

Microsoft® Virtual Labs

**Windows Mobile 5.0:
Developing Managed Applications
for Pocket PC and Smartphone
Devices**

Microsoft®

Table of Contents

Windows Mobile 5.0: Developing Managed Applications for Pocket PC and Smartphone Devices	1
Introduction.....	1
What is the Windows Mobile 5.0 Platform?	2
Exercise 1: Developing a Windows Mobile 5.0 Pocket PC Application.....	3
Exercise 2: Developing a Windows Mobile 5.0 Smartphone Application.....	16
Summary	26

Windows Mobile 5.0: Developing Managed Applications for Pocket PC and Smartphone Devices

Objectives

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

- Describe the Windows Mobile 5.0 Platform
- Create managed applications for Windows Mobile 5.0 Devices
- Use the new Windows Mobile 5.0 APIs
- Understand how to use the Device Emulators
- Share common code between devices

Prerequisites

Experience with Visual Studio .NET and .NET development

Estimated Time to Complete This Lab

60 Minutes

Login Information

The password for the Administrator account on this computer is: password.

Introduction

This tour is aimed primarily at readers with little or no experience with mobile development and will show you how easy it is to leverage your existing .NET development skills to create first class mobile solutions. After you finish this guided tour you will have a conceptual understanding of managed development for current mobile devices, including Pocket PC and Smartphone.

Windows Mobile-based devices, both Pocket PC and Smartphone, are widely deployed around the world. Much of what drives the popularity of these highly portable devices is their rapidly improving hardware capabilities. These devices now provide high-quality displays, cameras, increased memory sizes, and powerful communications capabilities that were unimaginable not long ago. Windows Mobile 5.0 puts these powerful device capabilities and much more into easy reach of developers.

What is the Windows Mobile 5.0 Platform?

The Windows Mobile 5.0 platform is the successor to Windows Mobile 2003 Second Edition platform and refers to the class of devices running Windows CE 5.0 and optionally the .NET Compact Framework 2.0. This comprehensive new platform enables Visual Studio 2005 developers to target both Pocket PC and Smartphone devices that run variants of Windows CE operating system. Out of the box, Visual Studio 2005 enables and assists the development of applications for devices running Windows CE version 4.2, but by installing two freely available SDKs, support for Windows Mobile 5.0 and Windows CE 5.0 is now available.



Figure 1 - The Windows Mobile 5.0 Platform

The WM 5.0 SDKs provide the following for Visual Studio 2005 users:

- Pocket PC and Smartphone project templates
- Native headers and libraries
- Managed assemblies
- Emulators
- Documentation and samples

Exercise 1: Developing a Windows Mobile 5.0 Pocket PC Application

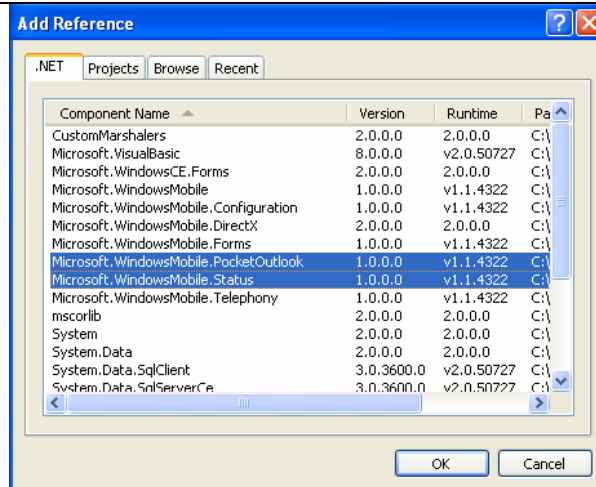
Scenario

This exercise demonstrates how to enable a mobile application to capture pictures, send and receive SMS Messages, exploit the new state model and its event-driven programming model, and work with Outlook Mobile to send email and generate appointments.

Part 1: Working with Mobile Forms

Tasks	Detailed Steps
1. Start Visual Studio 2005.	a. Click Start All Programs Microsoft Visual Studio 2005 Microsoft Visual Studio 2005 .
2. Load Exercise1.sln.	a. Click File Open Project/Solution and navigate to the Managed Guided Tour\Exercise 1 folder on the desktop. b. Select Exercise 1.sln and click Open .
3. Open the SendPicture form in the designer.	a. In the Solution Explorer , right-click on SendPicture.cs and select Open .  The screenshot shows the Visual Studio 2005 environment. On the left, the 'SendPicture.cs (Design)' window displays a mobile form with a red background. The form has a title bar 'Send Picture' and a 'File' menu. It contains a 'Name:' label, a 'Capture' button, a 'Choose' button, a 'Send Photo' button, and a 'Credited' checkbox. On the right, the 'Solution Explorer' shows the project structure with 'SendPicture.cs' selected. Below it, the 'Properties' window shows the 'photoPictureBox' control with properties like 'Tag, Bottom, Left, Right' set to '205, 5, 5'. <i>Most developers are surprised and delighted by the fact that developing for mobile devices using Visual Studio is very similar to any other kind of .NET project. This is not how it's always been. Until recently, device development has been something of a specialist skill. But now you can write code using the same languages (C#, VB.NET, C++), same code editors and debuggers, same on-line help system, and the same development environment that you use for Windows or Web development.</i>
4. Add references to the following components listed below:	a. In the Solution Explorer , right-click on the References node and select Add Reference to load the Add Reference dialog box.

Microsoft.WindowsMobile.Status and
Microsoft.WindowsMobile.PocketOutlook.



As you can see in the Add Reference dialog, there are a number of assemblies available that contain useful classes within the Microsoft.WindowsMobile namespace that provide developers with rich functionality for mobile development, including Configuration, DirectX, Forms, PocketOutlook, Status, and Telephony.

The PocketOutlook namespace does precisely what its name suggests by exposing Pocket Outlook programmability to managed developers. This allows you to connect to Pocket Outlook on a device and work with appointments, contacts, and tasks. We will make use of PocketOutlook later in this section.

The Status namespace provides a unified mechanism for storing device, application, and system state information. We will take a look at this in a later section when we show how to intercept SMS messages.

The Configuration namespace helps simplify the configuration and deployment of device applications by using XML. Using a static method called ProcessConfiguration, you can configure email, security policies, localization, the file system, and much more.

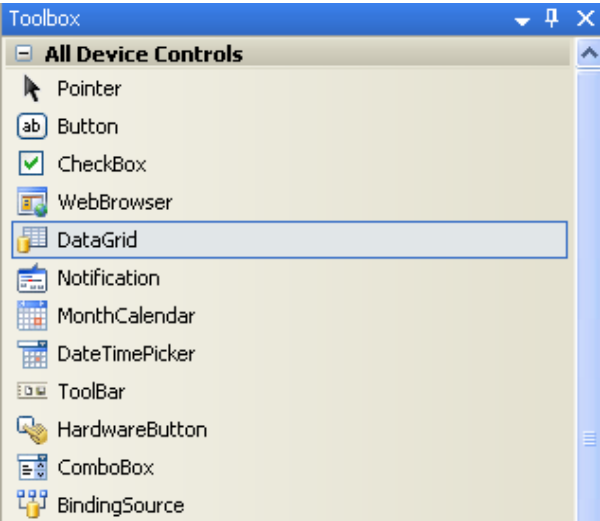
The DirectX namespace provides access to Direct3D, DirectDraw, and DirectShow, which provide functionality to help you produce rich two-dimensional and three-dimensional graphical environments as well as digital video.

The Telephony namespace exposes basic telephone behavior that allows applications to initiate phone calls.

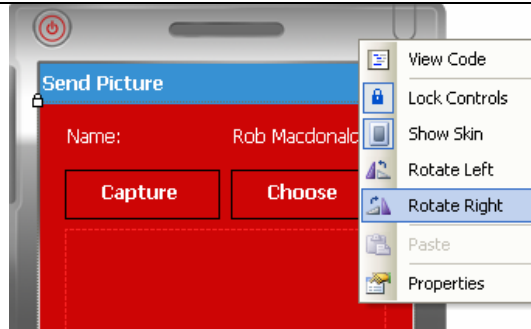
- b. Press and hold **Ctrl** and select **Microsoft.WindowsMobile.Status** and **Microsoft.WindowsMobile.PocketOutlook**.
- c. Click **OK**.

5. Now we'll delete the name label control and replace it with one of our own.

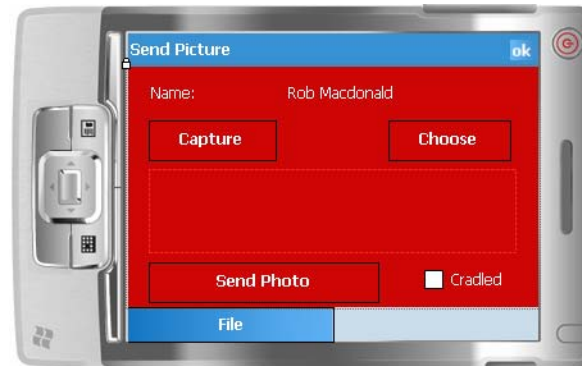
- a. In the design window, right-click on the label control with the text **name** and then select **Delete**.
- b. Next, drag a new **Label** control from the **ToolBox** onto the form where the previous one was deleted.
- c. In the **Properties** window, change the value of the **Text** property to your name.

	As you can see, the number of controls made available to device developers is fairly extensive and includes many of the same ones found in the full .NET Framework. Many of the common controls such as Button, CheckBox, and Label have been available since .NET Compact Framework 1.0, but with the release of the .NET Compact Framework 2.0, new controls have been introduced including MonthCalendar, DateTimePicker, WebBrowser, Notification, DataGrid on Smartphone, as well as the Toolbar for Pocket PC. In addition, the .NET Compact Framework 2.0 enables the development of user controls which can be added to your Toolbox in Visual Studio 2005. 
6. Change the Enabled property of cradledCheckBox.	a. In the Properties drop-down menu, select cradledCheckBox. b. Change the Enabled property of the cradledCheckBox control to False <i>Note: this control will be used later in this lab for read-only purposes.</i> <i>Let's take a moment to look at some of the improvements that have been made to the user interface designers in Visual Studio 2005. Not long ago, all Pocket PC and Smartphone devices had fixed screen sizes, respectively. In both cases, all devices always displayed as portrait.</i> <i>With the introduction of Windows Mobile 2003 Second Edition and continuing with Windows Mobile 5.0, devices can now support different screen resolutions. Additionally, devices are available with portrait, landscape, or square displays, and some devices support dynamically changing displays between portrait and landscape layouts. To simplify the process of designing applications that target these richer display options, Visual Studio 2005 has added several new features to the UI designer. In addition, all managed controls are now more screen-aware by supporting anchoring and docking.</i> <i>Anchoring designates that a control remains a fixed distance from the edge of the form. As the form size and orientation change, the control adjusts its position to remain the same distance from the form's edge.</i> <i>Docking a control designates that the control positions itself directly against the form's edge and that the control occupies the entire edge. Developers can dock multiple controls against the same edge to create a stacked effect. As the user rotates the screen, a docked control automatically adjusts to always occupy the designated edge.</i>
7. Simulate the landscape display mode for the Pocket PC	a. Return to the design view of the application, right-click somewhere on the device emulator skin, and select the Rotate Right option.

application to see how it will look when a user chooses this option.



The emulator now looks like the following:



As you look at the application in landscape mode, notice how some of the controls, like the buttons and empty picture box control, automatically adjusted their positions so that they fit within the new space. If you click on the **Choose** button you can see that its **Anchor** property is set to the **Top** and **Right**, which means that it will automatically adjust its location relative to the Top and Right of the form.

8. Return the emulator to its default portrait display mode.

a. Right-click somewhere on the device emulator skin and select the **Rotate Left** option.

9. Open the SendPicture.cs code and look at the ChoosePicture event handler method.

a. Right-click the form and select **View Code**.

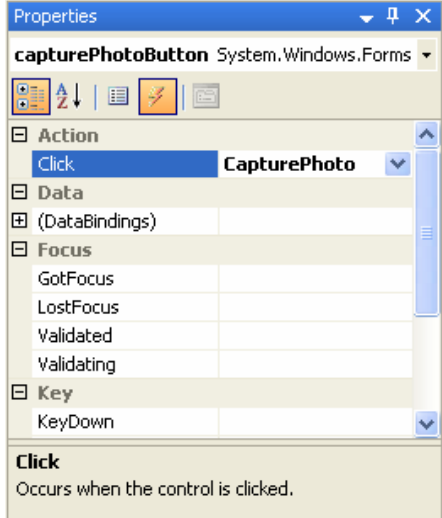
b. Locate the **ChoosePicture** event handler method.

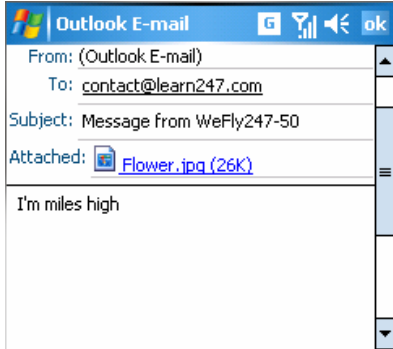
The ChoosePicture event handler method is invoked when the user selects the Choose button on the Pocket PC form.

This method takes advantage of the SelectPictureDialog form made available by Windows Mobile 5.0.



is located in the Forms namespace and can be used to provide a way for a user for selection. Two other forms that are CaptureDialog and ChooseContactDialog.

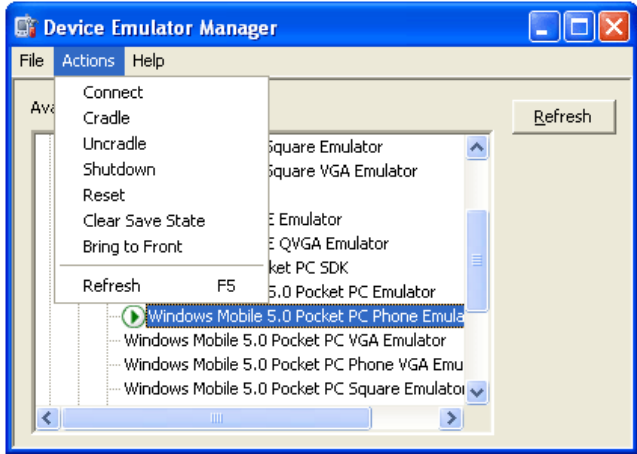
10. Add code.	a. Add the following using statements to the top of SendPicture.cs file: <pre>using Microsoft.WindowsMobile.PocketOutlook; using Microsoft.WindowsMobile.Status;</pre> b. Add the private variable declaration shown below to the beginning of the SendPicture class declaration: <i>Note: the imageFile variable will be used to store the selected picture file</i> <pre>private string imageFile;</pre> c. Modify the code in the ChoosePicture event handler so that it stores its image file in the new variable. The new ChoosePicture method should look like the code below: <i>Note: the new line is the one in bold.</i> <pre>SelectPictureDialog pictureDialog = new SelectPictureDialog(); pictureDialog.ShowDialog(); this.imageFile = pictureDialog.FileName; this.photoPictureBox.Image = new Bitmap(this.imageFile);</pre>
11. Create a Click event handler for the Capture button.	a. Switch back to the form design view for SendPicture.cs by clicking the SendPicture.cs [Design] tab. b. Select the Capture button. In the Properties window, switch the view to show the available events for the button. To see the available events, click on the lightning strike symbol.  c. Find the Click event in the list and enter CapturePhoto to create a handler for that event. Press Enter. This action will add an event handler called CapturePhoto to the source code and then display the SendPicture.cs code file.
12. Add code to the CapturePhoto event handler.	a. Within the CapturePhoto event handler, add the code shown below. <i>This code provides an alternative to using a picture that is already stored on the device. For camera-enabled devices, it will take a photograph and store it in imageFile. Note that this code will fail if executed on an emulator, or a device that is not camera-enabled. You will add code to prevent such errors occurring later in this lab.</i> <pre>CameraCaptureDialog cameraDialog = new CameraCaptureDialog(); cameraDialog.ShowDialog(); this.imageFile = cameraDialog.FileName;</pre>

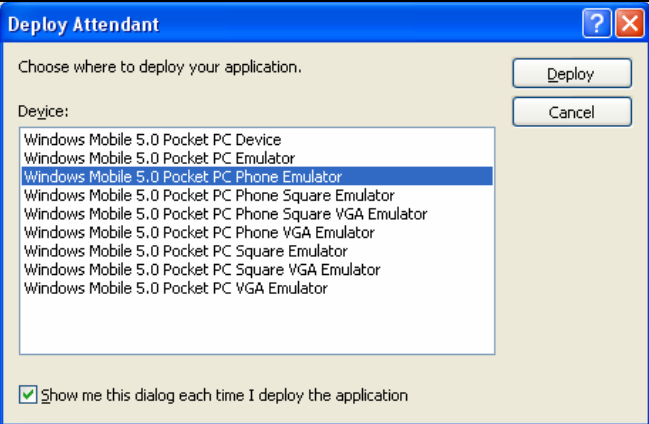
	<pre>this.photoPictureBox.Image = new Bitmap(this.imageFile);</pre>
13. Create a Click event handler for the button labeled Send Photo.	a. Click the SendPicture.cs [Design] tab. b. Select the button labeled Send Photo. c. In the Properties window find the Click event in the list and enter SendPhoto to create a handler for that event. Press Enter.
14. Add code.	a. Once the editor displays the new event handler, insert the implementation shown below into the SendPhoto handler. <pre>// Account name and message details hard coded for simplicity MessagingApplication.DisplayComposeForm("ActiveSync", "contact@learn247.com", "Message from WeFly247-50", "I'm miles high", new string[] { this.imageFile }); this.Close();</pre> <i>This code makes use of Outlook Mobile functionality by displaying a form that allows the user to compose and send an email.</i>  <i>One thing to keep in mind when developing for devices is that their physical characteristics do impact the design of applications, and especially their user interfaces, which follow different design rules from the desktop.</i> <i>Specifications are changing rapidly, but a typical Pocket PC device has a 240x320 (QVGA) screen in portrait mode (although full VGA resolution devices at 480x640 are now readily available) with input via a stylus or detachable keyboard. Expect from 32 to 64 MB RAM (expandable into GBytes via CompactFlash and SD cards) and a processor speed in the region of 500 MHz. Bluetooth, FIR/IrDA (infrared) and WiFi connectivity is increasingly standard, and Phone Editions add GSM and GPRS.</i>

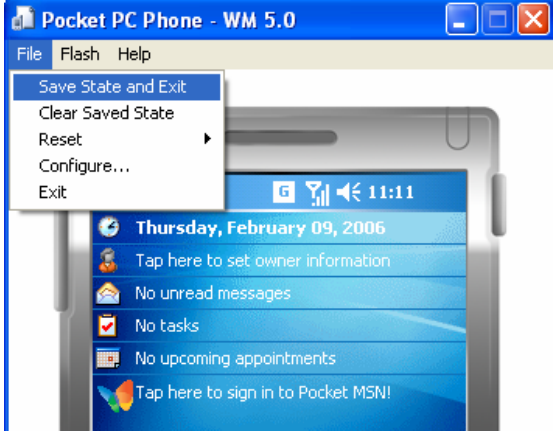
Part 2: Deploying to an Emulator

Like Windows Mobile 5.0, the goal of Visual Studio 2005 is to improve developer productivity. Key tools for reducing development time are the device emulators. Device emulators enable developers to develop and debug applications without needing access to a device. Visual Studio 2005 offers more emulators with more features than ever before.

Let's get started by taking a look at the **Device Emulator Manager** which provides a central utility for managing the many different device emulators.

Tasks	Detailed Steps
1. Open the Device Emulator Manager in Visual Studio.	a. Select Tools Device Emulator Manager. b. Select the Windows Mobile 5.0 Pocket PC Phone Emulator from the list. c. Select Actions Connect to connect to the emulator.  <i>The Device Emulator Manager saves the developer time interacting with the emulators as it provides one-stop-shopping for locating, connecting to, or shutting down any of the installed emulators. The Device Emulator Manager is also an important part of the emulator ActiveSync support because it is used to cradle and uncradle the device emulators, thereby connecting the emulators to and disconnecting them from ActiveSync.</i>
2. To test our application using the emulator, build the solution.	a. From the Visual Studio 2005 menu, select Build Build Solution. b. Correct any errors reported before continuing.
3. Deploy the application to the emulator.	a. Select Debug Start Without Debugging. b. In the Deploy Attendant dialog that appears, select Windows Mobile 5.0 Pocket PC Phone Emulator and click Deploy. <i>Note: the first time you deploy an application that relies upon the .NET Compact Framework 2.0 will take a few minutes.</i>

	
4. Test the solution in the emulator.	a. Once the application is running, click the Choose button. b. Select one of the pre-installed photos from the emulator.  c. Click the Send Photo button. <i>The Mobile Outlook email window will appear, pre-loaded with the arguments supplied in your code.</i>

	 d. Click Send.
5. View the message awaiting synchronization.	<i>The email message created will be stored on the emulator, awaiting synchronization with Outlook running on your desktop PC (if installed).</i> a. To see the contents of your emulator's outbox, click Start Messaging, from the Inbox dropdown select Outbox in order to see your message. b. Close Outlook E-Mail.
6. View save state functionality.	a. Click the File menu item of the emulator. <i>The emulators allow you to save state so that you can shutdown the emulator and restart it again without losing changes made. This is particularly useful if you are using the .NET Compact Framework 2.0 because this will prevent the need to re-deploy it during each test run of your application.</i>  b. Click outside the menu to close it without selecting any options. <i>Visual Studio 2005 introduces a true ARM emulator. As a true ARM emulator, it provides more accurate and complete application testing capabilities, providing a level of device-to-emulator consistency never before available. The ARM emulator runs the same computer instructions, has the same memory layout, and even uses the same drivers as an ARM device</i>

Part 3: Using Windows Mobile 5.0 State Features

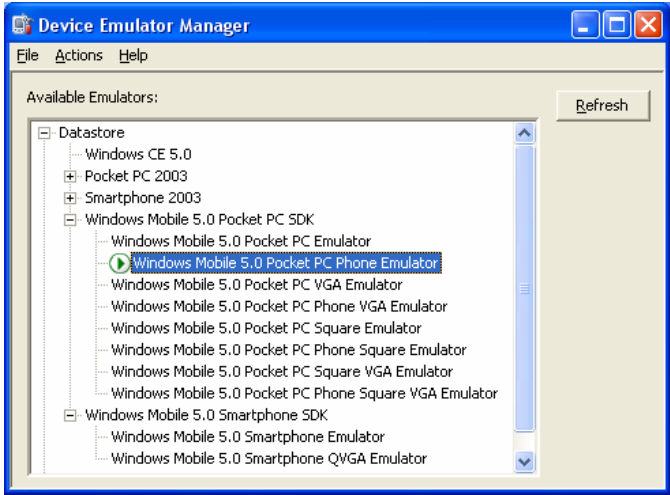
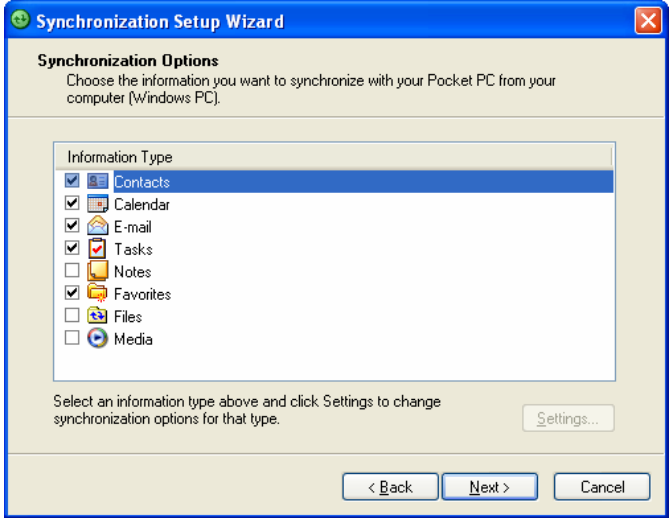
The Microsoft.WindowsMobile.Status namespace provides what is referred to as the State and Notification Broker API, which provides a unified mechanism for storing device, application, and system state information. The following list is just a small sample of the information available through the State and Notification Broker:

- The active application
- The device's ActiveSync status
- Current battery level
- Whether a camera is connected to the device
- The name of the song currently playing in Media Player
- Number of unread e-mail messages
- Whether a call is currently in progress
- Number of missed phone calls

Tasks	Detailed Steps
1. Modify the SendPicture constructor.	a. Switch to Visual Studio 2005. b. In the code window for SendPicture, add the following code directly below the call to InitializeComponent in SendPicture's constructor. <pre>InitializeComponent(); this.capturePhotoButton.Enabled = SystemState.CameraPresent; this.cradledCheckBox.Checked = SystemState.CradlePresent;</pre> <i>This code above makes use of two of the many status flags available from the SystemState class. The first line ensures that the capturePhotoButton is only enabled if a camera is present on the device. The second line of code sets the cradledCheckBox to show whether the device is currently cradled.</i> <i>This is useful information, but of course the device (or emulator) may become cradled after the SendPicture form is loaded. To make sure that the cradledCheckBox is always accurate, it is possible to subscribe to a status event which will fire when a subscribed for event (such as a change of status) occurs.</i>
2. Add code.	a. At the beginning of the SendPicture class declaration, directly after the declaration of imageFile, add the following private variable declaration: <pre>private string imageFile; private SystemState cradled = new SystemState(SystemProperty.CradlePresent, true);</pre>
3. Moving to the constructor, now add an event handler to respond to the CradlePresent status change events, as shown in the code below:	a. Add the following line at the end of the SendPicture constructor: <pre>this.cradled.Changed +=</pre> b. IntelliSense should now display the auto-completion option for the event handler. Hit the TAB key to insert the following. <pre>this.capturePhotoButton.Enabled = SystemState.CameraPresent; this.cradledCheckBox.Checked = SystemState.CradlePresent; this.cradled.Changed += new ChangeEventHandler(cradled_Changed); (Press TAB to insert)</pre> c. Once inserted, IntelliSense should then offer to insert the event handler itself. Type CradleStatusChanged to change the handler name and then TAB again to insert the handler code. The code itself should resemble the code shown below: <pre>void CradleStatusChanged(object sender, ChangeEventArgs args) { throw new Exception("The method or operation is not implemented."); }</pre>

4. Add code.	a. Replace the throw statement in the handler above with the code below: <pre>this.cradleCheckBox.Checked = SystemState.CradlePresent;</pre>
5. Build the solution.	a. Click Build Build Solution. b. Correct any errors if reported.

Part 4: Testing the Application for State

Tasks	Detailed Steps
1. Deploy the application to the emulator.	a. Select Debug Start Without Debugging. b. In the Deploy Attendant dialog that appears, select Windows Mobile 5.0 Pocket PC Phone Emulator and click Deploy. Wait until the application deploys to the emulator.
2. Load the Device Emulator Manager and use the Cradle feature for simulation.	a. In Visual Studio 2005, select Tools Device Emulator Manager. b. In the Device Emulator Manager dialog right-click Windows Mobile 5.0 Pocket PC Phone Emulator and select Cradle. 
3. Go through the Synchronization Setup Wizard.	a. At the Microsoft ActiveSync dialog, click OK. b. ActiveSync should now present you with the Synchronization Setup Wizard. c. Click Next to advance the wizard past the welcome page. d. The next page, Synchronization Options, presents us with the possible data that can be synchronized with the desktop computer. Leave the settings as they are and click Next. <i>Note: the following steps will not work because you are running in a virtual lab environment, but feel free to experiment on your own.</i>  e. Click Finish in the Completing the Pocket PC Sync Setup Wizard page.
4. Observe that after synchronization is	a. Make both the ActiveSync and emulator application windows visible and observe

complete the cradledCheckBox changes from Unchecked to Checked.	the progress of the synchronization. 
5. Use the SendPicture form to choose a picture and send an email.	a. While the emulator is cradled, ActiveSync will synchronize with Outlook on your desktop PC (if installed) which will then send the email with the photo attachment. <i>Note: this step will not work if you are running in a virtual lab environment, but feel free to experiment on your own.</i>
6. Uncradle the emulator from ActiveSync.	a. In Device Emulator Manager, right click on Windows Mobile 5.0 Pocket PC Phone Emulator and select Uncradle.

Exercise 2: Developing a Windows Mobile 5.0 Smartphone Application

Scenario

This exercise shows how to create a Smartphone application that mimics some of the same functionality that we developed in the previous Pocket PC exercise. As you go through this Smartphone exercise notice how similar the development experience is to developing for the Pocket PC.

Smartphones have good connectivity but screen size, memory and processor speed is even more constrained than that of a Pocket PC. Keep these considerations in mind when developing for Smartphone devices.

Part 1: Sending SMS Messages

In this section we will show how you can send SMS messages in your mobile applications.

Tasks	Detailed Steps
1. Load Exercise2.sln.	- Switch to Visual Studio 2005. - Click File Open Project/Solution and navigate to the Managed Guided Tour\Exercise 2 folder on the desktop. - Select Exercise 2.sln and click Open.
2. Study the SendSms method.	- In the Solution Explorer right-click the Form1.cs file under the SmartSms project and select View Code. - Navigate to the SendSms event handler. <i>The SendSms method is an event handler that is raised when a user clicks a menu item. The code snippet below shows an example of how to construct and send a new SMS message.</i> <i>The only two things that you need in order to send an SMS message are the phone number and a string containing the message body. This example, taken from the Microsoft WeFly247-50 project, shows how you can create a proprietary message format to transfer information from one phone to another. This is not necessary for an ordinary SMS message.</i> <pre>string emulatorSmsSelfCallNumber = "+14250010001"; DateTime time = DateTime.Now; // Message Format is: // Prefix Category Seat Hours Minutes Seconds Order string message = string.Format("M: Drink A1 {0} {1} {2} Orange Juice", time.Hour, time.Minute, time.Second); SmsMessage sms = new SmsMessage(emulatorSmsSelfCallNumber, message); sms.Send();</pre>

Part 2: Intercepting SMS Messages

Tasks	Detailed Steps
1. Study the Main method from the SmsHandler project.	a. In the Solution Explorer right-click the Program.cs file under the SmsHandler project and select View Code. <i>Notice that we have included a using statement at the top of the source file for the MessageInterception namespace which is located in the PocketOutlook namespace.</i> b. Navigate to the Program class where a private variable of type MessageInterceptor is declared (shown below). This variable will be configured to handle the SMS message interception. <pre>private static MessageInterceptor interceptor; static void Main(string[] args) interceptor = new MessageInterceptor(InterceptionAction.NotifyAndDelete, false); // false means: use a background thread using (interceptor) { interceptor.MessageCondition = new MessageCondition(MessageProperty.Body, MessagePropertyComparisonType.StartsWith, "M:", true); interceptor.MessageReceived += new MessageInterceptorEventHandler(ProcessSmsMessage); interceptor.EnableApplicationLauncher("WeFly247-50.SMS");</pre> <i>The first line of code instantiates the interceptor variable and configures it to intercept an SMS message, notify the application, and then delete the message.</i> <i>In the using block, the interceptor is first filtered to listen only for SMS messages whose body begins with “M:”. It is then hooked up to fire an event handler called ProcessSmsMessage. Lastly, the EnableApplicationLauncher registers this application to be launched (if it isn’t running) when the event fires.</i>

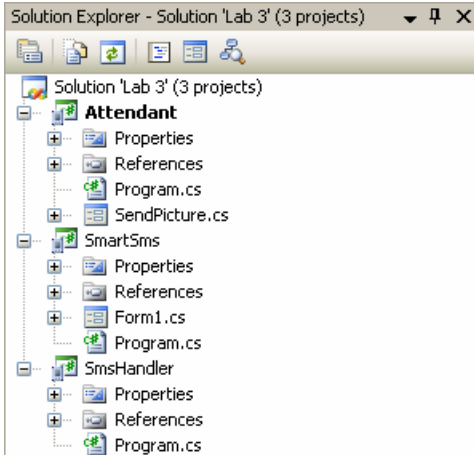
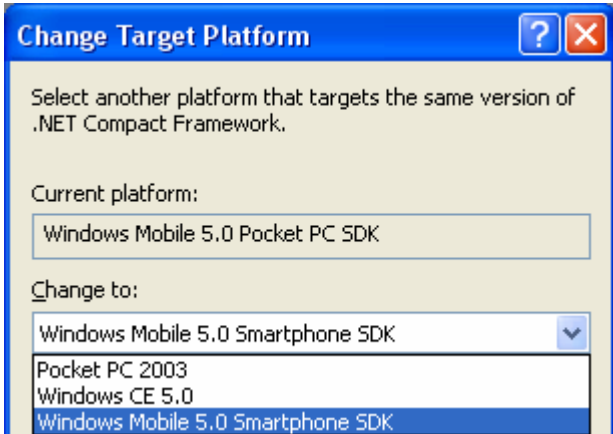
Part 3: Working with Outlook Mobile

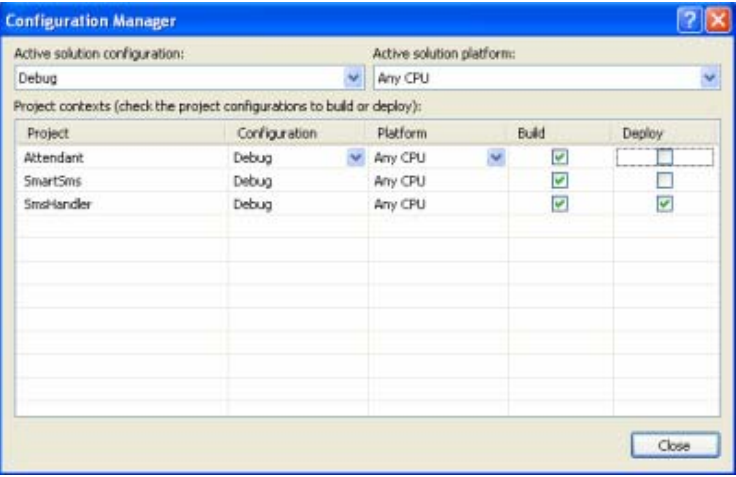
In this section we will show how you can create new Outlook Mobile appointments.

Tasks	Detailed Steps
1. Study the ProcessSmsMessage method.	a. In the Program.cs file locate the section of code where a new appointment is built (shown below). <pre>// Build Appointment Appointment appointment = new Appointment(); appointment.Subject = message[1]; appointment.Location = message[2]; appointment.Start = messageTime; appointment.End = messageTime; appointment.Body = message[6]; OutlookSession outlookSession = new OutlookSession(); using (outlookSession) { outlookSession.Appointments.Items.Add(appointment); }</pre> <i>This code builds an Outlook Mobile appointment using the SMS message content that was intercepted, and adds the appointment into the Appointments collection of an OutlookSession object</i>

Part 4: Sharing Code between Platforms

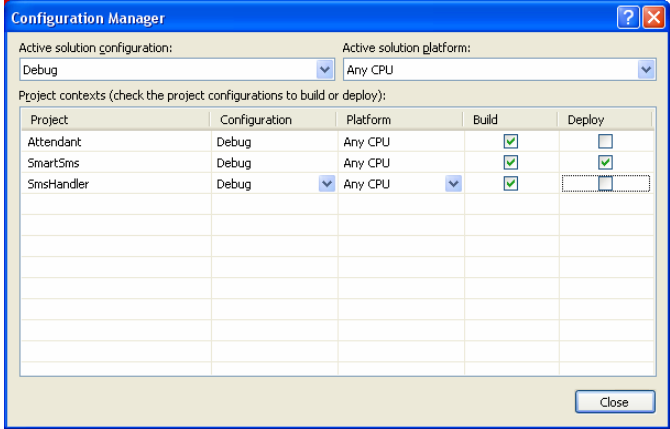
Visual Studio 2005's support for multiple-platform projects makes targeting multiple device platforms easier than ever. This allows developers to use the same code for both devices while maintaining separate user interfaces.

Tasks	Detailed Steps
1. Open Solution Explorer and view the listed projects.	a. Expand all the project nodes in the Solution Explorer (if needed) and view the layout.  Consider a scenario where we want to target both Windows Mobile 5.0 Pocket PC and Smartphone devices in a single project and share common code between them. Solution Explorer shows three projects – Attendant, SmartSms, and SmsHandler. The Attendant project contains the code for a Pocket PC application. The SmsHandler project represents an application that intercepts incoming SMS messages and then creates new appointments in Outlook Mobile. This code can be used for both Pocket PC and Smartphone applications. Finally, the SmartSms project represents a Smartphone application that can select pictures, send them via email, send SMS messages, and even show the appointments that were created by the SmsHandler interception code. The next few tasks will demonstrate how to make the switch between deploying the Pocket PC application to the Smartphone application.
2. To start with, change the target platform for the SmsHandler project from the Pocket PC to the Windows Mobile 5.0 Smartphone SDK.	a. Select the SmsHandler project node in Solution Explorer. b. From the Visual Studio 2005 menu, select Project Change Target Platform. c. In the Change Target Platform dialog select Windows Mobile 5.0 Smartphone SDK in the Change to: dropdown.  d. Click OK to close the dialog and Yes to accept the warning message regarding reopening the solution.

3. Change the StartUp Project to SmsHandler.	a. In Solution Explorer right-click on SmsHandler and select Set as StartUp Project.
4. Change the build configuration to stop deployment for the Attendant Pocket PC application.	a. Select the Build Configuration Manager menu. b. Uncheck the Deploy checkbox for Attendant.  c. Click Close.
5. Deploy the SmsHandler project to the Smartphone emulator so that future SMS messages will be intercepted and saved as appointments.	a. Click Debug Start Without Debugging. b. When the Deploy SmsHandler dialog appears, select Windows Mobile 5.0 Smartphone Emulator and then click Deploy.

Part 5: Running the SmartSms Application

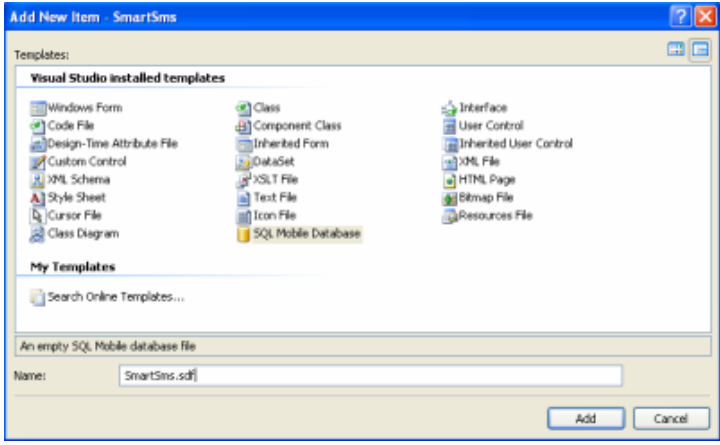
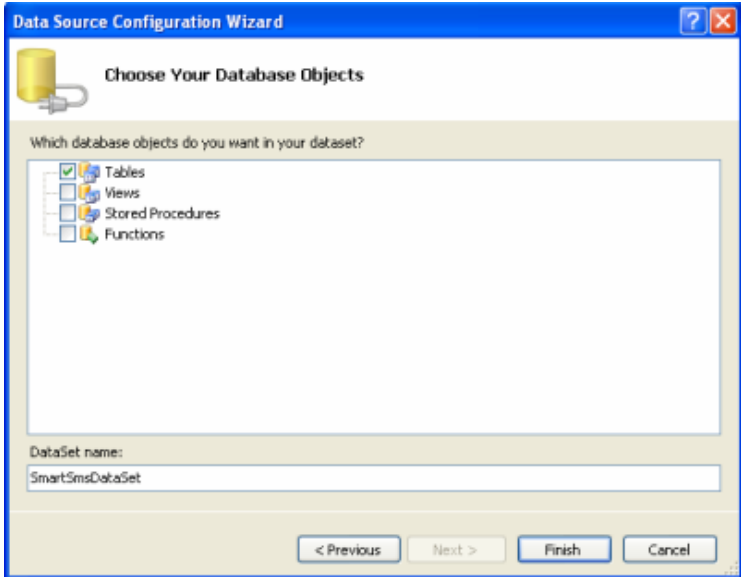
Now let's take a look at the **SmartSms** application in action.

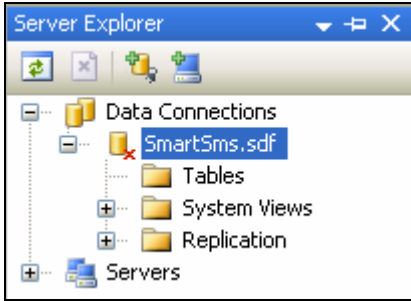
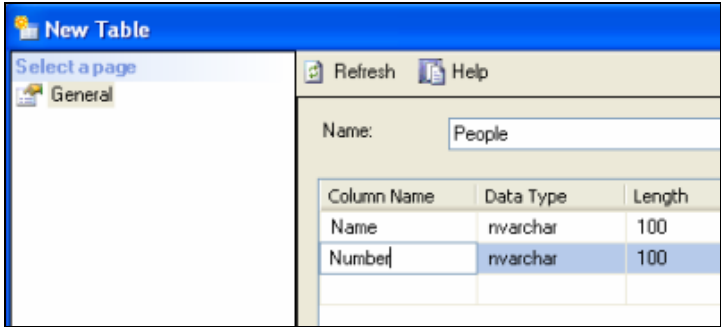
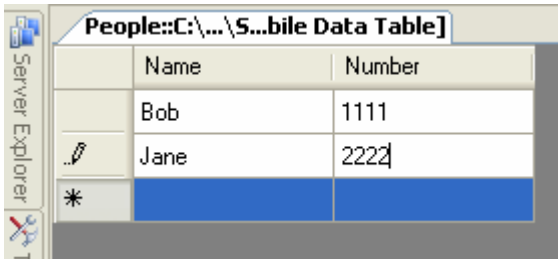
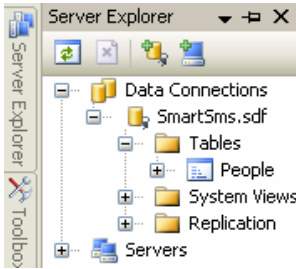
Tasks	Detailed Steps
1. Change the StartUp Project to SmartSms.	a. In the Solution Explorer, right-click the SmartSms project node and select Set as StartUp Project.
2. Change the build configuration to only deploy the SmartSms application.	a. From the Visual Studio 2005 menu bar, select Build Configuration Manager. b. In the Configuration Manager dialog, uncheck the Deploy option for the SmsHandler project and check the Deploy option for the SmartSms project.  c. Click Close.
3. Deploy the SmartSms project to the Smartphone emulator and test the functionality that sends SMS messages and view the resulting appointments created by the SmsHandler running in the background.	a. Select Debug Start Without Debugging. b. In the Deploy SmartSms dialog that appears, select Windows Mobile 5.0 Smartphone Emulator and click Deploy. c. Use the Menu to select SMS Message. You will see the wait cursor as SmsHandler launches and processes the message. When it has finished, send another SMS Message. Then use the Show Appointments menu to list the appointments created by SmsHandler. 
4. Return to the <i>Exercise2.sln</i> in Visual Studio.	a. Click the Done menu item of the SmartSms application and return to Visual Studio.

Part 6: Using Microsoft SQL Server 2005 Mobile Edition

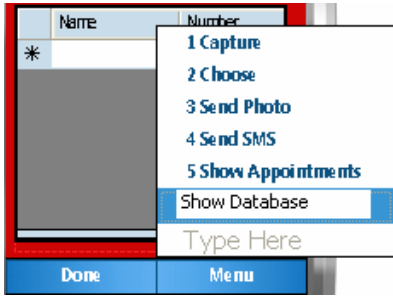
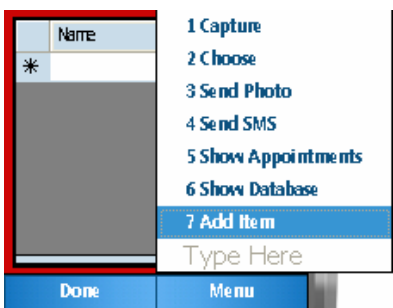
Microsoft SQL Server 2005 Mobile Edition (SQL Server Mobile), the successor to Microsoft SQL Server 2000 Windows CE 2.0 (SQL Server CE), is a feature-rich compact database that is designed to support an expanded list of smart devices and Tablet PCs. The smart devices include any device that runs Microsoft Windows CE 5.0, Microsoft Mobile Pocket PC 2003, Microsoft Mobile Version 5.0 Pocket PC, or Microsoft Mobile Version 5.0 Smart Phone. This additional device support enables developers to use the same database functionality across many devices.

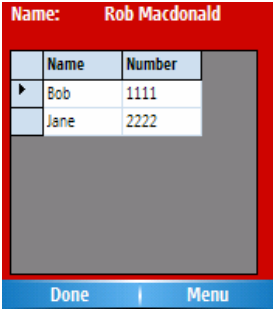
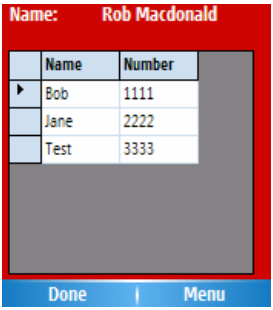
In this section you will have a chance to work with a SQL Server Mobile database while integrating it with the **SmartSms** application

Tasks	Detailed Steps
1. Create a new SQL Mobile database as part of the SmartSms project.	a. In Solution Explorer right-click on the SmartSms project node and select Add New Item. b. The Add New Item dialog box will appear.  c. Select the SQL Mobile Database project template and name the new database SmartSms.sdf. d. Click the Add button to create the database file and load the Data Source Configuration Wizard.
2. Configure the data source for the SmartSmsDataSet object using the Data Source Configuration Wizard.	a. Select the Tables database object and then select Finish to complete the process of adding a new mobile database to the SmartSms application.  <i>The steps that we have completed thus far have added a mobile database to our</i>

	application, added references to the needed assemblies, and even created a typed dataset class which we will finalize the setup of shortly. The next few steps will finalize the creation of the mobile database and complete the definition of the SmartSms dataset.
3. Add a new table named People to the SmartSms database with a column for Name and Number.	a. Open Server Explorer by selecting View Server Explorer from the main menu of Visual Studio. Server Explorer should already list the SmartSms database.  b. Right-click on the Tables node and select the Create Table option. This will load the New Table dialog box and allow you to name the database and create columns. c. Name the table People and create a column named Name and one named Number.  d. Select OK to finish creating the new database table.
4. Add a few rows to the SmartSms database so that we have something to display once the application runs.	a. In the Server Explorer, expand the Tables node, right-click the People table and select Open. This will open the database in a mode that allows you to edit the contents.  b. Add a few rows of example data by simply selecting each row and column and entering some text.
5. Use the Server Explorer to add a DataTable and TableAdapter to the SmartSms project.	a. In the Server Explorer, navigate to the People table.  b. In the Solution Explorer, double-click SmartSmsDataSet.xsd in the SmartSms

	project to bring up the Dataset Designer. c. Drag the People node on to the Dataset Designer surface to have Visual Studio create a typed DataTable and TableAdapter for the SmartSms database.  Now we need a way to view the database information in the SmartSms application and allow updates to be made to it.
6. Add a DataGrid control to the SmartSms application form.	a. In the Solution Explorer open the Form1 form of the SmartSms application in design mode. b. From the Toolbox drag a DataGrid control onto the design surface. Resize the DataGrid so that it fits within the client space of the smartphone and covers the existing ListBox control. 
7. Set the DataSource property of the DataGrid control to the SmartSms database file.	a. In the Properties window for the DataGrid control, select the combo box to the right of the DataSource property and select the People table from the SmartSmsDataSet data source.  The data source will be bound to the DataGrid control and instances of the SmartSmsDataSet and PeopleTableAdapter are added to Form1 of the SmartSms project.
8. Change the initial state of the DataGrid control so that it is not visible upon application startup.	a. In the Properties window for the DataGrid control, set the Visible property to False.
9. Add a new menu item	a. In the design view of Form1.cs click on the object labeled Menu and click the Type

to make the DataGrid control visible while the application is running.	Here option to add a new option to the list. b. Type Show Database to create a new menu item choice.  c. Once you have typed in the name of the new menu item choice, press Enter to save your changes. d. In the Properties window, change the Name of the new menu item choice to menuItemShowDatabase to help make the code easier to read.
10. Add a click handler for the Show Database menu item choice.	a. Double-click the Show Database menu item choice in the Form1 designer to have Visual Studio create a handler method for you. Add the line of code to make the DataGrid visible as shown below. <pre>private void menuItemShowDatabase_Click(object sender, EventArgs e) { this.dataGrid1.Visible = true; }</pre>
11. Add a new menu item to create new database entries.	a. Switch to the design view of Form1.cs. b. Click on the object labeled Menu and click the Type Here option to add a new option to the list. c. Type Add Item as the new menu item choice. d. In the Properties window, change the Name of the menu item to menuItemAdd to help make the code easier to read. 
12. Add a click handler for the Add Item menu item choice.	a. Double-click the Add Item menu item choice in the Form1 designer to have Visual Studio create a handler method. Add the following lines of code to create a new database record as show below: <pre>private void menuItemAdd_Click(object sender, EventArgs e) { // Add a new record to the database. this.peopleTableAdapter.Insert("Test", "3333"); // Updating the dataset will force the DataGrid to update. this.peopleTableAdapter.Fill(this.smartSmsDataSet.People); }</pre>

	<i>So far we have created a new SQL mobile database to deploy with the SmartSms project, added a way to view the database information by binding a DataGrid control to a dataset, and enabled the user to add new records to the database. In this example we add a new row to the database using some test data, but in practice you could easily add some controls to the application in order to gather user input instead.</i> <i>SQL Server Mobile also enables data to be updated simultaneously on the device and the server. Even if the device is out of range of a wireless network or wired network connections are not available, the SQL Server Mobile-based application on the device can be used continually, storing and managing the data within the device's local database. The data can later be synchronized when the device connection to the host SQL Server is reestablished. SQL Server Mobile supports two options for data synchronization—Merge Replication and Remote Data Access (RDA). If you are interested in more information on SQL Server Mobile and developing data-driven mobile applications, please visit the Microsoft SQL Mobile site at http://msdn.microsoft.com/sql/mobile.</i>
13. Let's take a look at the modified SmartSms application in action.	- Deploy the SmartSms application to the Smartphone emulator by selecting Debug Start Without Debugging. - When the Deploy SmartSms window appears, make sure that Windows Mobile 5.0 Smartphone Emulator is selected and click Deploy. - Switch to the Smartphone Emulator. <i>Note: the application may take a few moments to deploy and load.</i> - Once the application starts, click Menu and select the Show Database menu item. The DataGrid control you added earlier will appear showing the default test data that was entered by hand earlier.  - Next, click Menu and select the Add Item menu item to add a new entry to the database and notice that the DataGrid was updated as well. 

Summary

In this guided tour you had the opportunity to learn about developing managed applications for the Windows Mobile 5.0 platform. Developing applications that target Windows Mobile 5.0 Pocket PC and Smartphone devices has never been easier with many of the improved productivity gains found in Visual Studio 2005 and the Windows Mobile 5.0 SDKs.

Developing Managed Applications for Pocket PC and Smartphone Devices

Although it was not demonstrated in this guided tour, devices that have WiFi connectivity to a LAN based database server can make use of ADO.NET to communicate directly to the database over the wireless network. In Visual Studio 2005, DataSets have been enhanced considerably to approximate the full .NET DataSet capability, including the ability to use Typed DataSets.

Visual Studio 2005 simplifies device development through several productivity tools such as data designers, a faster debugger, and improved user interface designers, which include support for developers to view application presentations across different screen orientations, resolutions, and devices. Improved emulators, including a new ARM emulator, simplify application testing by more accurately representing the device environment, thereby reducing the number and level of tests developers need to run directly on a device.