre> if (newEntry) { string[] nameParts; nameParts = tbName.Text.Split(' '); if (nameParts[0].Length == 0) { MessageBox.Show("Name field must be filled"); e.Cancel = true; } else { thisContact.FirstName = nameParts[0]; if (nameParts.Length == 2) { thisContact.LastName = nameParts[1]; } else if (nameParts.Length > 2) { // Assuming that a middle name was // entered as well, // if more data was entered it will be // ignored! thisContact.MiddleName = nameParts[1]; thisContact.LastName = nameParts[2]; } } } thisContact.CompanyName = tbCompany.Text; thisContact.BusinessTelephoneNumber = tbPhone.Text; thisContact.EmailAddress = tbEmail.Text; thisContact.Update(); } } </pre> Note: Because this code mainly deals with controlling the user interface, the majority of it is identical to the code that you had to add for the ContactDetails form in Exercise 2. The biggest difference is that you now use a Contact object to refer to contact information, instead of a ContactItem object. Another difference is that you use an Update method to store changes to a contact instead of a Save method. For an explanation of the code, you can reread the description of the ContactDetails form in Exercise 2. The next thing you need to do is modify the Form1.cs file to be able to display the ContactDetails dialog when the Get Contact Details button is clicked. Prior to displaying the ContactDetails dialog, the currently selected contact in the list box is

Tasks	Detailed Steps
	<i>retrieved from the contact database and passed to the constructor of the ContactDetails dialog.</i> g. Locate the btnDetails_Click event handler in the Form1.cs file, and then add the following code to it. <pre> ContactFolder cFolder = oSession.Contacts; if (listBox1.SelectedIndex == -1) listBox1.SelectedIndex = 0; ContactDetails cd = new ContactDetails(cFolder.Items[listBox1.SelectedIndex], false); cd.ShowDialog(); </pre> <i>Note: Comparing the code of the btnDetails_Click event handler with the same event handler in Exercise 2 makes it clear that using the Windows Mobile 5.0 managed APIs is very straightforward. You have now finished with adding functionality to display and modify contact details. Next, you will add code to the event handler for the Add new Contact button that, as you might remember from the previous exercise, shares the ContactDetails dialog.</i> h. Locate the btnNewContact_Click event handler in the Form1.cs file, and then add the following code to it. <pre> ContactFolder cFolder = oSession.Contacts; ContactDetails cd = new ContactDetails(cFolder.Items.AddNew(), true); cd.ShowDialog(); ShowContacts(); </pre> <i>Note: As you can see, the code for the btnNewContact_Click event handler and the code for the btnDetails_Click event handler are almost identical, with the exception of the way the constructor of the ContactDetails dialog is called and an additional update of the list box in case a new contact has been added.</i> <i>Next, you will add some functionality to delete contact entries from the contact list. Again, the user interface is kept very simple; for instance, no confirmation is asked when you decide to delete a contact.</i> i. Locate the btnDelContact_Click event handler in the Form1.cs file, and then add the following code to it. <pre> ContactFolder cFolder = oSession.Contacts; if (listBox1.SelectedIndex == -1) listBox1.SelectedIndex = 0; cFolder.Items[listBox1.SelectedIndex].Delete(); ShowContacts(); </pre> <i>Note: Like updating and adding entries to the contact list, deleting entries from them is very straightforward. It is simply a matter of selecting the correct entry in the ContactFolder Items collection list and calling its Delete method.</i> <i>The last thing to do before testing the application is add functionality to be able to enter appointments for a contact that you selected on the contact list.</i> j. In the form designer, open the AddAppt.cs file.

Tasks	Detailed Steps
	<i>Note: The <code>AddAppt.cs</code> form already contains all of the user interface controls that you need, but the form does not yet contain any functionality.</i> k. In <code>AddAppt.cs [Design]</code>, click somewhere on the form outside the controls. In the Properties pane, click the Events button on the toolbar. l. Look for the Load event, and then double-click it. m. In the form designer, add a Form_Closing event. Look for the Closing event, and then double-click it. n. In the <code>AddAppt.cs</code> source file, select all of the text by pressing CTRL+A, and then replace all of the code with the following code. <pre> using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Text; using System.Windows.Forms; using Microsoft.WindowsMobile.PocketOutlook; namespace ManagedPOOM { public partial class AddAppt : Form { private Contact thisContact; private Appointment thisAppt; public AddAppt(Contact contact, Appointment appt) { InitializeComponent(); thisContact = contact; thisAppt = appt; } private void AddAppt_Load(object sender, EventArgs e) { tbName.Text = thisContact.FirstName + " " + thisContact.LastName; dtpDate.Value = DateTime.Now; } private void AddAppt_Closing(object sender, CancelEventArgs e) { if (tbSubject.Text.Length == 0) { MessageBox.Show("You have to add a valid subject"); e.Cancel = true; } else { </pre>

Tasks	Detailed Steps
	<pre data-bbox="544 195 1398 447"> thisAppt.Subject = tbName.Text + " - " + tbSubject.Text; thisAppt.Start = dtpDate.Value; thisAppt.Update(); } } } } </pre> Note: As with the ContactDetails dialog, the majority of the code that you have just added deals with the user interface. However, if you compare the code of the ContactDetails.cs source file with the code of AddAppt.cs source file, you can see that adding an appointment or adding a new contact are very similar actions. To be able to use the functionality that you just added, you must now implement the event handler of the New Appointment button. You need to make one last modification to the Form1.cs file to be able to display the AddAppts dialog when the New Appointment button is clicked. Prior to displaying the AddAppts dialog, the currently selected contact in the list box is retrieved from the contact database and passed to the constructor of the AddAppts dialog. A new, empty appointment is also passed to the dialog. o. Locate the btnNewAppt_Click event handler in the Form1.cs file, and then add the following code to it. <pre data-bbox="544 953 1398 1205"> ContactFolder cFolder = oSession.Contacts; AppointmentFolder aFolder = oSession.Appointments; if (listBox1.SelectedIndex == -1) listBox1.SelectedIndex = 0; AddAppt aa = new AddAppt(cFolder.Items[listBox1.SelectedIndex], aFolder.Items.AddNew()); aa.ShowDialog(); </pre> Note: The only action that remains is for you to test the application. If you want to verify that the information entered in this application is also available in Pocket Outlook, you can simply change to either Contacts or Calendar and then compare the information shown in those applications to the information that you entered in the ManagedPOOM application. p. In Visual Studio 2005, press F5 or select Debug Start Debugging to start the application in debugging mode. q. Verify the correct operation of the application by testing all of the options, and regularly compare the results with Contacts and Calendar. You should feel free to enter, modify, and delete contacts as well as enter new appointments. r. When you are finished, close the application by clicking the ok button on the upper-right corner of the form.

Summary

In this lab, you performed the following exercises:

- Creating and using a COM object by using Visual Studio 2005
- Accessing Pocket Outlook inside a managed application
- Accessing Pocket Outlook by using Windows Mobile 5.0 managed APIs

In this lab, you created a number of applications to explore the COM interoperability features of the .NET Compact Framework 2.0. You learned how to use a COM component that you wrote from scratch by using ATL inside a managed application. You also learned how to use existing COM components inside a managed application, by using the Pocket Outlook Object Model. Finally, you experienced that even with great COM support in the .NET Compact Framework version 2.0, using existing applications with a COM interface on a next-generation Windows Mobile-based device is even easier by making use of managed APIs that expose functionality of those applications and components.