Step by Step: Building a .NET Compact Framework Application for a Windows Mobile-based Device Using Visual Studio 2005

Microsoft Corporation

June 2006

Applies to:
- Microsoft .NET Compact Framework version 1.0
- Microsoft SQL Server 2005 Mobile Edition (SQL Mobile)
- Microsoft Visual Studio 2005
- Windows Mobile version 5.0 software for Pocket PCs
- Windows Mobile version 5.0 software for Smartphones

Summary: Learn how to reuse your existing Microsoft Visual Studio and Microsoft .NET Framework skills to develop a line-of-business application for a Windows Mobile–based device in this self-paced, hands-on lab (HOL). This HOL will take 60 minutes to complete. (42 printed pages)

Download the MEDC06_HOL201.msi from the Microsoft Download Center.

Contents

The following applications are required to run this HOL:

- Microsoft Windows XP Professional
- Internet Information Services (IIS)

  This HOL uses IIS for Windows XP Professional. Be sure that the IIS component is installed. See Appendix B for instructions about how to install the IIS component.

- Visual Studio 2005

  This HOL requires Visual Studio 2005 Standard, Professional, or Team System Editions. It will not work with any of the Express Editions. If you do not have the correct edition of Visual Studio 2005, find out how you can acquire it from the Visual Studio 2005 Developer Center.

- Microsoft ActiveSync 4.0 or later

  ActiveSync 4.0 or later allows for connectivity between a Windows Mobile–based device and your computer.

- Windows Mobile 5.0 SDKs.

  The Windows Mobile 5.0 SDKs for Pocket PC and Smartphone enable development for Windows Mobile–based devices in Visual Studio 2005:

  Download and install Windows Mobile 5.0 SDK for Pocket PC.

  Download and install Windows Mobile 5.0 SDK for Smartphone.

- Set up your computer

Follow the directions in Appendix A and Appendix B to set up your computer for this HOL.

Credentials Used

The following Web services are used in this HOL:

- http*://your computer's IP address/*ProductData/UpdateService.asmx

Introduction

In this lab, you will learn how to use your existing Visual Studio and .NET Framework skills to develop Windows Mobile–based device applications. You will start by building a simple Smartphone application. You will then access data that is stored in SQL Server 2005 Mobile Edition (SQL Mobile) to provide data-browsing capability, including displaying image files. By using the Microsoft Pocket Outlook Object Model API (POOM), you will send an e-mail message to a Microsoft Office Outlook Mobile contact. You will finish by adding the ability to call a Web service that provides updated data to the SQL Mobile database and downloads additional image files for the new data.

You will be creating a Smartphone application that sales representatives use from a high-end art and photography dealer. At the dealership, sales representatives spend a great deal of time on the road working closely with individual clients. These sales representatives must always have easy access to client information and must be able to work effectively when they are traveling.

The example application provides sales representatives with an easy way to browse through a list of available art and photo products; a way to send a picture of the product to a customer—whose contact information is stored in the Outlook Mobile Contacts list; and a way to update the mobile product list with fresh data, including new images.

Lab 1: Building a .NET Compact Framework Application for a Windows Mobile-based Device Using Visual Studio 2005

Upon completion of this lab, you will be familiar with the process to create a Windows Mobile application by using Visual Studio 2005 and SQL Mobile. You will also be familiar with the basics of working with the new Windows Mobile 5.0 managed application programming interfaces (APIs).

In this HOL, you will perform the following exercises:

- Creating a Smartphone application and using data from SQL Mobile
- Adding support for sending e-mail to a contact
- Accessing Web services and retrieving remote image files

Exercise 1: Creating a Smartphone Application and Using Data from SQL Mobile

In this exercise, you will use Visual Studio 2005 to create a Smartphone application. You will associate commands with a mobile device's soft keys, and you will add the ability to scroll through the list of available products (a list of images). You will create a data source for an existing SQL Mobile database, and then test the application in the Windows Mobile 5.0 emulator.

In the following procedure, you will create a new Windows Mobile 5.0 project in Visual Studio.

To create a Windows Mobile project in Visual Studio

1. Open Visual Studio 2005 by clicking **Start** | **All Programs** | **Microsoft Visual Studio 2005** | **Microsoft Visual Studio 2005**.
2. In Visual Studio, click **File** | **New** | **Project**. The **New Project** dialog box appears.
3. On the **New Project** dialog box, under **Project types**, expand **Visual C#**, expand **Smart Device**, and then select **Windows Mobile 5.0 Smartphone**, as shown in Figure 1.

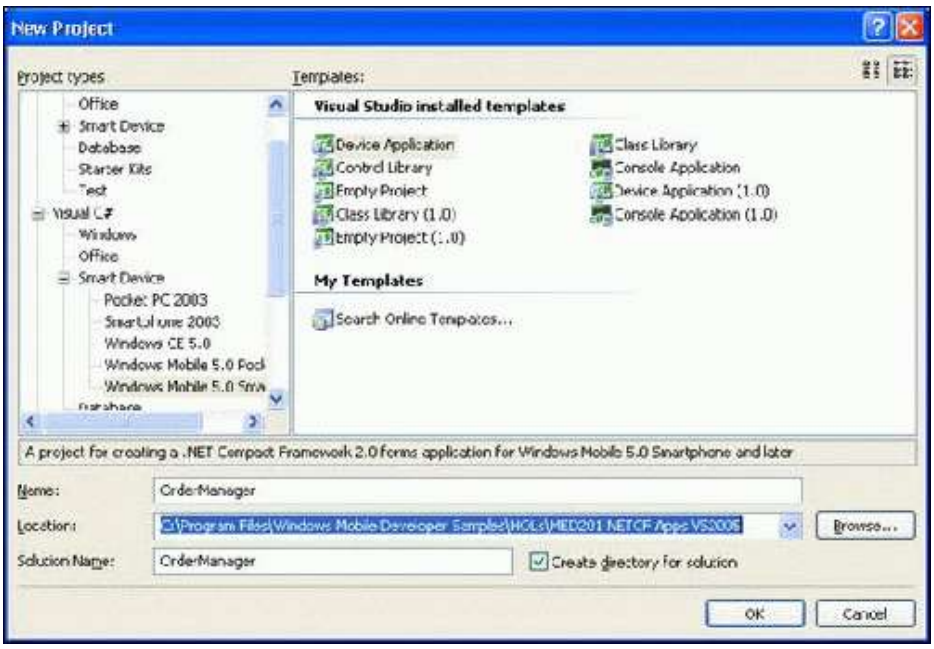

Figure 1. Selecting the Windows Mobile 5.0 Smartphone device. Click the thumbnail for a larger image.

4. In the **New Project** dialog box, under **Visual Studio installed templates**, select **Device Application**.
5. In the **Name** box, type **OrderManager**.
6. Click **Browse**. Browse to C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL201\Exercises, and then click Open.
7. Click **OK**.

   The project is created and loaded into Visual Studio, as shown in Figure 2.

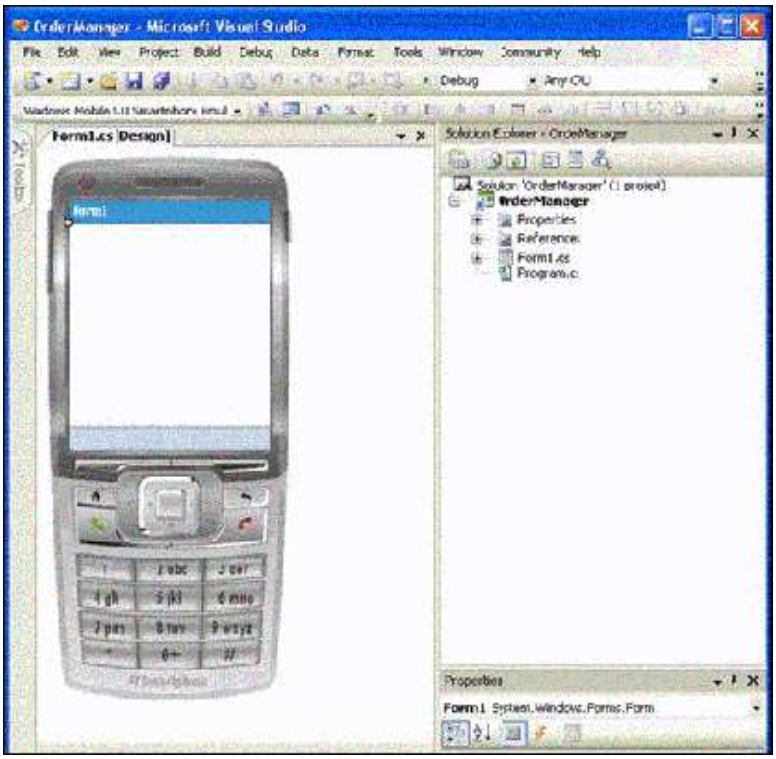

Figure 2. The OrderManager project. Click the thumbnail for a larger image.

In the following procedure, you will add an existing SQL Mobile database as a new data source. Later in this exercise, you will use this data source to add data-browsing capabilities to the application.

To add a new data source

1. In Visual Studio, click **Data | Add New Data Source**. The **Data Source Configuration Wizard** starts.

2. On the **Choose a Data Source type** page, select **Database**, and then click **Next**, as shown in Figure 3.

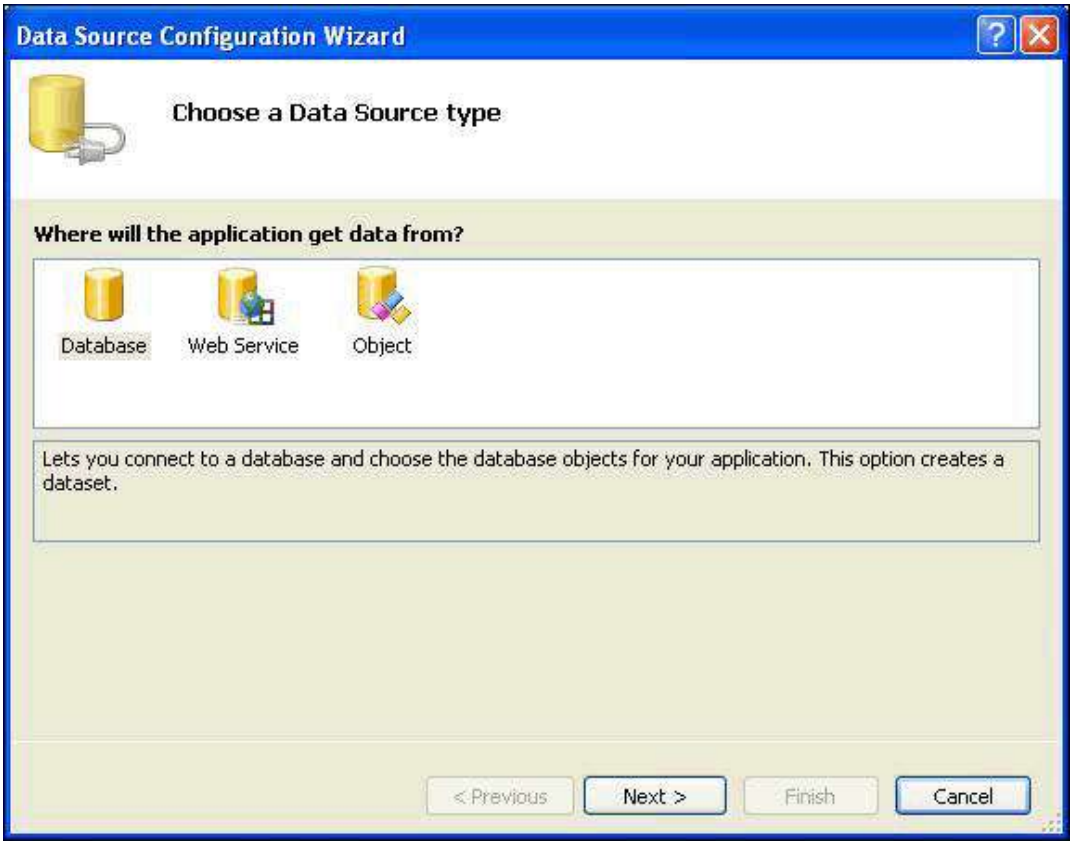

Figure 3. The Choose a Data Source type page in the Data Source Configuration Wizard.

3. On the next page, click **New Connection**.
4. If the **Add Connection** dialog box appears, click **Change**.
5. On the **Choose Data Source** dialog box, select **Microsoft SQL Server Mobile Edition**, and then click **OK**.

> **Note** If the **Add Connection** dialog box did not appear in step 4, but the **Choose Data Source** dialog box appeared immediately, click **Continue**, as shown in Figure 4.

Figure 4. Choosing the data source

6. On the **Add Connection** dialog box, verify that **My Computer** is selected as the **Data Source**, as shown in Figure 5.

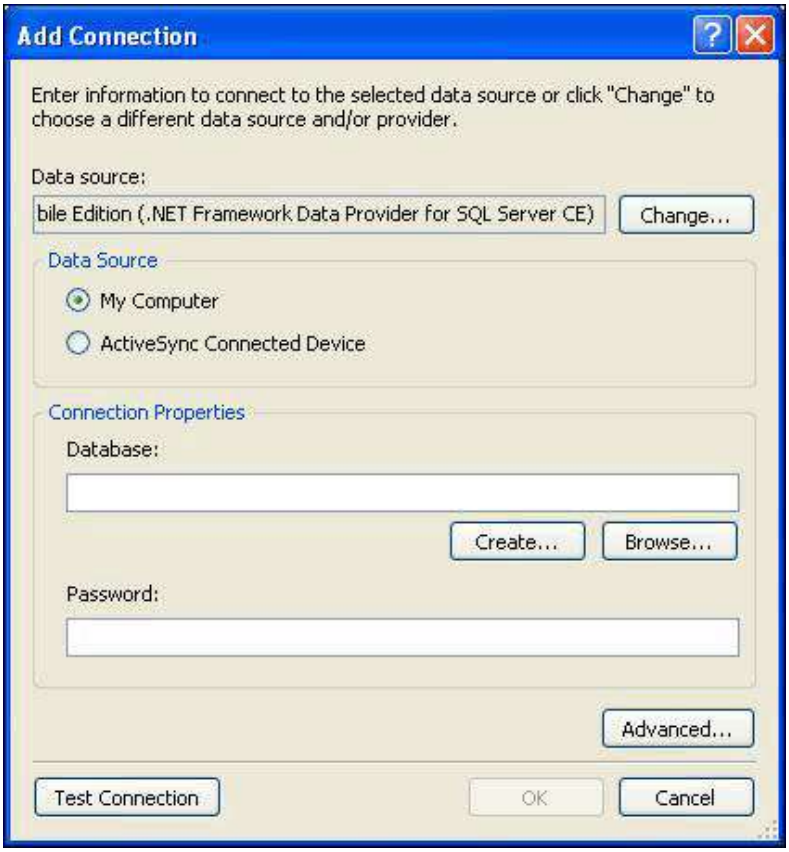

Figure 5. Add Connection dialog box

7. On the **Add Connection** dialog box under **Connection Properties**, click the **Browse** button, which is by the Database box.
8. Browse to C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL201\Exercises\Orders.sdf, as shown in Figure 6, and then click **Open**.

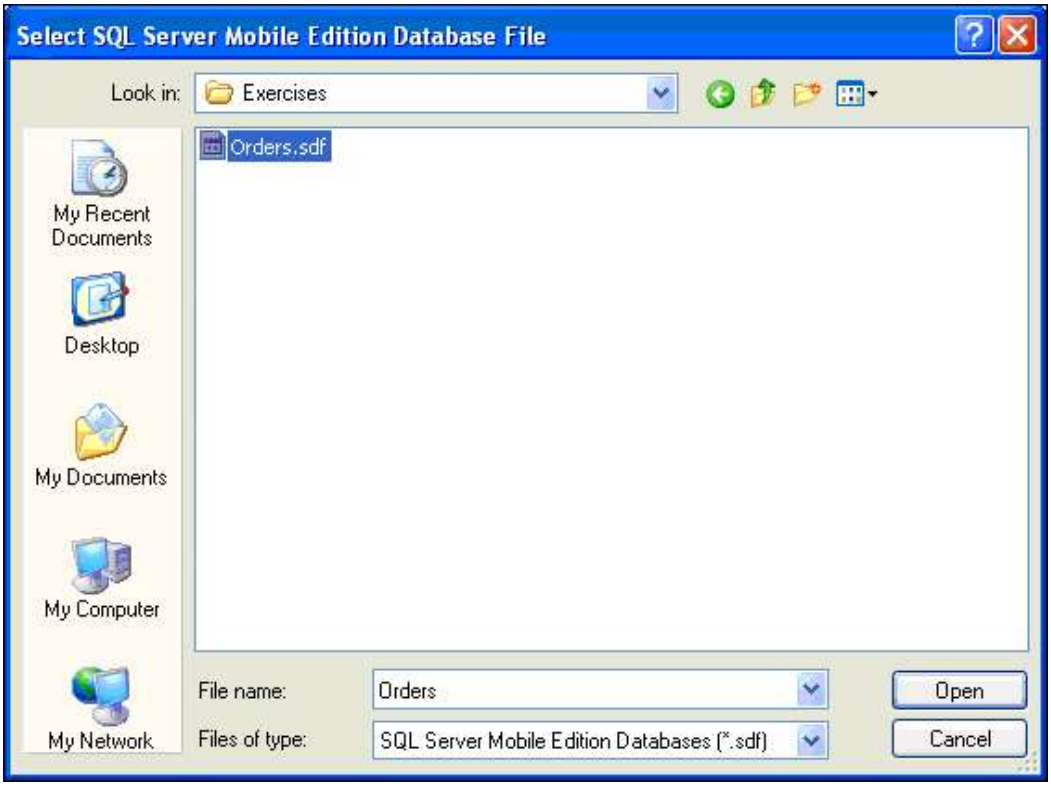

Figure 6. Browse to Orders.sdf

9. On the **Add Connection** dialog box, click **OK**.

Note SQL Mobile databases can exist either on a desktop computer or on a Windows Mobile–based device. They are portable between the two platforms. Visual Studio 2005 supports connecting to and managing the databases on both platforms.

10. On the **Choose your data connection** page, click **Next**, as shown in Figure 7.

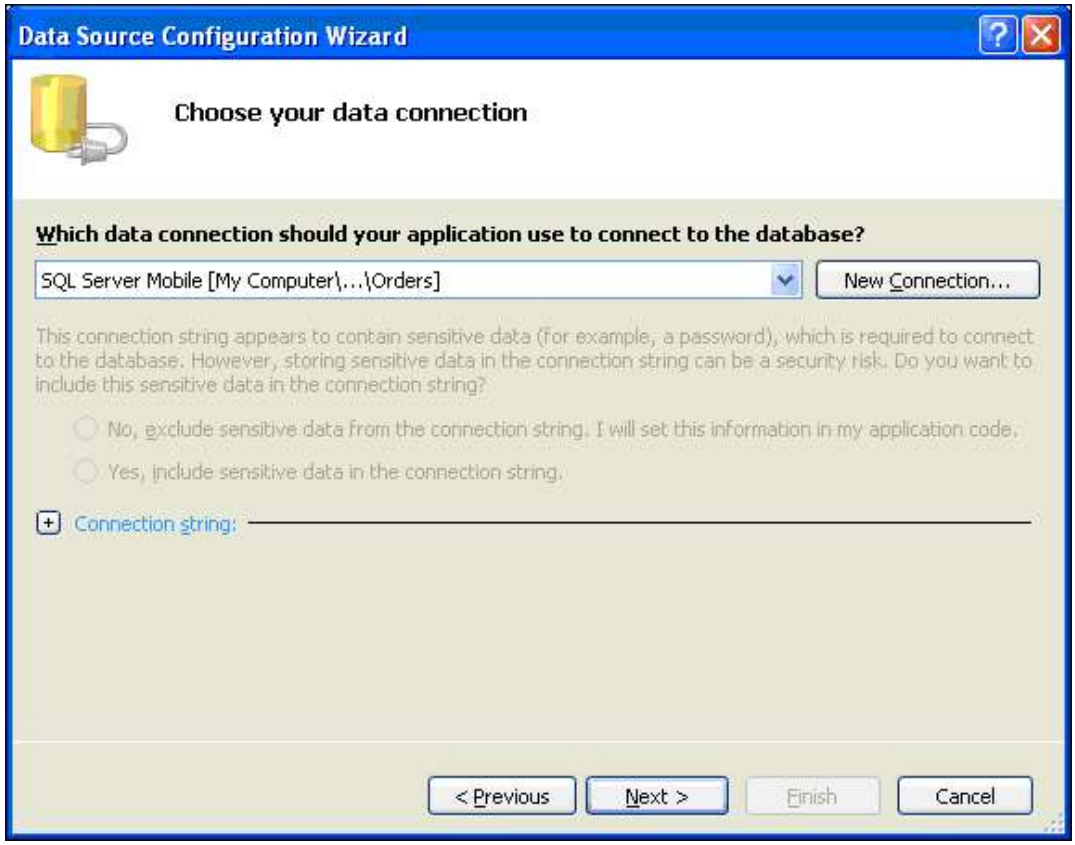

Figure 7. Choose your data connection page

11. In the **Microsoft Visual Studio** dialog box that prompts you to copy the local data file to your project and modify the connection, click **Yes**.

    Clicking Yes makes the database part of the Visual Studio project. The database is automatically copied to the device when the project is deployed. The **Data Source Configuration Wizard** page appears.

12. On the **Data Source Configuration** Wizard page, expand **Tables**, and then select **Inventory**, as shown in Figure 8.

Figure 8. Choosing the database objects

13. Click **Finish**.

    The **Orders.sdf** SQL Mobile database and the **OrdersDataSet.xsd** data source objects appear in **Solution Explorer**, as shown in Figure 9.

Figure 9. The Orders.sdf SQL Mobile database and the OrdersDataSet.xsd data source object

In the following procedure, you will create a new form to allow the user to scroll through the list of available products one by one. You will place controls on the form to display the database fields. Because the company using this mobile application sells pictures and artwork, this application displays a picture of each item on the form.

To create a data-browsing form

1. In **Solution Explorer**, right-click **Form1.cs**, and then click **Delete** on the shortcut menu.
2. Click **OK** when Visual Studio displays a message to confirm that Form1.cs will be permanently deleted.
3. In **Solution Explorer**, be sure the **OrderManager** project is selected, and then click **Project | Add Windows Form**.
4. In the **Add New Item** dialog box, in the **Name** box, change the name to **ProductForm.cs**, and then click **Add**, as shown in Figure 10.

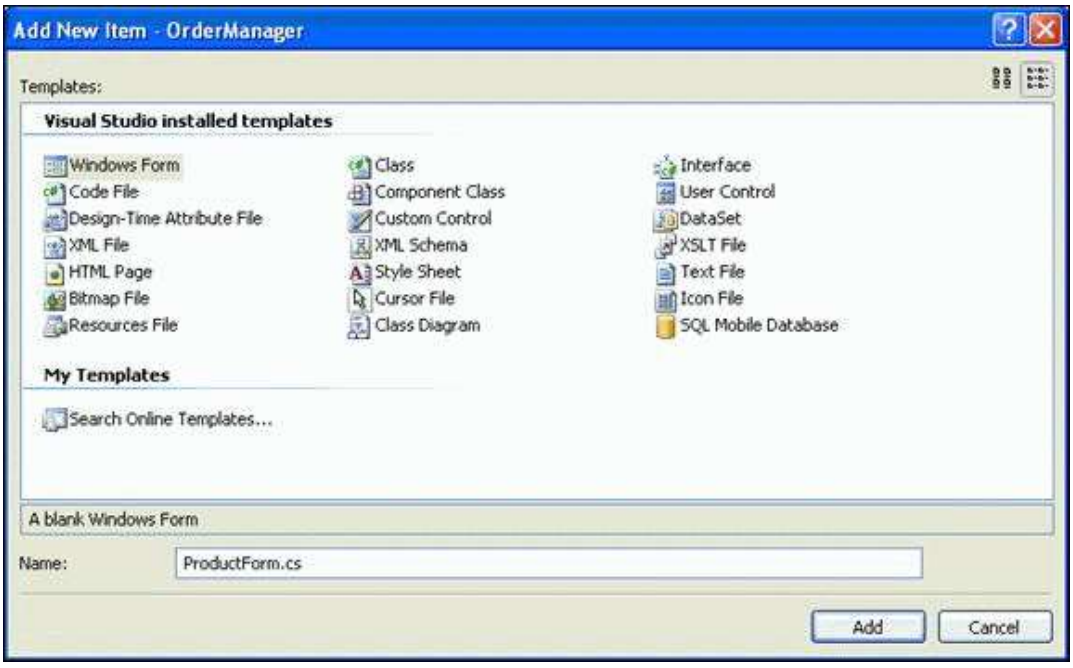

Figure 10. Adding ProductForm.cs

The form designer appears with the new form loaded in the Smartphone screen, as shown in Figure 11.

Figure 11. The Smartphone screen with ProductForm loaded

Before you proceed to add the new form's functionality, there is one small—but critical—change that needs to be made as a result of deleting the default Form1.cs that Visual Studio generates.

5. In **Solution Explorer**, double-click **Program.cs** to open it in Code view. Visual Studio automatically generates the **Program** class and its **Main** method when you create a Windows Forms application to launch the default **Form1** that Visual Studio also generates.
6. Locate the **Main** method. There is a single line of code that passes a new instance of **Form1**, which you deleted in Step 1, to the **Application.Run** method. Change the name of the class that is instantiated from **Form1** to the name of the form you just created, **ProductForm**.
7. Verify that the **Main** method looks like the following code example.

```
static void Main()
{
    Application.Run(new ProductForm());
}
```

8. Close the Program.cs file by clicking **File** | **Close** and by clicking **Yes** to save the file if prompted.

You may now proceed to add the new form's functionality.

9. In Visual Studio, click **Data** | **Show Data Sources**. The **Data Sources** pane appears.
10. In the **Data Sources** pane, expand **OrdersDataSet**, and then expand **Inventory**, as shown in Figure 12.

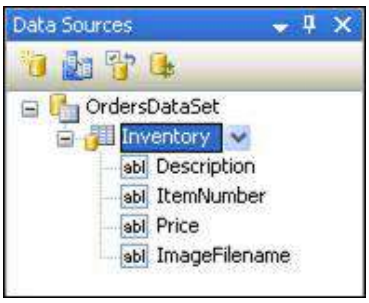

Figure 12. Data Sources pane

11. In the drop-down list box on the **Inventory** node, click **Details**, as shown in Figure 13.

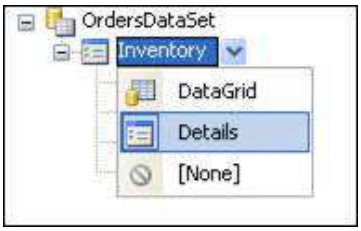

Figure 13. Inventory details

The selections in the list indicate what kind of data-binding controls should be used to create the form. Clicking **Details** indicates that each database column is displayed individually, with each having a separate label and text box, as shown in Figure 14.

Figure 14. Individual database columns

Now that you've indicated what kind of controls to use when data binding to the inventory table, you are ready to place the controls on the form.

12. Drag the **Inventory** node from the **Data Sources** pane, and drop it below the **ProductForm** caption in the form designer.

    Visual Studio creates controls for the four table columns: **Description**, **ItemNumber**, **Price**, and **Image Filename**, as shown in Figure 15.

Figure 15. Form with four table columns

Notice the Component Tray at the bottom of the form designer. Three components have been added: **ordersDataSet**, **inventoryBindingSource**, and **inventoryTableAdapter**, as shown in Figure 16.

Figure 16. The Component Tray

The **ordersDataSet** component stores data from the database table. The **inventoryBindingSource** component manages the details of data binding between the controls on the form and the **OrdersDataSet** data set. The **inventoryTableAdapter** component handles moving data between the database and the **OrdersDataSet** data set.

As you can see by looking at the form in the form designer, the limited real estate on the Smartphone requires that control placement be well thought out. To create more space, you will now reposition the controls. One way to recover screen space is to remove unnecessary labels, such as the **Description** Label.

13. To remove the **Description** Label, be sure no other controls are selected by clicking anywhere on a blank area of the form designer. Right-click the **Description** Label, and then click **Delete**.

    The focus moves to the **Description** Text Box.

14. In the **Properties** pane, resize and reposition the **Description** Text Box so its location is **62, 3** and its size is **96, 22**, as shown in Figure 17.

    > **Note** You can make these changes by manually sizing and repositioning the text box, but a more precise way is to modify the values of **Location** and **Size** in the **Properties** pane.

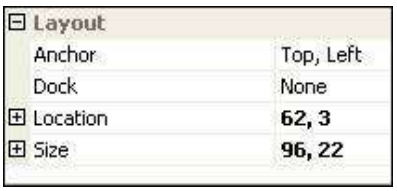

Figure 17. Resizing and repositioning the Description Text Box

15. Delete the **Item Number** Label by right-clicking it, and then by clicking **Delete**.
16. Press TAB to move the focus to the **Item Number** Text Box (or just click the Text Box).
17. Resize and reposition the **Item Number** Text Box so its location is **3, 3**; and its size is **51, 22**, as shown in Figure 18.

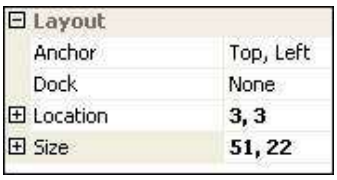

Figure 18. Resizing and repositioning the Item Number Text Box

18. Resize and reposition the **Price** Label to a location of **8, 28** and a size of **30, 15**.
19. Resize and reposition the **Price** Text Box to a location of **62, 26** and a size of **96, 22**.
20. Because you will be showing an actual image in the mobile application, there is no need to display the image file name on the form. Delete both the **Image Filename** label and its text box, **imageFilenameTextBox**.

    Now you are ready to add a PictureBox control to display the image file.

21. If the Toolbox is not visible, click **View | Toolbox**.
22. Drag the PictureBox control from the Toolbox to the form.
23. Resize and reposition the PictureBox control to a location of **20, 49** and a size of **122, 122**.
24. In the **Properties** pane, change the value of the **Name** (pictureBox1) property to **imagePictureBox**, as shown in Figure 19.

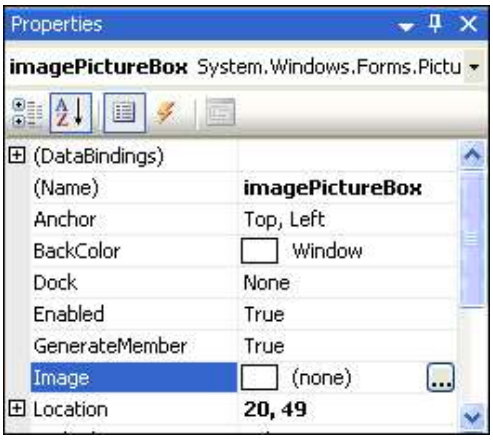

Figure 19. Changing the Name property

25. Set the **SizeMode** property to **StretchImage**.

    The **PictureBox** automatically scales the image to the size of the PictureBox control.

    You have completed the form layout. It should look something like Figure 20.

Figure 20. The revised ProductForm layout

In the following procedure, you will create the commands for the form. The commands include scrolling to the next item and closing the application.

To add menus to the product form

1. In the form designer for ProductForm.cs, click anywhere in the left half of the light-blue menu area of the emulator above the left soft key. The left soft key menu is selected, showing the text **Type Here**, as shown in Figure 21.

Figure 21. Selecting the left soft key's menu

> **Note** Another way to select the left soft key's menu is to expand the list box at the top of the **Properties** pane, and then select **mainMenu1**.

2. Type **Next**, and then press ENTER. The left soft key's menu now has the label **Next**.
3. Click in the right half of the menu area above the right soft key. The right soft key's menu is selected, showing the text **Type Here**, as shown in Figure 22.

Figure 22. The right soft key with the Type Here text

4. Type **Menu**, and then press ENTER. You are now ready to enter text for the first command on the menu, as shown in Figure 23.

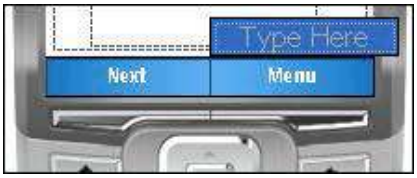

Figure 23. First command on the menu

5. Type **Exit**, and then press ENTER. The menu should now look like Figure 24.

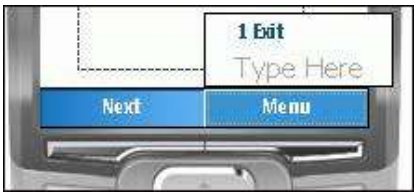

Figure 24. The ProductForm menu commands

6. Right-click **Next** (above the left soft key), and then click **Properties**.
7. In the **Properties** pane, double-click the value of **Name (menuItem1)**, type **menuNext**, and then press ENTER.
8. Click Menu (above the right soft key), right-click the Exit command, and then click Properties.
9. In the Properties pane, double-click the value of Name (menuItem3), type menuExit, and then press ENTER.

   You are done adding the initial menus and commands to the product form.

In the following procedure, you will implement the code that provides the ability to scroll through the list of available products. The data binding details are handled by the **inventoryBindingSource** component, so most of the work that is related to managing the data binding process involves the **inventoryBindingSource** component.

To add data scrolling

1. In the form designer for ProductForm.cs, double-click the **Next** menu label. This action opens the Code view for ProductForm.cs and creates a **menuNext_Click** method, as shown in Figure 25.

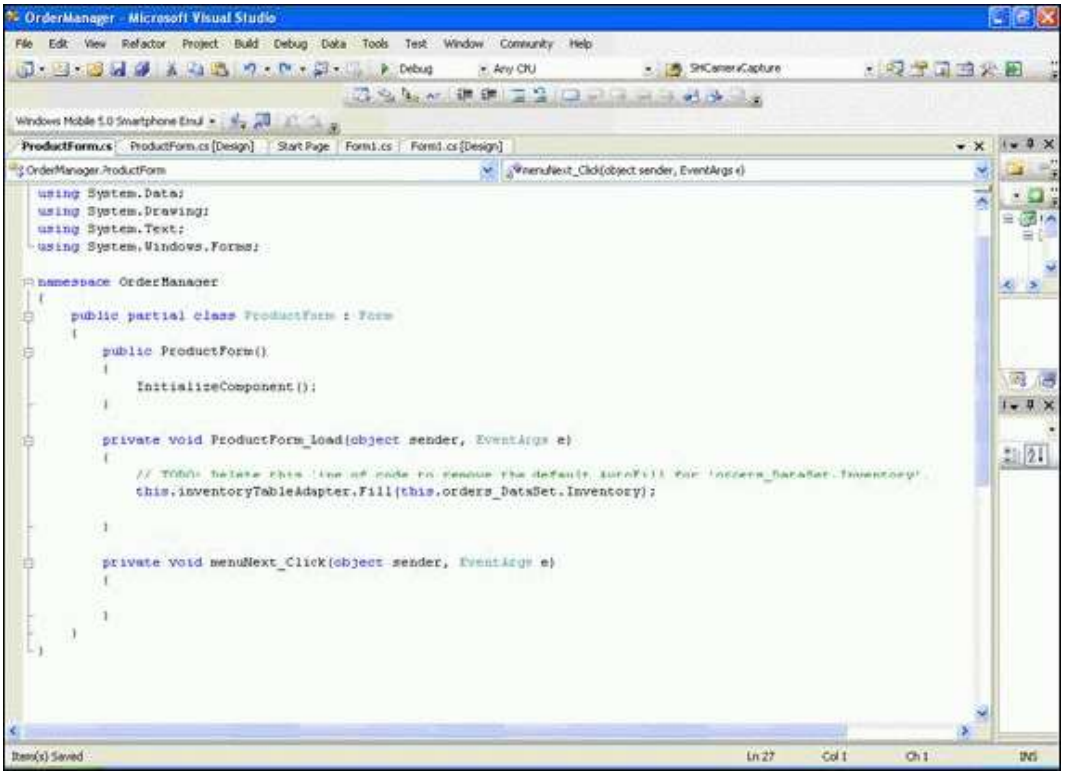

Figure 25. The menuNext_Click method in Code view of ProductForm.cs. Click the thumbnail for a larger image.

2. To scroll to the next record from the Inventory Table, you must increment the **Position** property of the **inventoryBindingSource** by one. Add the following code to the body of the **menuNext_Click** method.

Copy

```
inventoryBindingSource.Position += 1;
```

3. Verify that the **menuNext_Click** method looks like the following code example.

```
private void menuNext_Click(object sender, EventArgs e)
{
 inventoryBindingSource.Position += 1;
}
```

Incrementing the **Position** property by one is adequate to scrolling to the next item. The concern is how to make using the application easier for the user when the end of the list is reached. For ease of use, it would be nice if the application automatically looped back to the beginning.

4. Modify the line you added in Step 3. After incrementing the **Position** property by one, use the modulus operator (%) with the **Count** property, as shown in the following code example. This modification causes the **Position** property to be set to zero when incremented beyond the number of records.

```
inventoryBindingSource.Position = (inventoryBindingSource.Position + 1) % inventoryBindingSource
```

5. Verify that the **menuNext_Click** method looks like the following code example.

```
private void menuNext_Click(object sender, EventArgs e)
{
  inventoryBindingSource.Position = (inventoryBindingSource.Position + 1) % inventoryBindingSour
}
```

The application will now scroll through the data in the Inventory Table, and automatically update the text boxes with the correct data.

The next step is to display the appropriate image in the PictureBox control. Loading and displaying the image is still part of the data-binding process, but they require a few additional steps because a separate bitmap must be created for each image.

6. Open ProductForm.cs in the form designer.
7. Locate the **inventoryBindingSource** in the Component Tray, as shown in Figure 26.

Figure 26. Component Tray

8. Double-click **inventoryBindingSource**. This step opens ProductForm.cs in Code view, and it creates the **inventoryBindingSource_CurrentChanged** method.

This method is automatically called each time a different record is displayed in the form. In this application, the method is called when the **ProductForm** is first displayed, and each time the **inventoryBindingSource** Position property is modified.

9. Before you modify the **inventoryBindingSource_CurrentChanged** method, declare a class-level variable (contained in the **ProductForm** class but outside any method) named **_imageBitmap**, of type **Bitmap**, as shown in the following code example. The variable can be declared anywhere in the class. A good place is immediately after the **inventoryBindingSource_CurrentChanged** method.

```
Bitmap _imageBitmap;
```

Now you are ready to add code to the **inventoryBindingSource _CurrentChanged** method. The **inventoryBindingSource** component maintains a reference to **DataRowView** for the currently displayed row in the **Current** property.

10. Declare a **DataRowView** variable named **rowView** at the beginning of the **inventoryBindingSource _CurrentChanged** method. Assign it the value of the **inventoryBindingSource.Current** property. The **Current** property's type is **Object**, so you need to explicitly cast it, as shown in the following code example.

```
DataRowView rowView = (DataRowView) inventoryBindingSource.Current;
```

11. The **Row** property of **rowView** holds a reference to the row of the Inventory **DataTable** currently displayed. Declare an **OrdersDataSet.InventoryRow** variable named row, and assign it the value of **rowView.Row**. You need to explicitly cast the **rowView.Row** return type, as shown in the following code example.

```
OrdersDataSet.InventoryRow row = (OrdersDataSet.InventoryRow) rowView.Row;
```

If you have not worked with **typed DataSets** before, a data type name like **OrdersDataSet.InventoryRow** might be unfamiliar to you. This type name indicates that **InventoryRow** is a nested class, which means that the **InventoryRow** class declaration is in the **OrdersDataSet** class declaration. Although nested classes are not used very often, they are useful in scenarios like typed DataSets to avoid class-naming collisions between typed DataSets representing databases with common table names (because the nested class name is considered to be scoped in the outer class).

The name of the image to display is in the **ImageFileName** column of the Inventory **DataTable**, and it can be referenced by the row **ImageFilename** property. To display the image, you must first create a **Bitmap** class from the image file.

12. Create a new instance of the **Bitmap** class, and pass the file name to the constructor. Assign the newly created **Bitmap** to the **_imageBitmap** class-level variable that you declared previously, as shown in the following code example.

```
_imageBitmap = new Bitmap(row.ImageFilename);
```

Now that the **Bitmap** class is created, it can be displayed.

13. Assign the **_imageBitmap** to the **imagePictureBox.Image** property, as shown in the following code example.

```
imagePictureBox.Image = _imageBitmap;
```

You have now done the work necessary to display the appropriate image with each product.

There is one last bit of housekeeping to add. The **Bitmap** class implements the **IDisposable** interface, which means that an application that uses the **Bitmap** class should call the **Bitmap Dispose** method when the **Bitmap** class is no longer needed. Failure to do this step may result in resource leaks. In your application, the best place to dispose a **Bitmap** instance is before the next one is created.

14. At the beginning of the **inventoryBindingSource_CurrentChanged** method, add an **if** statement that checks to see if the value of **_imageBitmap** is not null, as shown in the following code example.

```
if (_imageBitmap != null)
{
}
```

15. In the if statement block, call the **_imageBitmap Dispose** method, as shown in the following code example.

```
_imageBitmap.Dispose();
```

16. Verify that the completed **inventoryBindingSource_CurrentChanged** method and **_imageBitmap** declaration look like the following code example.

```
void inventoryBindingSource_CurrentChanged(object sender, EventArgs e)
{
    if (_imageBitmap != null)
    {
        _imageBitmap.Dispose();
    }

    DataRowView rowView =  (DataRowView)inventoryBindingSource.Current;
    OrdersDataSet.InventoryRow row =
        (OrdersDataSet.InventoryRow) rowView.Row;

    _imageBitmap = new Bitmap(row.ImageFilename);
    imagePictureBox.Image = _imageBitmap;
```

```
}
Bitmap _imageBitmap;
```

Now you are ready to add the **Exit** menu handler to the **ProductForm** menu. As you might expect, the **Exit** command closes the application by closing the form that was instantiated as the main entry point of the application.

To add the Exit menu handler

1. Open **ProductForm.cs** in the form designer.
2. Click **Menu** (above the right soft key), and then double-click the **Exit** command. This step opens ProductForm.cs in Code view and creates the **menuExit_Click** method.
3. In the **menuExit_Click** method, use the form's **Close** method to close the form and exit the application, as shown in the following code example.

```
Close();
```

4. Verify that the complete implementation of the **Exit** menu handler looks like the following code example.

```
private void menuExit_Click(object sender, EventArgs e)
{
    Close();
}
```

You are now ready to test the data-browsing feature of your application.

To test your new application feature

1. In Visual Studio, click **Build** | **Build Solution**.
2. Click **Debug** | **Start Without Debugging**. The **Deploy OrderManager** dialog box appears.
3. Verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in **Device**, and then click **Deploy**, as shown in Figure 27.

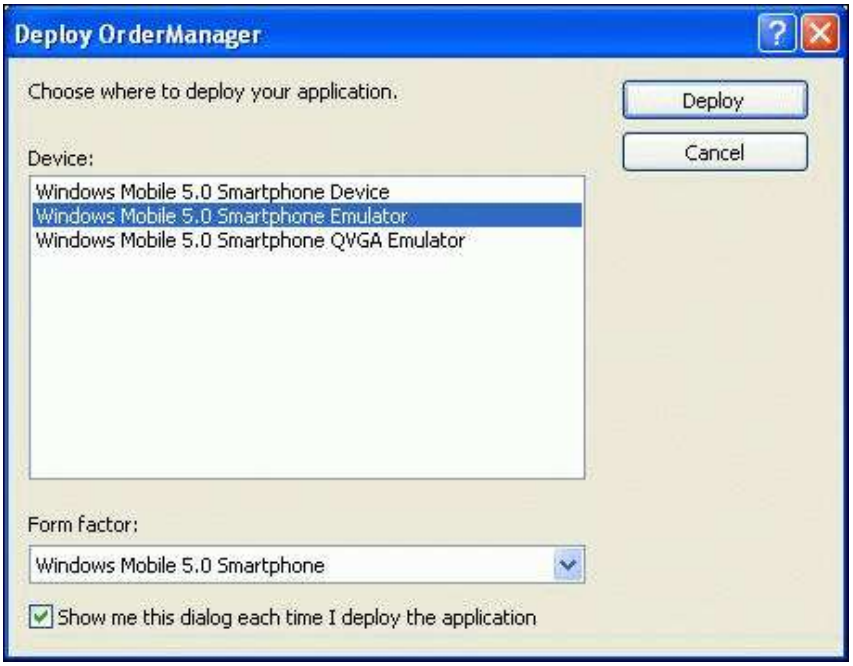

Figure 27. The Deploy OrderManager dialog box

The installation .cab files deploy to the emulated device. This step takes about three minutes. When the application is ready, it appears in the emulator, as shown in Figure 28.

Figure 28. Emulator with the application running. Click the thumbnail for a larger image.

> **Note** If the dialog box shown in Figure 29 appears, click **No**. You need to exit the version of this application that is currently running on the emulated device (for more information, see Appendix A in this HOL). After you've followed the instructions in Appendix A, click **Debug** | **Start Without Debugging** in Visual Studio.

Figure 29. Deployment error dialog box

4. When the form loads, scroll through several of the items by clicking the left soft key under the **Next** command. Notice that if you scroll past the end of the list, the application automatically loops back to the first item. (There are a dozen pictures total.)
5. When you are satisfied that the application runs properly, click the right soft key under the **Menu** command, and then use the device emulator's direction keys and its Action key (between the directional keys) to select the **Exit** command.

> **Note** You can also use the device's numeric keys that correspond to the numbers displayed next to each command. When using a device emulator, you can also use the mouse to click commands directly.

Exercise 2: Adding Support for Sending E-Mail to a Contact

In this exercise, you will add support for sending an e-mail message to a selected contact, with the selected picture as an attachment.

Rather than maintaining a separate version of contact data in the SQL Mobile database, the application takes advantage of the list of Contacts that are available in the Smartphone's Outlook Mobile Contact collection. You will use the Contact Picker, which provides an

easy way to browse contacts that are in Outlook Mobile. The Contact Picker is exposed to the .NET Compact Framework applications through the **ChooseContactDialog** method.

Before you can use the **ChooseContactDialog** class, you must add a reference for the Microsoft.WindowsMobile.Forms assembly to the project, and add a **using** declaration for the corresponding namespace to the source code.

To add assembly references and namespace declarations

1. In **Solution Explorer**, right-click the word **References**, and then select **Add Reference**.
2. In the **Add Reference** dialog box, press CTRL while you select **Microsoft.WindowsMobile.Forms** and **Microsoft.WindowsMobile.PocketOutlook**, as shown in Figure 30.

   > **Note** There are several assemblies that are visible and that contain the word Forms in the **Add Reference** dialog box. Be careful to choose the Microsoft.WindowsMobile.Forms assembly.

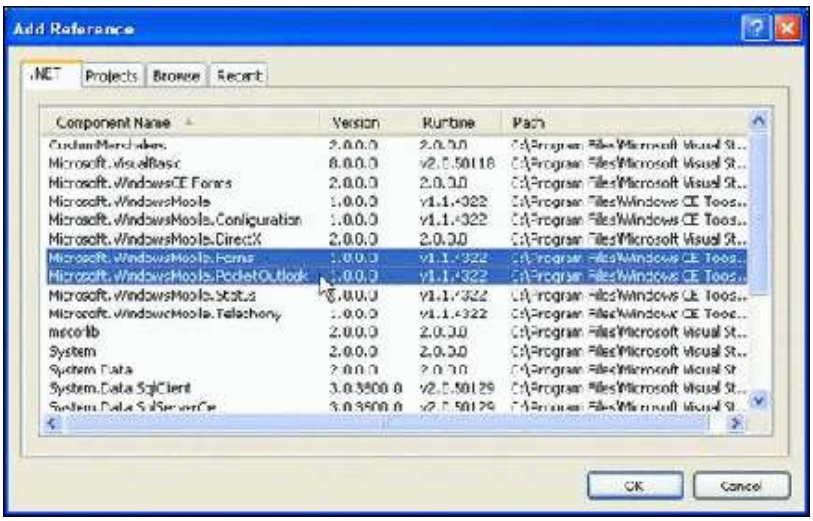

Figure 30. Add Reference dialog box. Click the thumbnail for a larger image.

3. Click **OK**.

   The references appear in the list of project references in **Solution Explorer**, as shown in Figure 31.

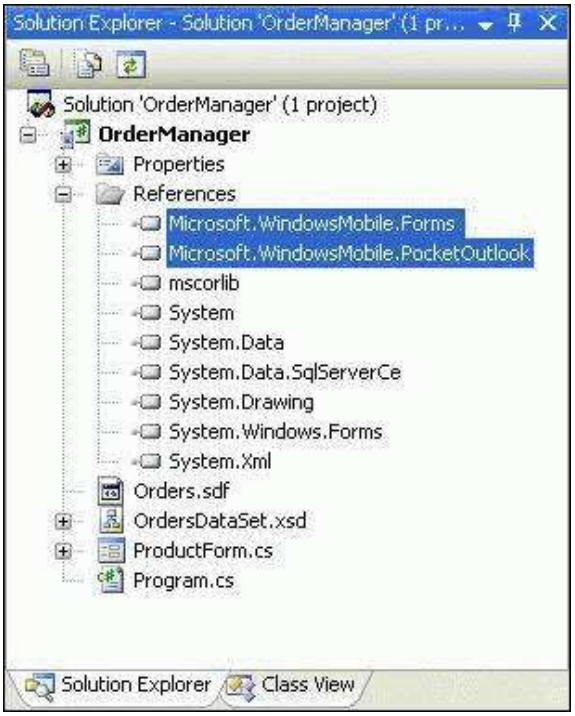

Figure 31. Project references in Solution Explorer

4. If **ProductForm.cs** is not already open in Code view, open it now.

5. Locate the existing using declarations at the top of the **ProductForm.cs** file.
6. Add **using** declarations for Microsoft.WindowsMobile.Forms and Microsoft.WindowMobile.PocketOutlook immediately following the existing **using** declarations, as shown in the following code example.

```
using Microsoft.WindowsMobile.Forms;
using Microsoft.WindowsMobile.PocketOutlook;
```

Now you are ready to add the **Send** command and its handler to the **ProductForm** menu. The **Send** command prompts the user for a contact and then sends an e-mail message to that contact. The e-mail message will contain the selected picture as an attachment. To add these features, you will use the POOM **ChooseContactDialog** and **EmailMessage** classes.

To display the Contact Picker and send an e-mail message

1. Open **ProductForm.cs** in the form designer.
2. Click the **Menu** command (above the right soft key), and then click the text **Type Here**.
3. Type **Send**, and then press ENTER. The menus should look like Figure 32.

Figure 32. The ProductForm menus

4. Right-click the **Send** command, and then click **Properties**.
5. In the **Properties** pane, double-click the value of **Name (menuItem1)**, type **menuSend**, and then press ENTER.
6. Double-click the **Send** command. This step opens ProductForm.cs in Code view and creates the **menuSend_Click** method.
7. In the body of the **menuSend_Click** method, declare and create an instance of **ChooseContactDialog**, and name the variable **contactPicker**, as shown in the following code example.

```
ChooseContactDialog contactPicker = new ChooseContactDialog();
```

8. Display the dialog box by calling **ShowDialog**, checking the **ShowDialog** method's return value to determine whether or not the user chose to continue, as shown in the following code example.

```
if (contactPicker.ShowDialog() == DialogResult.OK)
{
}
```

9. Inside the **if** condition, declare a variable named **rowView** with a type of **DataRowView** and assign it the value of the **inventoryBindingSource.Current** property. Remember to cast the return type, as shown in the following code example.

```
DataRowView rowView =
    (DataRowView)inventoryBindingSource.Current;
```

10. Declare a variable named **selectedProduct** with a type of **OrdersDataSet.InventoryRow**, and assign it the value of the **rowView.Row** property. Remember to cast the return type, as shown in the following code example.

```
OrdersDataSet.InventoryRow selectedProduct =
    (OrdersDataSet.InventoryRow)rowView.Row;
```

11. Declare and create an instance of the **EmailMessage** class, and then name the variable **message**, as shown in the following code example.

```
EmailMessage message = new EmailMessage();
```

12. Assign the text **The picture you requested** to the **EmailMessage Subject** property, and the text **Attached is the picture we discussed** to the **BodyText** property, as shown in the following code example.

```
message.Subject = "The picture you requested";
message.BodyText = "Attached is the picture we discussed";
```

13. Add the e-mail address for the selected contact to the **EmailMessageTo** property. To do this action, create an instance of the **Recipient** class, and then pass the **contactPicker.SelectedContact Email1Address** property to the constructor. Then, add the newly created **Recipient** instance to the **EmailMessageTo** property. Name the **Recipient** variable **client**, as shown in the following code example.

```
Recipient client =
    new Recipient(contactPicker.SelectedContact.Email1Address);
message.To.Add(client);
```

14. Attach the image file to the e-mail message. To do this action, create an instance of the **Attachment** class, and then pass the **selectedProduct ImageFilename** property to the constructor. Add the newly created **Attachment** to the **EmailMessage Attachment** property. Name the **Attachment** variable **image**, as shown in the following code example.

```
Attachment image = new Attachment(selectedProduct.ImageFilename);
message.Attachments.Add(image);
```

Now you are ready to send the e-mail message.

15. To send the e-mail message, call the **message.Send** method, and then pass the name of the e-mail account to send from. Use **ActiveSync** as the account name, as shown in the following code example.

```
message.Send("ActiveSync");
```

16. Verify that the complete **menuSend_Click** method looks like the following code example.

```
private void menuSend_Click(object sender, EventArgs e)
{
    ChooseContactDialog contactPicker = new ChooseContactDialog();

    if (contactPicker.ShowDialog() == DialogResult.OK)
    {
        DataRowView rowView =
            (DataRowView)inventoryBindingSource.Current;
        OrdersDataSet.InventoryRow selectedProduct =
            (OrdersDataSet.InventoryRow)rowView.Row;

        EmailMessage message = new EmailMessage();
        message.Subject = "The picture you requested";
        message.BodyText = "Attached is the picture we discussed";

        Recipient client =
            new Recipient(contactPicker.SelectedContact.Email1Address);
        message.To.Add(client);

        Attachment image = new Attachment(selectedProduct.ImageFilename);
        message.Attachments.Add(image);

        message.Send("ActiveSync");
    }
}
```

In the following procedure, you will test the updated application.

To test the updated application

1. In Visual Studio, click **Build** | **Build Solution**. Correct any compilation errors.
2. Click **Debug** | **Start Without Debugging**.
3. Verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in **Device**, and then click **Deploy**.
4. When the application starts, click the right soft key (under **Menu**), and then choose **Send**, as shown in Figure 33.

Figure 33. Choosing the Send command in the updated application. Click the thumbnail for a larger image.

The Contact Picker appears, as shown in Figure 34.

Figure 34. The Contact Picker. Click the thumbnail for a larger image.

5. Select the contact, **Adams, Jay**, by clicking the left soft key. The Contact Picker closes, and the message is sent.
6. Close the application by clicking the right soft key, and then by choosing **Exit**.

You can verify that the message was actually submitted to the e-mail system by checking the Messaging application's Outbox folder.

To verify that the message was sent

1. Open the **Home** screen by clicking the **Home** button (the button with a picture of a house, immediately below the left soft key).
2. Click the left soft key to select **Start**. Browse to and select the **Messaging** icon by clicking the Action key in the middle of the navigational pad.
3. Click **Outlook E-mail** if it is not already highlighted, and then click the left soft key under **Select**, as shown in Figure 35.

Figure 35. Selecting Outlook E-mail. Click the thumbnail for a larger image.

4. Click the right soft key under **Menu**.
5. Click **Folders**.
6. Highlight **Outbox**, and then click the left soft key under **Select**. You should see a message with the subject "The picture you requested" addressed to jadams@wingtiptoys.com, as shown in Figure 36.

Figure 36. Outbox with sent mail confirmation. Click the thumbnail for a larger image.

7. Click the Action key in the navigational pad to open the message.

   You should see the message details with the body "Attached is the picture we discussed" and the file beautiful.jpg attached, as shown in Figure 37.

Figure 37. The contents of the e-mail message that was sent. Click the thumbnail for a larger image.

8. You can return to the **Home** screen by clicking the **Home** button.

Exercise 3: Accessing Web Services and Retrieving Remote Image Files

In this exercise, you will add support for refreshing the SQL Mobile database with product data from a Web service and downloading additional image files from a Web server. You will see that it is very simple to call Web services from mobile applications, that you can update SQL Mobile data the same way that you update standard SQL Server data, and that downloading files from a Web server to a device is straightforward.

In the following procedure, you will add a reference to and call a Web service that retrieves data that is used to refresh the SQL Mobile database.

To reference the Web service

1. In **Solution Explorer**, right-click the **OrderManager** project, and then click **Add Web Reference**. The **Add Web Reference** dialog box appears, as shown in Figure 38.

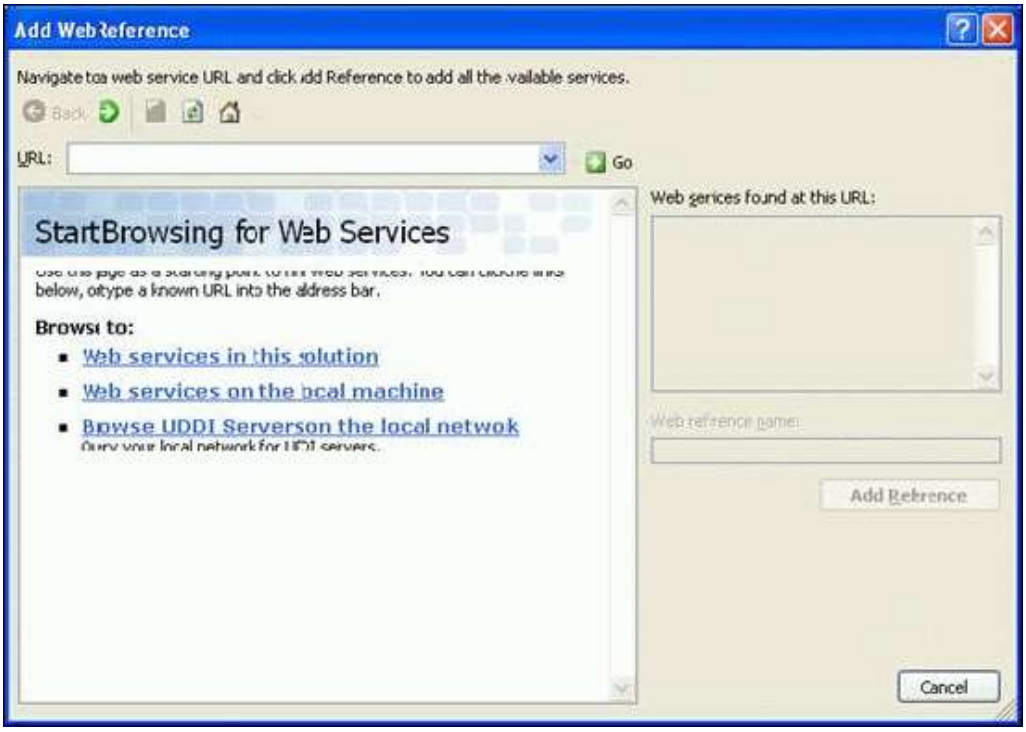

Figure 38. Browsing for Web services by using the Add Web Reference dialog box. Click the thumbnail for a larger image.

2. In the **URL** box, type **http:**//*your computer's IP address*/**ProductData/UpdateService.asmx**, and then click **Go**. Be sure to replace *your computer's IP address* with the actual IP address of your computer. Be certain not to use localhost in place of your computer's IP address; the Web service will be called from the device emulator, so the actual network IP address must be used.

   > **Note** To determine your computer's IP address, do the following: On your desktop computer, click **Start** | **Run**. On the Command Prompt window, type **cmd**, and then click **OK**. Type **ipconfig**, and then press ENTER. Your computer's IP Address appears with other information. Close the Command Prompt window after you've recorded the IP address.

3. After the UpdateService service detail appears in the Web services browser, type **ProductDataServices** as the **Web reference name**, and then click **Add Reference**, as shown in Figure 39.

Figure 39. Adding the ProductData Web service reference. Click the thumbnail for a larger image.

The Web reference appears in the list of project Web references in **Solution Explorer**, as shown in Figure 40.

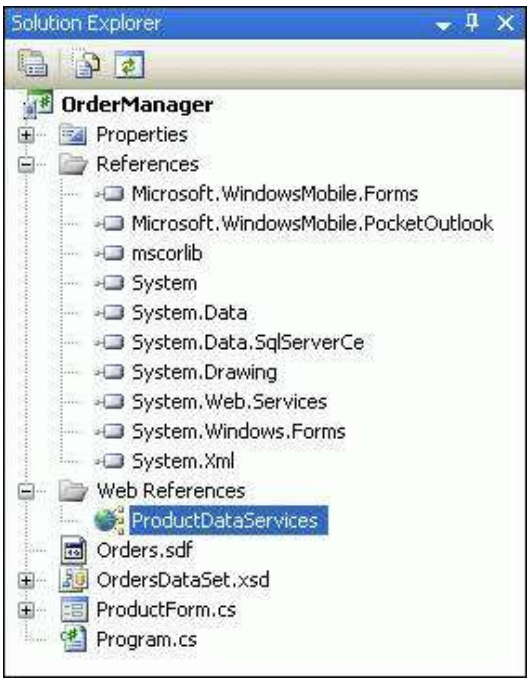

Figure 40. Web references in Solution Explorer

Now, you need to add a command to initiate Web service data retrieval.

To add a command that calls the Web service

1. Open **ProductForm.cs** in the form designer.
2. Click **Menu** (above the right soft key), and then click the text **Type Here**.
3. Type **Refresh from Web**, and then press ENTER. The menus should look like Figure 41.

Figure 41. The ProductForm commands

4. Right-click the **Refresh from Web** command, and then click **Properties**.
5. In the **Properties** pane, double-click the value of **Name (menuItem1)**, type **menuRefresh**, and then press ENTER.
6. Double-click the **Refresh from Web** command. This step opens ProductForm.cs in Code view, and creates the **menuRefresh_Click** method.
7. In the body of the **menuRefresh_Click** method, indicate to the user that this operation may take some time by setting the current cursor to the **Cursors.WaitCursor** enumeration value. Because the application's current thread will be busy performing the actual work to call the Web service, to force the device's user interface to show the wait cursor, call **Application.DoEvents** twice, as shown in the following code example.

Copy
```
Cursor.Current = Cursors.WaitCursor;
Application.DoEvents();
Application.DoEvents();
```

> **Note** The **Application.DoEvents** method is used in this lab for simplicity. In general, using asynchronous Web service calls or multithreading is preferable to using the Application.DoEvents methods.

8. Declare and create an instance of the **ProductDataServices.UpdateService** Web service class named **updateService**, as shown in the following code example.

Copy
```
ProductDataServices.UpdateService updateService =
    new ProductDataServices.UpdateService();
```

9. Declare and create an instance of the **ProductDataServices.OrdersDataSet** class named **newOrders** by calling the **GetInventory** method of the **updateService** Web service, as shown in the following code example.

```
ProductDataServices.OrdersDataSet newOrders =
    updateService.GetInventory();
```

The **GetInventory** Web service method is hard-coded to return the same two additional product records each time it is called. An actual production Web service could be written to query a server-side database of products and return a more dynamic set of records. The records are returned as a **ProductDataServices.OrdersDataSet** data set, which has a schema that matches the **OrdersDataSet** data set that Visual Studio created in this application when you added the SQL Mobile data source.

10. Merge the data that has been returned from the Web service with the data that is already bound to the form by calling the **ordersDataSet.Merge** method, passing it the **newOrders** data set, as shown in the following code example.

```
ordersDataSet.Merge(newOrders);
```

11. Update the device's SQL Mobile database by calling the **inventoryTableAdapter.Update** method, passing it the newly merged **ordersDataSet** data set, as shown in the following code example.

```
inventoryTableAdapter.Update(ordersDataSet);
```

12. Reset the cursor now that the data has been refreshed, as shown in the following code example.

```
Cursor.Current = Cursors.Default;
```

13. Verify that the current implementation of the **menuRefresh_Click** menu handler looks like the following code example.

```
private void menuRefresh_Click(object sender, EventArgs e)
{
    Cursor.Current = Cursors.WaitCursor;
    Application.DoEvents();
    Application.DoEvents();
    ProductDataServices.UpdateService updateService =
        new ProductDataServices.UpdateService();
    ProductDataServices.OrdersDataSet newOrders =
        updateService.GetInventory();
    ordersDataSet.Merge(newOrders);
    inventoryTableAdapter.Update(ordersDataSet);
    Cursor.Current = Cursors.Default;
}
```

In Exercise 1, you added code to the **inventoryBindingSource_CurrentChanged** event handler that attempts to read an image file for each product. To prevent this code from malfunctioning when displaying the new records, you need to be sure that any new image files are also available on the device. The ProductData virtual directory not only serves as a Web service for product data retrieval, but also as a Web server that hosts product image files. Rather than retrieving updated image files via the Web service, you will use HTTP methods to simply download the image files from the Web server.

To add support for downloading additional image files from the Web server

1. Add using declarations for **System.Net** and **System.IO**, as shown in the following code examples, immediately following the existing using declarations in ProductForm.cs.

```
using System.Net;
using System.IO;
```

To save time typing, you will use a code example to insert the following code.

2. Place the cursor inside the **ProductForm** class, but outside any other methods.
3. On the Visual Studio 2005 menu, click **Edit** | **IntelliSense** | **Insert Snippet**. Double-click **My Code Snippets**, and then double-click **SnippetDownloadFileBinary**. Visual Studio inserts the following declaration for a new method, **DownloadFileBinary**, into

the class, as shown in the following code example. This method uses the standard HTTP request **GET** method to copy a binary file from a Web server to the local file system.

Copy

```
private void DownloadFileBinary(string localFile, string downloadUrl)
{
    HttpWebRequest req =
        (HttpWebRequest)WebRequest.Create(downloadUrl);
    req.Method = "GET";
    HttpWebResponse resp = (HttpWebResponse)req.GetResponse();
    // Retrieve response stream
    Stream respStream = resp.GetResponseStream();
    // Create a local file
    FileStream wrtr = new FileStream(localFile, FileMode.Create);
    // Allocate a byte buffer to hold stream contents
    byte[] inData = new byte[4096];
    // Loop through the response stream reading each data block
    //  and writing to the local file
    int bytesRead = respStream.Read(inData, 0, inData.Length);
    while (bytesRead > 0)
    {
        wrtr.Write(inData, 0, bytesRead);
        bytesRead = respStream.Read(inData, 0, inData.Length);
    }
    respStream.Close();
    wrtr.Close();
}
```

4. Add a constant to the **ProductForm** class that contains the URL to the Web server that stores product images, as shown in the following code example. Be sure to replace your computer's IP address in the following code example with your computer's actual IP address. A good place for this constant is inside the class declaration, immediately before the **ProductForm** constructor.

Copy

```
private const string _imageServerUrl = "http://your computer's IP address>/ProductData/";
```

Finally, you need to add code to loop through any new product records received from the Web service and copy any new image files from the Web server to the device.

5. In the **menuRefresh_Click** menu handler, after retrieving the updated data with the call to **updateService.GetInventory**, but before merging it with the data binding source with the call to **ordersDataSet.Merge**, add a **foreach** loop that iterates through the **InventoryRow** rows contained in the Inventory table in the **newOrders** data set, as shown in the following code ex

Copy

```
foreach (ProductDataServices.OrdersDataSet.InventoryRow row in
    newOrders.Inventory.Rows)
{
}
```

6. Inside the **foreach** loop, assign the **row.ImageFileName** property value to a string variable named **imageFileName**, as shown in the following code example.

Copy

```
string imageFileName = row.ImageFilename;
```

7. Call the **DownloadFileBinary** method, passing it the **imageFileName** value as the local file name and the **_imageServerUrl** constant value concatenated with the **imageFileName** value as the download URL, as shown in the following code exar

Copy

```
DownloadFileBinary(
    imageFileName, _imageServerUrl + imageFileName);
```

8. Verify that the complete implementation of the **menuRefresh_Click** menu handler looks like the following code exampl

Copy

```
private void menuRefresh_Click(object sender, EventArgs e)
{
    Cursor.Current = Cursors.WaitCursor;
    Application.DoEvents();
    Application.DoEvents();
```

```
        ProductDataServices.UpdateService updateService =
            new ProductDataServices.UpdateService();
        ProductDataServices.OrdersDataSet newOrders =
            updateService.GetInventory();
        foreach (ProductDataServices.OrdersDataSet.InventoryRow row in
            newOrders.Inventory.Rows)
        {
            string imageFileName = row.ImageFilename;
            DownloadFileBinary(
                imageFileName, _imageServerUrl + imageFileName);
        }
        ordersDataSet.Merge(newOrders);
        inventoryTableAdapter.Update(ordersDataSet);
        Cursor.Current = Cursors.Default;
    }
```

To enable the device emulator to call a Web service running on your local computer, you need to provide the emulator with network connectivity. The easiest way to provide network connectivity is to use ActiveSync 4.0 or later to connect to the emulator. This is a newly provided feature of Visual Studio 2005 and ActiveSync 4.0.

To establish connectivity between the emulator and the desktop computer

1. Click **Start** | **All Programs** | **Microsoft ActiveSync**.
2. In ActiveSync, click **File** | **Connection Settings**.
3. In the **Connection Settings** dialog box, select **Allow connections to one of the following** if it is not already selected, and then select DMA in the corresponding list.
4. Click **OK**. ActiveSync is now able to connect to the emulators as well as physical devices.

   The next step is to connect and cradle the emulator by using the new Visual Studio 2005 Device Emulator Manager.

5. In Visual Studio, click **Tools** | **Device Emulator Manager**.
6. In the Device Emulator Manager window, select **Windows Mobile 5.0 Smartphone Emulator**. You may have to expand the **Windows Mobile 5.0 Smartphone SDK** node to locate **Windows Mobile 5.0 Smartphone Emulator**.
7. If there is not already a green arrow next to **Windows Mobile 5.0 Smartphone Emulator**, click **Actions** | **Connect** on the Device Emulator Manager window. A green arrow should appear next to **Windows Mobile 5.0 Smartphone Emulator**.
8. With Windows Mobile 5.0 Smartphone Emulator still selected, click **Actions** | **Cradle**. ActiveSync should begin the connection process, which may take a minute or two.
9. If the **Microsoft Office Outlook** dialog box appears, click **OK**.
10. If the **Microsoft ActiveSync** dialog box appears, click **OK**.
11. On the **Synchronizing Setup Wizard** page, click **Cancel**. Clicking **Cancel** establishes connectivity between the emulator and desktop computer without requiring that ActiveSync be configured to keep files and Outlook information on the emulator and the desktop computer synchronized.

In the following procedure, you will test the entire application.

To test the complete application

1. In Visual Studio, click **Build** | **Build Solution**. Correct any compilation errors.
2. Click **Debug** | **Start Without Debugging**.
3. Verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in **Device**, and then click **Deploy**.

   > **Note** If the dialog box shown in Figure 29 appears, click **No**. You need to exit the version of this application that is currently running on the emulated device (for more information, see Appendix A in this HOL). After you've followed the instructions in Appendix A, click **Debug** | **Start Without Debugging** in Visual Studio.

4. When the application starts, use the **Next** command (the left soft key) to browse through all of the records. Notice that if you scroll past the end of the list, the application automatically loops back to the first item. (There are a dozen pictures total.)
5. Click the right soft key, and then select **Refresh from Web**, as shown in Figure 42.

Figure 42. Choosing the Refresh from Web command. Click the thumbnail for a larger image.

> **Note** Because the device emulator is accessing a Web service on your computer, it may take some time for the action to complete.

6. Notice that the original product with an **ItemNumber** of **1** has been replaced by a new product and a picture, as shown in Figure 43.

Figure 43. The first product has been replaced with a new product and picture. Click the thumbnail for a larger image.

The **DataSet.Merge** method found a primary key value (**ItemNumber 1**) in the new data that already existed in the original data. It replaced the old values, including the image file name, with new ones. As you can see from the picture, the new image was also downloaded from the Web server to the device.

7. Click **Next** (the left soft key) to scroll through the remaining records until you get to the other new product record (**ItemNumber 13**), as shown in Figure 44. As you browse, notice that the other original products (2 through 12) remain unchanged.

Figure 44. A completely new product. Click the thumbnail for a larger image.

The **DataSet.Merge** method has added a final record with a primary key value (**ItemNumber 13**) that did not exist in the original data set.

8. Close the application by clicking the left soft key, and then choosing **Exit**.

You have successfully added the ability to refresh the device's product data with data retrieved from a Web service. In addition to the refreshed data, you added the ability to download additional image files from the Web server.

Summary

In this lab, you performed the following exercises:

- Creating a Smartphone application and using data from SQL Mobile
- Adding support for sending e-mail to a contact
- Accessing Web services and retrieving remote image files

In this lab, you used your existing Visual Studio and .NET Framework skills to create a Windows Mobile 5.0–based Smartphone application. You tested the application with the Windows Mobile 5.0 Smartphone emulator and accessed data stored in a SQL Mobile database. You displayed an image file and scaled it to fit in the available screen space. Using POOM, you sent a contact an e-mail message with an attachment. You finished by adding the ability to call a Web service that provides updated data to the SQL Mobile database and downloads additional image files for the new data.

Appendix A: Terminating an Application That Is Running on a Device or Emulator

This appendix describes how to terminate an application that is running on a device or emulator. This is useful for cases where an application has been started without having the debugger attached and the application needs to be terminated so a new copy of the application can be deployed. You will terminate the application by using the Remote Process Viewer remote tool in Visual Studio.

Before you can terminate a process, you need to know the name of the executable file. In most cases, this is the same name as the Visual Studio project. If you are uncertain about the name of the executable file, you can find it in the project's properties.

To terminate an application that is running on a device or emulator by using the Remote Process Viewer remote tool

1. In Visual Studio, click **Project** |*xxx* **Properties**, where *xxx* represents the name of the current project.
2. In the **Project Properties** dialog box, note the value in the **Assembly Name** field. This is the name that the executable file will be running on the device or emulator.
3. Close the **Properties** dialog box.

   Now, you can terminate the process.

4. On the desktop computer, click **Start** | **Microsoft Visual Studio 2005** | **Visual Studio Remote Tools** | **Remote Process Viewer**.
5. When prompted by the **Select a Windows CE Device** dialog box, select the emulator or device where the application is running, as shown in Figure 45, and then click **OK**.

Figure 45. Select a Windows CE Device dialog box

6. After the connection to the emulator or device completes, select the application that you want to terminate in the top pane of the **Remote Process Viewer**, as shown in Figure 46.

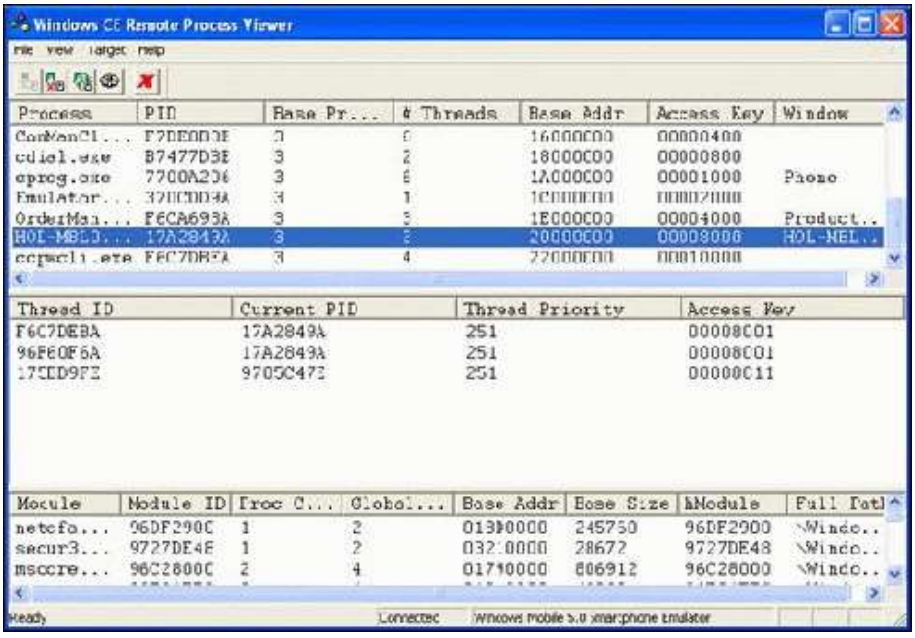

Figure 46. Selecting the application to terminate. Click the thumbnail for a larger image.

You may need to widen the **Process** column (the leftmost column) to fully view the process names.

7. In Windows CE Remote Process Viewer, click **File** | **Terminate Process** to terminate the process.

   Note Be certain that the correct process is selected before you click **Terminate Process**. Terminating the incorrect process may render the device or emulator unusable, requiring it to be reset before you can use it again.

8. Verify that the process is terminated by selecting **Target** | **Refresh**. Scroll to the top pane of Windows CE Remote Process Viewer again. If the application name still appears, the process was not terminated, and you need to repeat these steps.

   Note Most processes terminate in one try; however, depending on the state of the application, terminating a process occasionally takes two attempts.

Appendix B: Setting Up Your Computer

Before you start this HOL, you need to install Internet Information Services (IIS), as described in the HOL Requirements section.

To install Internet Information Services

1. Click **Start** | **Control Panel** | **Add or Remove Programs**.
2. On the Add or Remove Programs window, click **Add/Remove Windows Components**.
3. In the **Windows Components** list, select **Internet Information Services (IIS)** if it is not already complete.
4. Continue clicking **Next** until the Windows Components wizard is complete.
5. Close the Add or Remove Programs window.

Also, before you start this HOL, you need to run an application to add the necessary Pocket Outlook contacts and image files to the emulator.

To populate the emulator with test data

1. Download and install MEDC06_HOL201.msi.

   Note If you used the Windows Mobile 5.0 Smartphone emulator in a previous HOL, you should do a hard reset on the emulator before starting this HOL. (On the emulator, click **File** | **Reset** | **Hard**.)

   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 any time in the HOL that you deploy to the emulator. See Appendix A in this HOL for instructions about exiting a running application.

2. If Visual Studio 2005 is not already open, open it by clicking **Start** | **All Programs** | **Microsoft Visual Studio 2005** | **Microsoft Visual Studio 2005**.
3. In Visual Studio 2005, click **File** | **Open** | **Project/Solution**.
4. In the **Open Project** dialog box, browse to C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL201\Setup Files.
5. Select **InitializeLab.sln**.
6. Click **Open**. The InitializeLab solution opens.
7. In the Configuration Manager, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected as the targeted device, as shown in Figure 47.

Figure 47. Emulator selection

8. Click **Debug** | **Start Debugging** to start the application.

9. If prompted by the **Deploy InitializeLab** dialog box, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected, and then click **Deploy**.

    Note If the emulator does not appear, look for a Windows taskbar button, and click it to bring the emulator to the foreground. Be aware that the first time the emulator starts, it may take several minutes.

    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 exited before a new copy can be deployed and run. This error may appear any time in the HOL that you deploy the emulator. See Appendix A in this HOL for instructions about exiting a running application.

10. When the application appears on the emulator, click the left soft key, which is the grey button located under the word **Start**, as shown in Figure 48. The application displays a wait cursor and indicates that contacts and then images are being added.

Figure 48. Initializing the device for the HOL

11. When the application displays **Initialization Complete**, click the right soft key, which is the grey button located under the word **Exit** to close the application.
12. In the Visual Studio, click **File | Close Solution**.
13. If Visual Studio prompts you to save, click **Yes**.

    Note If you close the emulator, be sure to save the emulator state.

Also, before you start this HOL, you need to set up an IIS virtual directory that serves as a Web service and image file server.

To set up the virtual directory

1. On the desktop computer, click **Start | Run**.
2. Type **%SystemRoot%\system32\inetsrv\iis.msc**, and then click **OK** to open the Internet Information Services window.
3. On the Internet Information Services window, expand the node with the name of your computer, and then expand **Web Sites**.
4. Right-click **Default Web Site**. On the shortcut menu, click **New | Virtual Directory**, as shown in Figure 49.

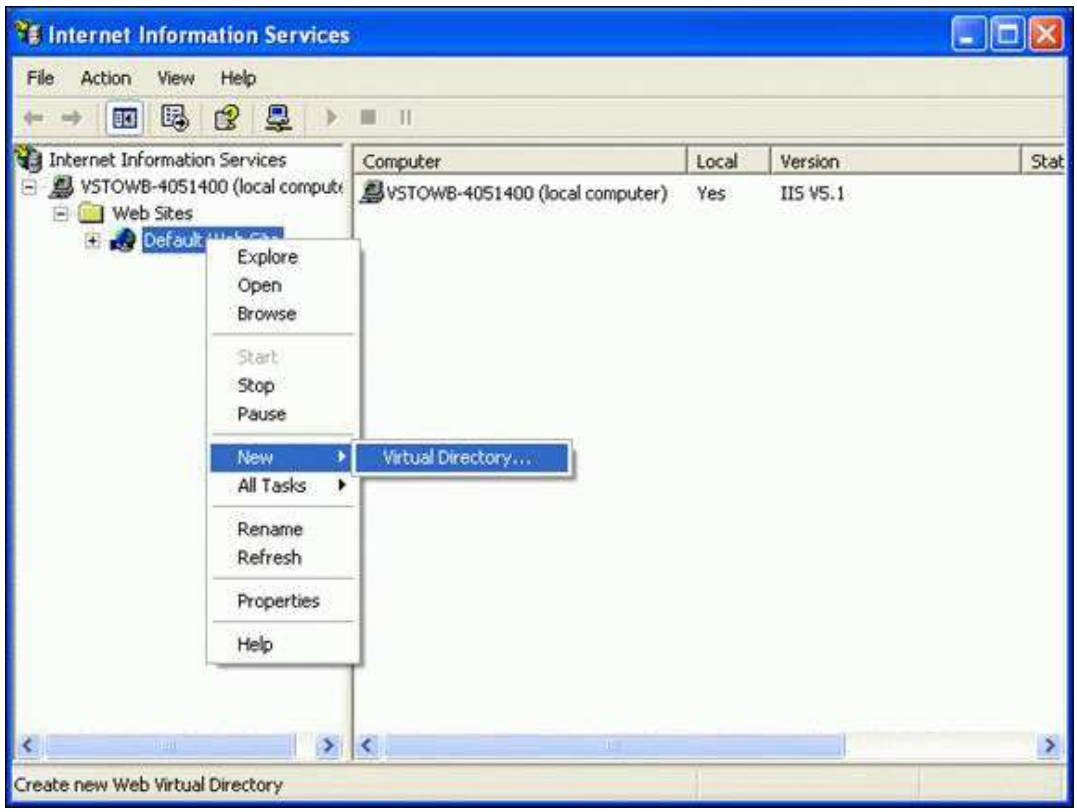

Figure 49. Creating a new virtual directory

5. On the **Virtual Directory Creation Wizard** welcome page, click **Next**.
6. In the **Alias** box, type **ProductData**, and then click **Next**, as shown in Figure 50.

Figure 50. Creating the ProductData virtual directory

7. In the **Directory** box, type **C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL201\ProductData**, and then click **Next**, as shown in Figure 51.

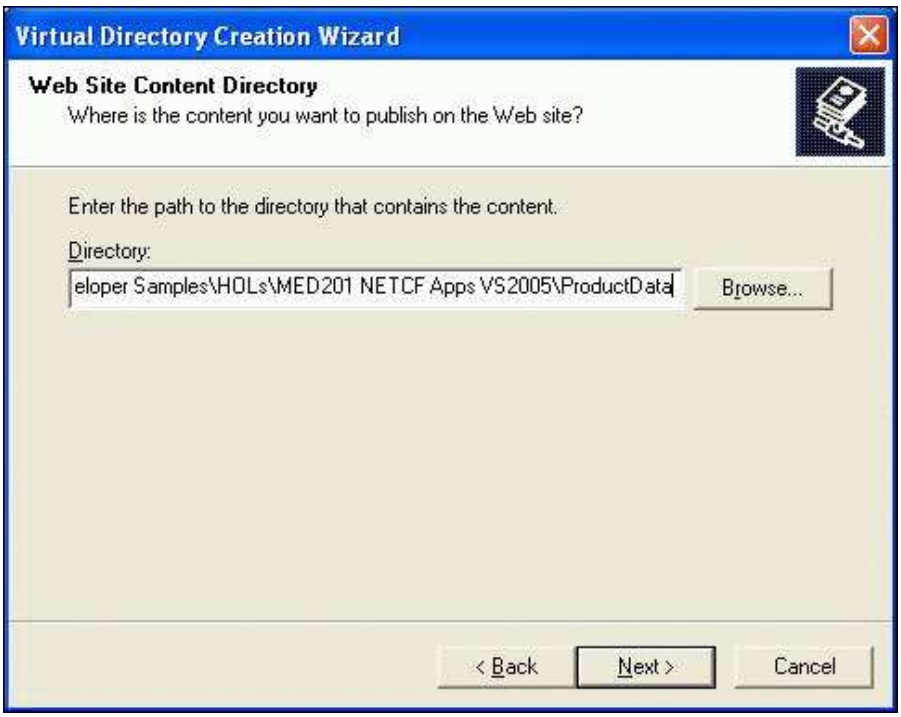

Figure 51. Setting the virtual directory Web site content directory

8. On the **Access Permissions** page of the wizard, leave the items of **Read** and **Run scripts (such as ASP)** selected. Do not select the other items. Click **Next**, as shown in Figure 52.

Figure 52. Setting the virtual directory's access permissions

9. Click **Finish**.
10. Right-click the new **ProductData** virtual directory, and then click **Properties** on the shortcut menu.
11. Click the **ASP.NET** tab.
12. Verify that the **ASP.NET version** that is selected is **2.0.50727** or later, or select it if this option is not already, as shown in Figure 53.

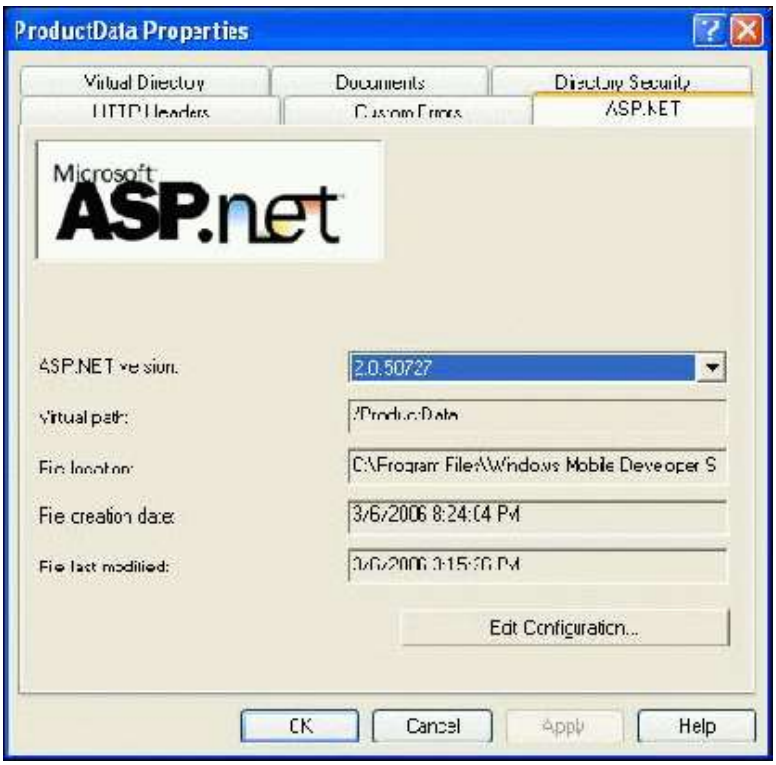

Figure 53. Configuring the virtual directory for ASP.NET 2.0. Click the thumbnail for a larger image.

13. Click **OK**.
14. Close the Internet Information Services window.

This HOL uses code examples to help save time when typing certain code. Therefore, you need to import the provided code examples before you start the HOL.

To import the code examples

1. In Visual Studio, click **Tools** | **Code Snippets Manager** to open the Code Snippets Manager.
2. On the **Code Snippets Manager** dialog box, click **Import**.
3. In the **Code Snippets Directory** dialog box, browse to C:\Program Files\Windows Mobile Developer Samples\HOLs\MEDC06_HOL201\Setup Files\.
4. Select the **WindowsMobileDemo.snippet** file, and then click **Open**.
5. On the **Import Code Snippet** dialog box, select **My Code Snippets**, and then click **Finish**.
6. Click **OK** to close the **Code Snippets Manager** dialog box.

© 2010 Microsoft Corporation. All rights reserved. Terms of Use | Trademarks | Privacy Statement | Feedback