Hands-On Lab (MBL02)

Lab Manual

Introduction to the New Managed APIs in Windows Mobile

Please do not remove this manual from the lab.
The lab manual will be available from CommNet.

HOL101 (MBL02): Introduction to the New Managed APIs in Windows Mobile

This lab introduces you to the new managed APIs in Windows Mobile. Improve on an existing application by making use of these new interfaces. Upon completion of this lab, you will be ready to use these APIs to simplify your existing code or extend your applications to make use of Windows Mobile's many new capabilities.

Intended Audience: Application developer

Level 200

Estimated Completion Time: 90 minutes

Usernames/passwords used in this lab

Location	Username	Password
<none>	<none>	<none>

Contents

Lab 1: Introduction

This lab introduces many of the newly available Microsoft® Windows Mobile™–managed APIs such as Telephony, Messaging, Microsoft Pocket Outlook®, Picture Picker, Contact Picker, and the State and Notification Broker API. To understand these APIs and how they might fit into an application, you will be updating an existing .NET Compact Framework application replacing legacy features with more appropriate features provided by the new APIs.

You will then add several new features to the application that more closely integrates the application with the phone capabilities. These new features improve the productivity of the user by automating common tasks.

The application that you will be updating is an existing Smartphone application used by field sales representatives from a high-end art and photography dealer. At a high-end 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 not in the office.

The existing application is a simple contact manager keeping track of contact names, work phone number, mobile phone number, and notes about that client. The application uses a custom data format to track client information because it does not take advantage of the available Pocket Outlook features. It also provides no integration with any phone features. Actions such as dialing phone numbers or sending e-mail messages must all be manually performed by the user.

In this lab, you will integrate the application with Pocket Outlook and replace the custom data format. You will then use the other APIs to closely integrate the application with the features of the phone such as automatically dialing a contact and sending e-mail messages. You will also update the application to monitor for incoming or manually placed calls with a client so that client information is automatically displayed.

Lab Objective

This lab will take approximately 90 minutes.

The objective of this lab is to introduce the managed APIs that are available as part of the Windows Mobile platform and demonstrate their effectiveness in improving user efficiency. In this lab, you will do the following exercises:

- Using Pocket Outlook to Replace a Proprietary Data Store
- Using Contact Picker to Add Searching
- Using Telephony to Provide Automatic Dialing
- Using Messaging and the Picture Picker to Send E-Mail Messages with Attachments
- Using the State and Notification Broker API to Retrieve System State Information
- Using the State and Notification Broker API to Receive Notifications of Changes in System State

Exercise 1 – Using Pocket Outlook to Replace a Proprietary Data Store

In this exercise, you will update the application so that contact information comes from Pocket Outlook rather than relying on a separate, proprietary data store. The Pocket Outlook integration duplicates the existing application behavior as well as introduces a new Find Contact feature.

Task 1 – Populate the Emulator with Test Data

- Before starting the lab, you need to run an application to add the necessary Pocket Outlook contacts and image files to the emulator.
  - If Microsoft Visual Studio® 2005 is not already open, start it by navigating to **Start | All Programs | Microsoft Visual Studio 2005 Beta 2 | Microsoft Visual Studio 2005 Beta 2**.
  - In Visual Studio 2005, click **File | Open | Project/Solution**.
  - In the **Open Project** dialog box, browse to C:\Microsoft Hands-On-Lab\HOL-MBL02\Source\Setup Files\.
  - Select **HOL-MBL02Initialize.sln**.
  - Click the **Open** button. The HOL-MBL02Initialize solution should now open.
  - Verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the drop-down list on the Visual Studio 2005 Device toolbar as shown in Figure 1.

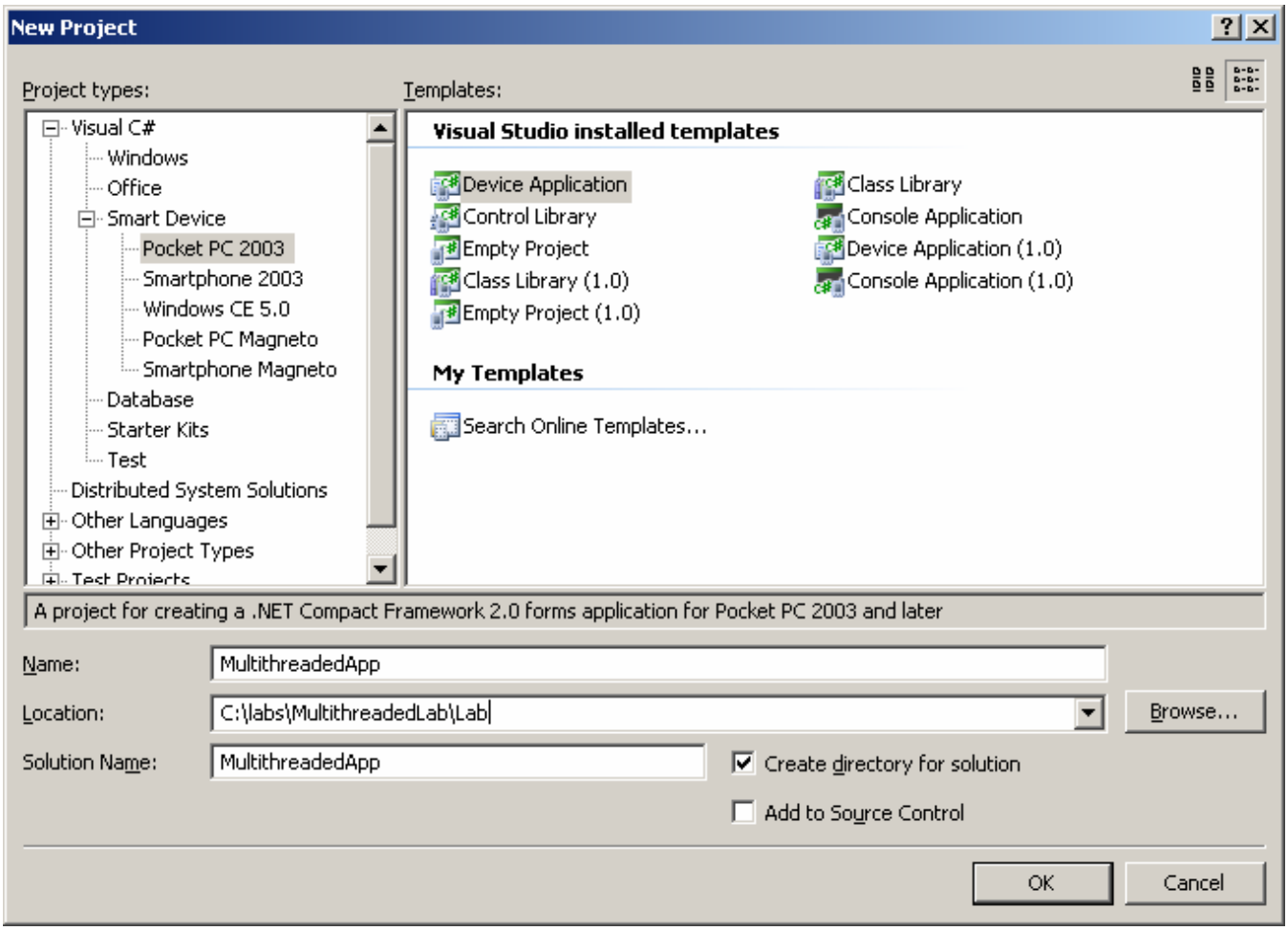

Figure 1 Emulator selection on the Visual Studio 2005 Device toolbar

- Start the application by clicking **Debug | Start Without Debugging** on the Visual Studio 2005 menu.
- If prompted by the **Deploy HOL-MBL02Initialize** dialog box, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the Device window, and then click **Deploy**.

  Note: If the emulator does not appear, look for an icon similar to Figure 2 on the Microsoft Windows® taskbar and click it to move the emulator to the foreground. Be aware that the first time the emulator starts, it may take several minutes.

Figure 2 Emulator icon on the Windows taskbar

- When the application appears on the emulator, press softkey 1 under the word **Start**. The application displays a wait cursor and indicates that contacts and then images are being added.

- When the application displays Initialization Complete, press softkey 2 under the word **Exit** to exit the application.
- Close the solution by clicking **File | Close Solution** on the Visual Studio 2005 menu.

Task 2 – Become Familiar with the Existing Application

- Before modifying the application, you should briefly become familiar with its behavior and implementation. To understand the program's current behavior, you will run it by using the Windows Mobile Smartphone emulator.
  - If Visual Studio 2005 is not already open, start it by navigating to **Start | All Programs | Microsoft Visual Studio 2005 Beta 2 | Microsoft Visual Studio 2005 Beta 2**.
  - In Visual Studio 2005, click **File | Open | Project/Solution**.
  - In the **Open Project** dialog box, browse to C:\Microsoft Hands-On-Lab\HOL-MBL02\Source\Exercises\.
  - Select **HOL-MBL02.sln**.
  - Click the **Open** button. The HOL-MBL02 solution should now open.

    Note: After they are opened, the solution, project, and source files should now be visible in the Visual Studio 2005 Solution Explorer. If the Solution Explorer doesn't automatically start, click **View | Solution Explorer** on the Visual Studio 2005 menu.

  - Verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the drop-down list on the Visual Studio 2005 Device toolbar as shown in Figure 1.
  - Start the application by clicking **Debug | Start Without Debugging** on the Visual Studio 2005 menu.
  - If prompted by the **Deploy HOL-MBL02** dialog box, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the Device window and then click **Deploy**.

    Note: If the emulator does not appear, look for an icon similar to Figure 2 on the Windows taskbar and click it to move the emulator to the foreground. Be aware that the first time the emulator starts, it may take several minutes.

  - On the emulator, press softkey 1 under the word **Next** to scroll through the list of contacts.
  - After you are comfortable with the behavior of the application, press softkey 2 under the word **Exit** to exit the application.
- To understand the implementation of the existing application, examine the contents of the FormMain.cs source file. To open this file, right-click the FormMain.cs file name in the Solution Explorer and click **View Code**.
- The key things to notice in this source file are the declaration of the **_dataManager** variable, **_currentItem** variable, and the **ShowNext** method. The **_dataManager** variable is the object responsible for managing the existing legacy data. The **_currentItem** variable is the currently selected contact. The **ShowNext** function is called each time the user clicks the **Next** option. It is

responsible for retrieving the next contact from **_dataManager**, storing it in **_currentItem**, and updating the form.

- Although this implementation provides an adequate solution for managing client data, it duplicates the features provided by Pocket Outlook and in many cases may duplicate the Pocket Outlook contact data as well.

Task 3 – Update Application to Use Pocket Outlook

- In this task, you will remove the existing legacy data management implementation from the application and update it to provide the same behavior by using Pocket Outlook.
- Add an assembly reference to the Pocket Outlook assembly, Microsoft.WindowsMobile.PocketOutlook.
  - Right-click the word **References** in the Visual Studio 2005 Solution Explorer, and click **Add Reference**.
  - In the **Add Reference** dialog box, click **Microsoft.WindowsMobile.PocketOutlook**.
  - Click the **OK** button.
- Add a **using** declaration for the Microsoft.WindowsMobile.PocketOutlook namespace.
  - If FormMain.cs is not already open in the Visual Studio 2005 editor, open it by right-clicking **FormMain.cs** in the Solution Explorer and clicking **View Code**.
  - Locate the existing **using** declarations at the top of the FormMain.cs file.
  - Add the **using** declaration for Microsoft.WindowsMobile.PocketOutlook immediately following the existing **using** declarations.

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

- To communicate with Pocket Outlook, applications must first create a session. The session provides connectivity to Pocket Outlook in much the same way that a database connection provides connectivity to a database and will be used throughout the application to communicate with Pocket Outlook. When the application no longer needs the connection, the session must be disposed of.
  - To create the Pocket Outlook session, add the following declaration to the FormMain.cs file immediately after the existing **_dataManager** declaration.

```
OutlookSession _outlook = new OutlookSession();
```

  - Locate the **menuExit_Click** method in the FormMain.cs file.
  - To dispose of the Pocket Outlook session when the application is closed, call the Pocket Outlook session's **Dispose** method in the **menuExit_Click** method immediately prior to the existing call to the **Close** method.

```
_outlook.Dispose();
```

- Contact items in Pocket Outlook are represented by the Contact class. To keep track of the current Contact displayed by your application, add the following declaration to FormMain.cs immediately after the **_currentItem** declaration.

```
Contact _currentContact = null;
```

- Now update the **ShowNext** method to scroll through the Pocket Outlook contacts. You are going to modify the implementation so that each time **ShowNext** is called it asks Pocket Outlook the index of the currently displayed contact and then increments that index by 1. Use the modulus operator (%) to loop back to the beginning when the end is reached.
  - Locate the **ShowNext** method in the FormMain.cs file, and declare an **int** variable named **index** with an initial value of 0 as the first line of the method.

```
int index = 0;
```

  - The index value needs to be incremented only if a contact is already displayed, so add an **IF** statement checking that the value of _currentContact is not null.

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

  - In the **IF** statement, first ask the Pocket Outlook Contacts folder's **ContactsCollection** the index of the currently displayed contact by calling the **IndexOf** method. Store the result in index.

```
index = _outlook.Contacts.Items.IndexOf(_currentContact);
```

  - Still in the **IF** statement, increment the index by 1. Use the modulus operator (%) to set the index back to 0 if the index reaches the number of contacts contained in the Contacts folder's **ContactsCollection**.

```
index = (index + 1) % _outlook.Contacts.Items.Count;
```

  - With the index of the next contact to retrieve determined, you can now use the Contacts folder's **ContactsCollection** indexer to retrieve that contact. Add this line immediately *after* the **IF** statement.

```
_currentContact = _outlook.Contacts.Items[index];
```

  - Now modify the four assignments to the form text boxes to use **_currentContact** rather than **_currentItme**. When you're finished, the lines should appear like the following.

```
txtName.Text = _currentContact.FileAs;
txtWorkPhone.Text = _currentContact.BusinessTelephoneNumber;
txtMobilePhone.Text = _currentContact.MobileTelephoneNumber;
txtNotes.Text = _currentContact.Body;
```

  - Finally, comment out the line in the **ShowNext** method, where **_currentItem** is assigned the value of **_dataManager.Next** because you no longer need to access the legacy data system. The complete **ShowNext** method should now look like the following.

```
private void ShowNext()
{
  int index = 0;
  if (_currentContact != null)
  {
    index = _outlook.Contacts.Items.IndexOf(_currentContact);
    index = (index + 1) % _outlook.Contacts.Items.Count;
  }

  _currentContact = _outlook.Contacts.Items[index];
  //_currentItem = _dataManager.Next;

  txtName.Text = _currentContact.FileAs;
  txtWorkPhone.Text = _currentContact.BusinessTelephoneNumber;
  txtMobilePhone.Text = _currentContact.MobileTelephoneNumber;
  txtNotes.Text = _currentContact.Body;
}
```

- You've added all of the necessary code. The last thing to do is to comment out the class declarations for **_dataManager** and **_currentItem** because you are no longer using them.
  - Locate the declaration of **_dataManager** just after the FormMain constructor, and comment out the line.

    ```
    // FieldSalesDataManager _dataManager = new FieldSalesDataManager();
    ```

  - Locate the **_currentItem** declaration a few lines further down, and comment it out as well.

    ```
    // FieldSalesData _currentItem;
    ```

- You are now ready to test the new version of the application.
  - Build your application by clicking **Build | Build Solution** on the Visual Studio 2005 menu. Correct any compilation errors before proceeding.
  - Verify that **Windows Mobile 5.0 Smartphone Emulator** is still selected in the drop-down list on the Visual Studio 2005 Device toolbar as shown in Figure 1.
  - Start the application by clicking **Debug | Start Without Debugging** on the Visual Studio 2005 menu.
  - If prompted by the **Deploy HOL-MBL02** dialog box, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the Device window, and then click **Deploy**.

    Note: If you should receive an error during deployment that indicates that the process or file is in use, this just means that the program is still running on the emulator and must be exited before a new copy can be deployed and run. See Appendix A of this document for instructions for exiting a running application.

  - Pressing softkey 1 under the word **Next**, scroll through the contact list. Notice that there are now several more contacts available than before. These additional contacts are available because the application is now reading directly from the Pocket Outlook contact list rather than a separate, proprietary data store.

- After you are comfortable with the behavior of the application, press softkey 2 under the word **Exit** to exit the application.

Exercise 2 – Using Contact Picker to Add Searching

In this exercise, you will add a search feature to the application that users can use to look up a specific contact rather than only being able to scroll through contacts as they do now. To implement this task, you use the **Contact Picker** class, **ChooseContactDialog**.

Task 1 – Add Assembly References and Namespace Declarations

- Before you can use the **ChooseContactDialog**, 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.
  - Right-click the word **References** in the Visual Studio 2005 Solution Explorer, and click **Add Reference**.
  - In the **Add Reference** dialog box, click **Microsoft.WindowsMobile.Forms**.

    Note: There are several assemblies visible in the **Add Reference** dialog box containing the word "Forms." Be sure to choose the assembly named **Microsoft.WindowsMobile.Forms**.

  - Click the **OK** button.
  - If FormMain.cs is not already open in the Visual Studio 2005 editor, open it by right-clicking **FormMain.cs** in the Solution Explorer and clicking **View Code**.
  - Locate the existing **using** declarations at the top of the FormMain.cs file.
  - Add the **using** declaration for Microsoft.WindowsMobile.Forms immediately following the existing **using** declarations.

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

Task 2 – Updating the Menu

- Now you need to modify the application menu to provide the user with additional options. You do this by deleting the existing **Exit** menu option and replacing it with a pop-up menu that provides the user with options to find a contact and exit the program.
  - Open FormMain.cs in design mode by double-clicking **FormMain.cs** in the Visual Studio 2005 Solution Explorer.
  - Locate the **Exit** menu option on the form designer as shown in Figure 3. Right-click the **Exit** menu option on the form designer, and click **Delete**. The words **Type Here** should replace the word **Exit**.

Figure 3 Application Exit menu option as it appears in the form designer

- Click the menu where **Type Here** appears, type the word **Menu** and press ENTER. The area around the word **Menu** should turn blue and the words **Type Here** should appear above.
- Type the word **Exit** and press ENTER.
- Type the word **Find** and press ENTER. The menu should now look similar to Figure 4.

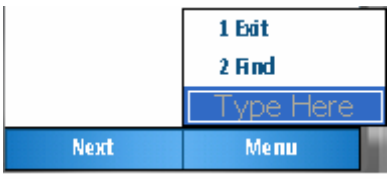

Figure 4 The new application menu as it appears in the form designer

- Right-click the word **Find** in the menu, and click **Properties**. In the Properties window, type **menuFind** for the value of (Name) and press ENTER.
- Right-click the word **Exit** in the menu, and click **Properties**. In the Properties window, type **menuExit** for the value of (Name) and press ENTER.

Task 3 – Add the Find Functionality

- You are now ready to add the functionality to let the user choose a specific contact.
  - Double-click the **Find** menu option in the form designer. This opens the FormMain.cs code and creates a **menuFind_Click** method.
  - In the **menuFind_Click** method, declare and create an instance of the **ChooseContactDialog** class; name the variable **contactDialog**.

```
ChooseContactDialog contactDialog = new ChooseContactDialog();
```

  - The **ChooseContactDialog** class provides the ability to select either a contact or specific contact properties such as an e-mail address or phone number. In this case, the user should select the actual contact, not individual properties. Set the **ChooseContactDialog ChooseContactOnly** property to **TRUE** to indicate that only the contact names are to be displayed.

```
contactDialog.ChooseContactOnly = true;
```

  - Display the **ChooseContactDialog** by calling the **ShowDialog** method. The **ShowDialog** method returns a **DialogResult**, which indicates whether the user made a selection or canceled the dialog box. Store the result of the call to **ShowDialog** in a **DialogResult** variable named **result**.

```
DialogResult result = contactDialog.ShowDialog();
```

  - When the call to **ShowDialog** is returned, you need to check whether the user actually made a selection. You do this by adding an **IF** statement to check whether the **result** variable contains the enumeration **DialogResult.OK**. If the user had canceled the dialog box without making a selection, the **result** variable would contain **DialogResult.Cancel**.

```
if (result == DialogResult.OK)
{
}
```

  - If the user did make a selection, the application should now display the selected contact. In the **IF** statement, assign the **contactDialog.SelectedContact** property to **_currentContact**.

```
_currentContact = contactDialog.SelectedContact;
```

  - Now that **_currentContact** is updated to reflect the user's selection, the only thing left to do is update the text boxes with this contact's information. The easiest way to do this is to the copy four lines from the **ShowNext** method where the **_currentContact** properties are assigned to

the form text boxes and past the lines into the body of the **IF** statement just after the assignment to **_currentContact**.

```
txtName.Text = _currentContact.FileAs;
txtWorkPhone.Text = _currentContact.BusinessTelephoneNumber;
txtMobilePhone.Text = _currentContact.MobileTelephoneNumber;
txtNotes.Text = _currentContact.Body;
```

- You've added all of the necessary code to the **menuNext_Click** method. It should now look like the following.

```
private void menuFind_Click(object sender, EventArgs e)
{
  ChooseContactDialog contactDialog = new ChooseContactDialog();
  contactDialog.ChooseContactOnly = true;
  DialogResult result = contactDialog.ShowDialog();
  if (result == DialogResult.OK)
  {
    _currentContact = contactDialog.SelectedContact;
    txtName.Text = _currentContact.FileAs;
    txtWorkPhone.Text = _currentContact.BusinessTelephoneNumber;
    txtMobilePhone.Text = _currentContact.MobileTelephoneNumber;
   txtNotes.Text = _currentContact.Body;
  }
}
```

- The last thing you need to do before testing your new Find feature is attach the code necessary to close the application to your **Exit** menu option.
  - Open FormMain.cs in design mode by double-clicking **FormMain.cs** in the Visual Studio 2005 Solution Explorer.
  - Double-click the **Exit** menu option in the form designer. This adds the new method **menuExit_Click_1**. The "_1" is appended to the name because the form already contains a **menuExit_Click** method.
  - Locate the existing **menuExit_Click** method. Copy the two lines contained in that method, and paste them in your new **menuExit_Click_1** method. The lines to copy are shown below for your reference.

```
_outlook.Dispose();
this.Close();
```

  - The original **menuExit_Click** method is no longer executed by the application because it was attached to the original **Exit** menu option, which you deleted to create the pop-up menu in the previous task. Because it is no longer executed, delete the **menuExit_Click** method.

Task 4 – Testing the New Functionality

- You are now ready to test the new application functionality.
  - Build your application by clicking **Build | Build Solution** on the Visual Studio 2005 menu. Correct any compilation errors before proceeding.

- Verify that **Windows Mobile 5.0 Smartphone Emulator** is still selected in the drop-down list on the Visual Studio 2005 Device toolbar as shown in Figure 1.
- Start the application by clicking **Debug | Start Without Debugging** on the Visual Studio 2005 menu.
- If prompted by the **Deploy HOL-MBL02** dialog box, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the Device window, and then click **Deploy**.
- Press softkey 2 under the word **Menu** to display the pop-up menu. Select the **Find** option either by clicking it by using your mouse or by using your mouse to select the number 2 on the emulator keypad.
- Select one of the contacts. The application then displays the appropriate information.
- Select the **Find** option again. When the Contact Picker opens, start typing a portion of one of the contact names. Notice that the list is automatically filtered based on the entered text. Select one of the contacts as before.
- Press softkey 1 under the word **Next**. Notice that the scrolling continues from the point of the most recent selection. This feature works correctly because the implementation of the **ShowNext** method you wrote in the previous task uses the **ContactCollection.IndexOf** method to determine the index to use when scrolling rather than just incrementing a simple index managed by the application.
- After you are comfortable that the new feature works correctly, exit the application by pressing softkey 2 under the word **Menu** and clicking **Exit**.

Exercise 3 – Using Telephony to Provide Automatic Dialing

In this exercise, you will use the new managed Telephony API to provide automatic phone dialing as part of your application. You will do this by adding two new menu options Dial Work Phone and Dial Mobile Phone, which will automatically dial the appropriate number for the currently displayed contact. Adding dialing support to your application allows the user to place a phone call without needing to dial it manually or leave your application.

Task 1 – Add Assembly References and Namespace Declarations

- Before you can use the Telephony API, you must add a reference for the Microsoft.WindowsMobile.Telephony assembly to the project and add a **using** declaration for the corresponding namespace to the source code.
  - Right-click the word **References** in the Visual Studio 2005 Solution Explorer, and click **Add Reference**.
  - In the **Add Reference** dialog box, click **Microsoft.WindowsMobile.Telephony**.
  - Click the **OK** button.
  - If FormMain.cs is not already open in the Visual Studio 2005 editor, open it by right-clicking **FormMain.cs** in the Solution Explorer and clicking **View Code**.
  - Locate the existing **using** declarations at the top of the FormMain.cs file.

- Add the **using** declaration for Microsoft.WindowsMobile.Telephony immediately following the existing **using** declarations.

```
using Microsoft.WindowsMobile.Telephony;
```

Task 2 – Updating the Menu

- Now modify the application menu to provide the user with the two new menu options.
  - Open FormMain.cs in design mode by double-clicking **FormMain.cs** in the Visual Studio 2005 Solution Explorer.
  - The new menu options are added to the same menu containing the **Exit** and **Find** options that you added in the last task. If the menu is not already expanded as shown in Figure 4, click the word **Menu** in the form designer to expand it. Then click the text **Type Here**.
  - Type **Dial Work Phone** and press ENTER.
  - Type **Dial Mobile Phone** and press ENTER.
  - Right-click **Dial Work Phone** on the menu, and click **Properties**. In the Properties window, type **menuDialWorkPhone** for the value of (Name) and press ENTER.
  - Right-click **Dial Mobile Phone** on the menu, and click **Properties**. In the Properties window, type **menuDialMobilePhone** for the value of (Name) and press ENTER.

Task 3 – Add Automatic Dialing

- In this task, you will use the **Phone** class to initiate a phone call to the current client by using the information stored in the **_currentContact** variable.
- First, add the code to place a call to the client's work phone.
  - Double-click the **Dial Work Phone** menu option in the form designer. This opens the FormMain.cs code and creates a **menuDialWorkPhone_Click** method.
  - In the **menuDialWorkPhone _Click** method, declare and create an instance of the **Phone class**; name the variable **telephone**.

```
Phone telephone = new Phone();
```

  - Now use the **Phone** class's **Talk** method to place a call to the current contact's **BusinessTelephoneNumber**.

```
telephone.Talk(_currentContact.BusinessTelephoneNumber);
```

- Now add the code to place a call to the client's mobile phone.
  - Open FormMain.cs in design mode by double-clicking **FormMain.cs** in the Visual Studio 2005 Solution Explorer.
  - Double-click the **Dial Mobile Phone** menu option in the form designer. This opens the FormMain.cs code and creates a **menuDialMobilePhone_Click** method.

- In the **menuDialMobilePhone_Click** method, add the code to declare an instance of the **Phone** class and place a call by using the current client's **MobileTelephoneNumber**.

- The **menuDialWorkPhone_Click** and **menuDialMobilePhone_Click** methods should now look like the following.

```
private void menuDialWorkPhone_Click(object sender, EventArgs e)
{
  Phone telephone = new Phone();
  telephone.Talk(_currentContact.BusinessTelephoneNumber);
}

private void menuDialMobilePhone_Click(object sender, EventArgs e)
{
  Phone telephone = new Phone();
  telephone.Talk(_currentContact.MobileTelephoneNumber);
}
```

Task 4 – Testing the New Functionality

- You are now ready to test the new application functionality.
  - Build your application by clicking **Build | Build Solution** on the Visual Studio 2005 menu. Correct any compilation errors before proceeding.
  - Verify that **Windows Mobile 5.0 Smartphone Emulator** is still selected in the drop-down list on the Visual Studio 2005 Device toolbar.
  - Start the application by clicking **Debug | Start Without Debugging** on the Visual Studio 2005 menu.
  - If prompted by the **Deploy HOL-MBL02** dialog box, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the Device window and then click **Deploy**.
  - Press softkey 2 under the word **Menu** to display the pop-up menu. Select the **Dial Work Phone** option to initiate the phone call. The emulator display switches to the phone screen. The phone screen first indicates that the phone is dialing and then is connected as shown in Figure 5.

Figure 5 The emulator display when dialing and connecting a phone call

Note: There may be a delay of a few seconds between the time you select the **Dial Work**

Phone menu option and when the emulator updates the display to show the phone screen.

- To end the phone call, click the emulator **End** button (the button that has a red telephone on it).
- Scroll through different contacts, and test both the **Dial Work Phone** and **Dial Mobile Phone** menu options. Verify that the call is placed to the correct phone number in each case.
- When you are comfortable that the new feature works correctly, exit the application by pressing softkey 2 under the word **Menu** and clicking **Exit**.

Exercise 4 – Using Messaging and the Picture Picker to Send E-Mail Messages with Attachments

In this exercise, you will use the new Picture Picker and managed Messaging API to provide the user with the capability to send a contact an e-mail message with a picture as an attachment. You will add this feature by providing a new **Send Picture** menu option.

Task 1 – Add Assembly References and Namespace Declarations

- Before you can use the Messaging API, you must add a reference for the System.Messaging assembly to the project and add a **using** declaration for the corresponding namespace to the source code. You do not need to add an assembly reference or namespace declaration for the Picture Picker because it is part of the Microsoft.WindowsMobile.Forms assembly and namespace that you already added in Exercise 2.
  - Right-click the word **References** in the Visual Studio 2005 Solution Explorer, and click **Add Reference**.
  - In the **Add Reference** dialog box, click **System.Messaging**.
  - Click the **OK** button.
  - If FormMain.cs is not already open in the Visual Studio 2005 editor, open it by right-clicking **FormMain.cs** in the Solution Explorer and clicking **View Code**.
  - Locate the existing **using** declarations at the top of the FormMain.cs file.
  - Add the **using** declaration for System.Messaging immediately following the existing **using** declarations.

```
using System.Messaging;
```

Task 2 – Updating the Menu

- Now modify the application menu to provide the user with a new menu option.
  - Open FormMain.cs in design mode by double-clicking **FormMain.cs** in the Visual Studio 2005 Solution Explorer.

- The new menu option is added to the same menu that you used in the previous two exercises. If the menu is not already expanded as shown in Figure 4, click the word **Menu** in the form designer to expand it. Then click the text **Type Here**.
- Type **Send Picture** and press ENTER.
- Right-click **Send Picture** in the menu, and click **Properties**. In the Properties window, type **menuSendPicture** for the value of (Name) and press ENTER.

Task 3 – Using Picture Picker

- In this task, you will use the Picture Picker to let the user select which photo he or she wants to send to the client.
  - Double-click the **Send Picture** menu option in the form designer. This opens the FormMain.cs code and creates a **menuSendPicture_Click** method.
  - In the **menuSendPicture_click** method, declare and create an instance of the **Picture Picker** class, **SelectPictureDialog**; name the variable **picturePicker**.

```
SelectPictureDialog picturePicker = new PicturePicker();
```

  - Now set the **SelectPictureDialog** class to initially display pictures in the \Images folder. Because in this application the user will only be showing professional photos, the application should not allow the user to select pictures that might be stored on an attached camera. So the **CameraAccess** property is set to **FALSE**.

```
picturePicker.InitialDirectory = @"\Images";
picturePicker.CameraAccess = false;
```

  - You must also consider how Digital Rights Management (DRM) issues are to be handled. Assuming that the users of this application have properly purchased any photos or artwork being sold, the application should allow the user to select DRM protected files. So the ShowDrmContent property is set to **TRUE.**

```
picturePicker.ShowDrmContent = true;
```

  - Any images that have been specifically protected against being forwarded should not be displayed because the attempt to send them by e-mail will fail. So the ShowForwardLockedContent property is set to **FALSE**.

```
picturePicker.ShowForwardLockedContent = false;
```

  - Now display the **SelectPictureDialog** class by using the **ShowDialog** method, and assign the returned DialogResult to a variable named **result**.

Task 4 – Using the Messaging API to E-Mail the Picture

- In this task, you will use the Messaging API to create an e-mail message, attach the selected picture, and display the message to the user before sending.

- The application should create an e-mail message only if the user actually selected a picture. So add an **IF** statement verifying that the user made a selection in **SelectPictureDialog**. Do this immediately after the call to **ShowDialog** in the **menuSendPicture_Click** method.

```
if (result == DialogResult.OK)
{
}
```

- In the **IF** statement block, declare and create an instance of the **EmailMessage** class; name the variable message.

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

- To make communicating with clients as easy as possible, the application should populate the e-mail message. Start by setting the Subject of the message to “The picture we discussed” and the BodyText to “Attached please find the picture we discussed. Please feel free to contact me with any questions”.

```
message.Subject = “The picture we discussed”;
message.BodyText = “Attached please find the picture we discussed. ” +
   “Please feel free to contact me with any questions”;
```

- Address the e-mail message by declaring and creating an instance of the **Recipient** class and pass the current client’s **Email1Address** to the constructor; name the variable **addressee**. Add the **addressee** to the **message.To** collection.

```
Recipient addressee = new Recipient(_currentContact.Email1Address);
message.To.Add(addressee);
```

- Now attach the user-selected picture to the message. This requires creating an instance of the **Attachment** class that passes the selected picture’s file name to the **Attachment** constructor. The **SelectPictureDialog** exposes the selected file name as the **FileName** property. After it’s created, the attachment is added to the **message.Attachments** collection.

```
Attachment picture = new Attachment(picturePicker.FileName);
message.Attachments.Add(picture);
```

- The message is now ready to send. Although the application could just send the message, it is probably better to give the user a chance to review and, if desired, modify the message before it’s sent. The **MessagingApplication.DisplayMessage** method is a static method that provides this capability because it uses the standard compose form of the device-messaging application to display an e-mail message. After the message is displayed, the user can review, modify, and send the message just as the user would if using the messaging application directly.

  Because it is not uncommon for a device to contain several messaging accounts, you should identify which account to use to display the message. If you do not do this, the **DisplayMessage** method will prompt the user to select the account. In this lab, assume that the first e-mail account exposed by the Pocket Outlook **EmailAccounts** collection is the default account.

  - While still in the **IF** statement block, declare an **EmailAccount** variable named **defaultAccount** and assign it the first account in the **EmailAccounts** collection.

```
EmailAccount defaultAccount = _outlook.EmailAccounts[0];
```

- To display the message to the user, call the **MessagingApplication.DisplayMessage** method and pass **defaultAccount** as the first argument and **message** as the second.

```
MessagingApplication.DisplayMessage(defaultAccount, message);
```

- You added all of the necessary code to the **menuSendPicture_Click** method. It should now look like the following.

```
private void menuSendPicture_Click(object sender, EventArgs e)
{
  SelectPictureDialog picturePicker = new SelectPictureDialog();
  picturePicker.InitialDirectory = @"\Images";
  picturePicker.CameraAccess = false;
  picturePicker.ShowDrmContent = true;
  picturePicker.ShowForwardLockedContent = false;
  DialogResult result = picturePicker.ShowDialog();

  if (result == DialogResult.OK)
  {
    EmailMessage message = new EmailMessage();
    message.Subject = "The picture we discussed";
    message.BodyText = "Attached is the picture we discussed. " +
        "Please feel free to contact me with any questions";
    Recipient addressee = new Recipient(_currentContact.Email1Address);
    message.To.Add(addressee);
    Attachment picture = new Attachment(picturePicker.FileName);
    message.Attachments.Add(picture);

    EmailAccount defaultAccount = _outlook.EmailAccounts[0];
    MessagingApplication.DisplayComposeForm(defaultAccount, message);
  }
}
```

Task 5 – Testing the New Functionality

- You are now ready to test the new application functionality.
  - Build your application by clicking **Build | Build Solution** on the Visual Studio 2005 menu. Correct any compilation errors before proceeding.
  - Verify that **Windows Mobile 5.0 Smartphone Emulator** is still selected in the drop-down list on the Visual Studio 2005 Device toolbar.
  - Start the application by clicking **Debug | Start Without Debugging** on the Visual Studio 2005 menu.
  - If prompted by the **Deploy HOL-MBL02** dialog box, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the Device window and then click **Deploy**.
  - Press softkey 2 under the word **Menu** to display the pop-up menu. Select the **Send Picture** option. The Picture Picker now appears and looks similar to Figure 6.

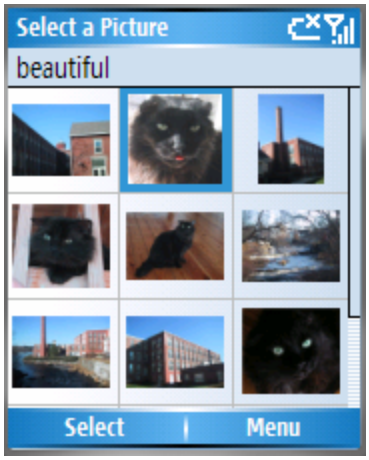

Figure 6 The Picture Picker

- Using the navigation pad, browse to a picture. Press softkey 1 under the word **Select** to select the picture. The e-mail form appears and is fully populated with the e-mail address, subject, body text, and attachment.
- At this point, you can just send the message by pressing softkey 1 under the word **Send**. If you want, you can modify any portion of the message before sending.
- Exit the application by pressing softkey 2 under the word **Menu** and clicking **Exit**.

- To verify that the message was actually submitted to the e-mail system, check the Pocket Outlook Outbox folder.
  - Display the Smartphone Home Screen by clicking the **Home** button (the button immediately below softkey 1 that has a picture of a house on it).
  - Press softkey 1 under the word **Start**. Browse to and select the **Messaging** icon.
  - Select **Outlook E-mail**.
  - Press softkey 2 under the word **Menu**.
  - Select **Folders**.
  - Select **Outbox**.
  - You should see a message with the subject “The picture we discussed.” This is the message sent by the application.

Exercise 5 – Using the State and Notification Broker API to Retrieve System State Information

In this exercise, you will use the new State and Notification Broker API to retrieve state information from the phone and be notified of changes to the phone’s state. The State and Notification Broker API is a very comprehensive API that provides access to over 100 device state values and state value change notifications. In this exercise, you will use it first to retrieve the device owner’s phone number. You will then use it to be notified of when a call comes into the phone so that the application will automatically

display information about the caller.

Task 1 – Add Assembly References and Namespace Declarations

- Before you can use the State and Notification Broker API, you must add a reference for the Microsoft.WindowsMobile.Status and Microsoft.WindowsMobile assemblies to the project. You also must add a **using** declaration for the Microsoft.WindowsMobile.Status namespace to the source code.
  - Right-click the word **References** in the Visual Studio 2005 Solution Explorer, and click **Add Reference**.
  - In the **Add Reference** dialog box, click **Microsoft.WindowsMobile.Status** and **Microsoft.WindowsMobile**.

    Note: You will not be using any classes from the Microsoft.WindowsMobile assembly. However, you must add a reference to it because the **SystemState** class (defined in the Microsoft.WindowsMobile.Status assembly) implements an interface defined in the Microsoft.WindowsMobile assembly.

  - Click the **OK** button.
  - If FormMain.cs is not already open in the Visual Studio 2005 editor, open it by right-clicking FormMain.cs in the Solution Explorer and clicking **View Code**.
  - Locate the existing **using** declarations at the top of the FormMain.cs file.
  - Add the **using** declaration for Microsoft.WindowsMobile.Status immediately following the existing **using** declarations.

```
using Microsoft.WindowsMobile.Status;
```

Task 2 – Using State and Notification Broker API to Retrieve Device Information

- As mentioned at the beginning of this exercise, the State and Notification Broker API provides access to over 100 different device state values. In this task, you will use the State and Notification Broker API to retrieve the device owner's name and phone number. You will then use this information to update the e-mail message you added in the last exercise to provide a more professional closing.
- Before performing this task, you need to populate the device owner information.
  - On the emulator, display the Smartphone Home Screen by pressing the **Home** button (the button immediately below softkey 1 that has a picture of a house on it).
  - Press softkey 1 under the word **Start**. Browse to and click the **Settings** icon.
  - Select **option 9 More**.
  - Select **option 4 Owner Information**.
  - Type **Judy Lew** for the value of Name.

- Type **603.555.9999** for the value of Telephone number.
- Press softkey 1 under the word **Done**.
- Return to the Smartphone Home Screen by clicking the **Home** button.

- Much of the functionally of the State and Notification Broker API is made available through the **SystemState** class. In this part of the task, you will use the **SystemState.OwnerName** and **SystemState.OwnerPhoneNumber** static properties to create the message closing phrase.
  - Locate the **menuSendPicture_Click** method that you added in the last exercise.
  - In the **menuSendPicture_Click** method, locate the line where **message.BodyText** is assigned to.
  - Just before this line, declare a string variable named **ownerName** and assign it the value of **SystemState.OwnerName**.

    ```
    string ownerName = SystemState.OwnerName;
    ```

  - On the next line, declare a string variable named **ownerPhoneNumber** and assign it the value of **SystemState.OwnerPhoneNumber**.

    ```
    string ownerPhoneNumber = SystemState.OwnerPhoneNumber;
    ```

  - Now generate a closing phrase for the e-mail message containing the owner name and phone number. Create the closing phrase by using **string.Format** and the format string "\nSincerely {0}\n{1}". Assign the result to a string variable named **closing**.

    ```
    string closing = string.Format("\nSincerely {0}\n{1}",
      ownerName, ownerPhoneNumber);
    ```

  - The last step is to concatenate the closing to the end of **message.BodyText**.

    ```
    message.BodyText = "Attached please find the picture we discussed. " +
       "Please feel free to contact me with any questions" + closing;
    ```

  - To verify that the message now contains the closing, follow the steps in Task 5 of Exercise 4. You should see that the e-mail message now contains a proper closing with "Sincerely Judy Lew" on the second line from the bottom of the message followed by "603.555.9999" on the next line.

Exercise 6 – Using the State and Notification Broker API to Receive Notifications of Changes in System State

In addition to providing access to the many state values on the device, the State and Notification Broker API also supports notifying an application of changes to a value. In this task, you will update the application to automatically show a contact's information if the contact should call the user or if the user should initiate the call from outside of the application. You do this by creating an instance of the **SystemState** class and using the **SystemProperty** enumeration to indicate that the application should be notified any time the user is talking to someone who is in the user's contact list. After the SystemState instance is created, the application can handle **SystemState Change** event.

Task 1 – Creating the SystemState Instance

- To monitor for state changes, you must first create an instance of the SystemState class and use the SystemProperty enumeration to indicate which value should be monitored.
  - Locate the declaration of the _**FormMain** constructor in FormMain.cs.
  - Immediately after the constructor body, declare a **SystemState** variable named _**phoneCallContactState**.
  - In the body of the **FormMain** constructor immediately after the call to **InitializeComponent**, construct a new instance of the **SystemState** class and pass the SystemProperty.PhoneTalkingCallerContact to the constructor. Assign the result to _**phoneCallContactState**. _phoneCallContactState =

```
new SystemState(SystemProperty.PhoneTalkingCallerContact);
```

- The _phoneCallContact SystemState instance now monitors **SystemState.PhoneTalkingCallerContact** for changes during execution of the application.

Task 2 – Handling SystemState Change Notifications

- Now add the code necessary for _**phoneCallContact** to notify your application when the value of **SystemState.PhoneTalkingCallerContact** changes. You do this by handling the _**phoneCallContact.Change** event.
  - On the next line (the line immediately following the construction of the SystemState instance), type the following code

```
_phoneCallContactState.Changed +=
```

  - After you type the equal sign (=), Visual Studio 2005 prompts you to press the TAB key. Press the TAB key. The line automatically completes to look similar to the following.

```
_phoneCallContactState.Changed +=
  new ChangEventHandler(_phoneCallContactState_Changed);
```

  - Visual Studio 2005 prompts you a second time to press the TAB key. Press the TAB key. Visual Studio 2005 generates a default implementation of the **phoneCallContactState_Changed** method to handle notifications of changes to the **SystemState.PhoneTalkingCallerContact** value.
- The application now contains all of the code necessary to handle changes to the value of the **SystemState.PhoneTalkingCallerContact** value. The FormMain constructor, **phoneCallContactState_Changed** method, and _**phoneCallContactState** declaration should look like the following.

```
public FormMain()
{
  InitializeComponent();

  _phoneCallContactState =
```

```
        new SystemState(SystemProperty.PhoneTalkingCallerContact);
    _phoneCallContactState.Changed +=
        new ChangeEventHandler(_phoneCallContactState_Changed);

    ShowNext();
}

void _phoneCallContactState_Changed(object sender,
    ChangeEventArgs args)
{
    throw new Exception("The method or operation is not implemented.");
}

SystemState _phoneCallContactState;
```

Task 3 – Displaying the Caller's Contact Information

- At this point, the **_phoneCallContactState_Changed** method is automatically called any time the user is in a phone conversation with someone in the contact list. The only thing left to do is to modify the **_phoneCallContactState_Changed** method so that contact information is changed.
  - In the body of the **_phoneCallerContactState_Changed** method, delete the line that throws the Exception.
  - Add an **IF** statement that checks that the value of **SystemState.PhoneTalkingCallerContact** is not null. This is necessary because the value is null any time the user is not involved in a phone call or if the phone call is with someone who is not in the contact list.

```
if (SystemState.PhoneTalkingCallerContact != null)
{
}
```

  - In the **IF** statement block, assign the value of **SystemState.PhoneTalkingCallerContact** to **_currentContact**.

```
_currentContact = SystemState.PhoneTalkingCallerContact;
```

  - Now that **_currentContact** contains a reference to the contact that the user is talking to, update the text boxes with the contact's information. The easiest way to do this is to the copy four lines from the **ShowNext** method where the **_currentContact** properties are assigned to the form textboxes and past the lines into the body of the **IF** statement just after the assignment to **_currentContact**.

```
    txtName.Text = _currentContact.FileAs;
    txtWorkPhone.Text = _currentContact.BusinessTelephoneNumber;
    txtMobilePhone.Text = _currentContact.MobileTelephoneNumber;
    txtNotes.Text = _currentContact.Body;
```

  - There's a strong chance that your application may be running in the background at the time the phone call occurs. To give your application form focus, call the **Activate** method.

```
this.Activate();
```

- The complete **_phoneCallContactState_Changed** method now looks like the following.

```
void _phoneCallContactState_Changed(object sender,
  ChangeEventArgs args)
{
  if (SystemState.PhoneTalkingCallerContact != null)
  {
    _currentContact = SystemState.PhoneTalkingCallerContact;
    txtName.Text = _currentContact.FileAs;
    txtWorkPhone.Text = _currentContact.BusinessTelephoneNumber;
    txtMobilePhone.Text = _currentContact.MobileTelephoneNumber;
    txtNotes.Text = _currentContact.Body;
    this.Activate();
  }
}
```

Task 4 – Testing the New Functionality

- You are now ready to test the new application functionality. The new feature automatically displays a contact's information if a contact calls the Smartphone or if the user manually calls a contact. There is no support for receiving calls to the emulator, so you'll test the feature by manually placing a phone call to one of the contacts.
  - Build your application by clicking **Build | Build Solution** on the Visual Studio 2005 menu. Correct any compilation errors before proceeding.
  - Verify that **Windows Mobile 5.0 Smartphone Emulator** is still selected in the drop-down list on the Visual Studio 2005 Device toolbar.
  - Start the application by clicking **Debug | Start Without Debugging** on the Visual Studio 2005 menu.
  - If prompted by the **Deploy HOL-MBL02** dialog box, verify that **Windows Mobile 5.0 Smartphone Emulator** is selected in the Device window and then click **Deploy**.
  - Display the Smartphone Home Screen by clicking the **Home** button. The application is still running in the background.
  - Using the emulator keypad, dial the phone number **6035551212**.
  - Press the **Talk** button (the button immediately below the **Home** button that has a picture of a green phone receiver on it). The phone dials and then connects the call. Just after the phone call connects, the contact information for Jim Wilson automatically appears.
  - Try repeating steps 5 through 7 by using a phone number such as "2122225555" that does not correspond to a contact. Notice that the application remains in the background.
  - Try another phone number such as "3105553662" that does correspond to a contact.
  - When you are satisfied that the application is working as you expect, exit the application by pressing softkey 2 under the word **Menu** and clicking **Exit**.

Lab Summary

In this lab, you performed the following exercises.

- Using Pocket Outlook to Replace a Proprietary Data Store
- Using Contact Picker to Add Searching
- Using Telephony to Provide Automatic Dialing
- Using Messaging and the Picture Picker to Send E-Mail with Attachments
- Using the State and Notification Broker API to Retrieve System State Information
- Using the State and Notification Broker API to Receive Notifications of Changes in System State

In this lab, you used the managed Windows Mobile APIs to update an existing .NET Compact Framework application to be more closely integrated with the supported features of the phone and improve user productivity by automating common tasks.