18

Developing Windows Mobile Applications

The mobile application platform has gained a lot of interest among enterprise developers in recent years. With so many mobile platforms available, customers are spoiled for choice. However, at the front of developers' minds are the various criteria that they need to evaluate before deciding on the platform to support. These factors are:

- ❑ Size of device install base
- ❑ Ease of development and support for widely known/used programming languages
- ❑ Capability to run one version of an application on a large number of devices

One mobile platform of choice among developers is the Microsoft Windows Mobile platform, now into its sixth generation. Today, the Windows Mobile platform is one of the most successful mobile device platforms in the market, with several handset manufacturers (such as HP, Asus, HTC, and even Sony Ericsson and Palm) supporting it.

This chapter presents the basics of Windows Mobile. It shows you how to create an RSS Reader application and then how to test and deploy the application to a real device. In particular, you will:

- ❑ Examine the basics of the Windows Mobile platform
- ❑ Learn how to download and install the various Software Development Kits (SDKs) to target the different platforms
- ❑ Create an RSS Reader application that allows users to subscribe to RSS feeds
- ❑ Explore various ways to deploy your Windows Mobile applications
- ❑ Create a professional-looking setup application to distribute your Windows Mobile applications

The Windows Mobile Platform

The Windows Mobile platform defines a device running the Windows CE operating system customized with a standard set of Microsoft-designed user interface shells and applications. Devices that use the Windows Mobile platform include:

- ❑ Pocket PCs
- ❑ Smartphones
- ❑ Portable Media Centers
- ❑ Automobile computing devices

For this chapter, the discussion is restricted to the first two categories — Pocket PCs and Smartphones. (The latter two categories use a different shell and are not widely used in today's market.)

The latest version of the Windows Mobile platform at the time of writing is Windows Mobile 6.1. With this new release, there are some new naming conventions. Here's a list of the Pocket PC and Smartphone names used by Microsoft over the years.

Pocket PCs	Smartphones
Pocket PC 2000/Pocket PC 2000 Phone Edition	
Pocket PC 2002/Pocket PC 2002 Phone Edition	Smartphone 2002
Windows Mobile 2003 for Pocket PC/Windows Mobile 2003 for Pocket PC Phone Edition	Windows Mobile 2003 for Smartphone
Windows Mobile 2003 SE (Second Edition) for Pocket PC/Windows Mobile 2003 SE (Second Edition) for Pocket PC Phone Edition	Windows Mobile 2003 SE for Smartphone
Windows Mobile 5.0 for Pocket PC/Windows Mobile 5.0 for Pocket PC Phone Edition	Windows Mobile 5.0 for Smartphone
Windows Mobile 6 Classic/Windows Mobile 6 Professional	Windows Mobile 6 Standard

Beginning with Windows Mobile 6, Microsoft defines a device with a touch screen but without phone capability as a Windows Mobile 6 Classic device (previously known as Pocket PC or Windows Mobile). Figure 18-1 shows a Windows Mobile 6 Classic device (the iPaq 211).

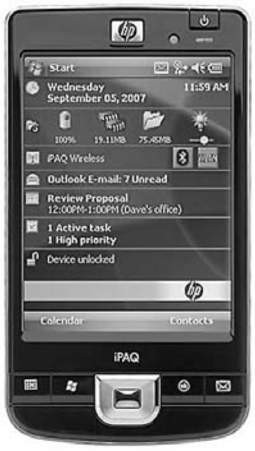


Figure 18-1

Touch-screen devices with phone functionality are now known as Windows Mobile 6 Professional (previously Windows Mobile Phone Edition). Figure 18-2 shows such a device (the HTC Touch Cruise).

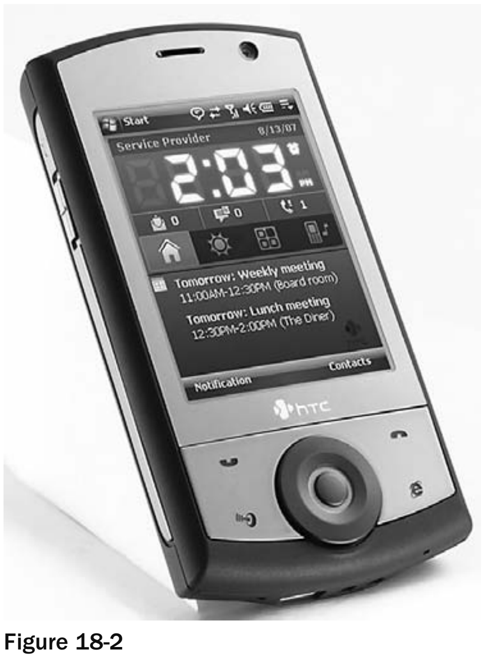


Figure 18-2

Devices that do not support touch screens are now known as Windows Mobile 6 Standard (previously Smartphones). One is the Moto Q9h, shown in Figure 18-3.

Figure 18-3

Developing Windows Mobile Applications Using the .NET Compact Framework

The easiest way to develop for the Windows Mobile platform is to use the Microsoft .NET Compact Framework (.NET CF). The .NET CF is a scaled-down version of the .NET Framework and is designed to work on Windows CE (a scaled-down version of the Windows OS supporting a subset of the Win32 APIs) based devices. The .NET CF contains a subset of the class libraries available on the desktop version of the .NET Framework and includes a few new libraries designed specifically for mobile devices.

At the time of writing, the latest version of .NET CF is version 3.5. Following is a list of the various version names of the .NET CF and their corresponding version numbers:

Version Name	Version Number
1.0 RTM	1.0.2268.0
1.0 SP1	1.0.3111.0
1.0 SP2	1.0.3316.0
1.0 SP3	1.0.4292.0
2.0 RTM	2.0.5238.0
2.0 SP1	2.0.6129.0
2.0 SP2	2.0.7045.0
3.5 Beta 1	3.5.7066.0
3.5 Beta 2	3.5.7121.0
RTM	3.5.7283.0

Source: `http://en.wikipedia.org/wiki/.NET_vCompact_Framework`

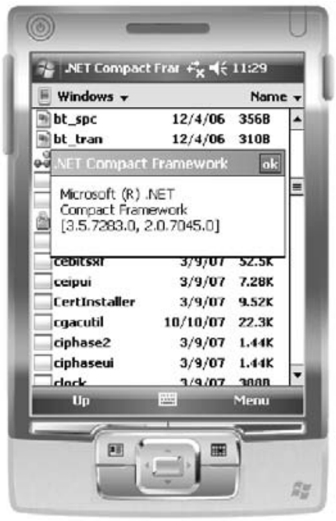
Knowing the version number of the .NET CF installed in your device is useful at development time because it helps you determine the exact version of the .NET CF installed on the target device/emulator.

As a developer, you can use either the C# or VB.NET language to write applications for the Windows Mobile platform. All the functionalities required by your applications can be satisfied by:

- The class libraries in the .NET CF, and/or
- APIs at the OS level via Platform Invoke (P/Invoke), and/or
- Alternative third-party class libraries such as the OpenNetCF's Smart Device Extension (SDE)

You can determine the versions of the .NET Compact Framework currently installed on your Windows Mobile device by going to Start ⇨ File Explorer and launching the `cgacutil.exe` utility located in \Windows.

Figure 18-4 shows the version of the .NET CF installed on a Windows Mobile emulator (more on this later).



```
bt_spc          12/4/06    356B
bt_tran         12/4/06    310B
cebitsxr         3/9/07    52.5K
ceipui           3/9/07    7.28K
CertInstaller    3/9/07    9.52K
cgacutil       10/10/07    22.3K
ciphase2         3/9/07    1.44K
ciphaseui        3/9/07    1.44K
clock            3/9/07    300B
```

```
.NET Compact Framework                ok

Microsoft (R) .NET
Compact Framework
[3.5.7283.0, 2.0.7045.0]
```

Figure 18-4

Windows Mobile 5.0 devices comes with the .NET CF 1.0 preinstalled in ROM, whereas the newer Windows Mobile 6 devices come with the .NET CF 2.0 preinstalled in ROM. If your application uses the newer .NET CF v3.5, you will need to install it onto the device before applications based on it can execute.

Obtaining the Appropriate SDKs and Tools

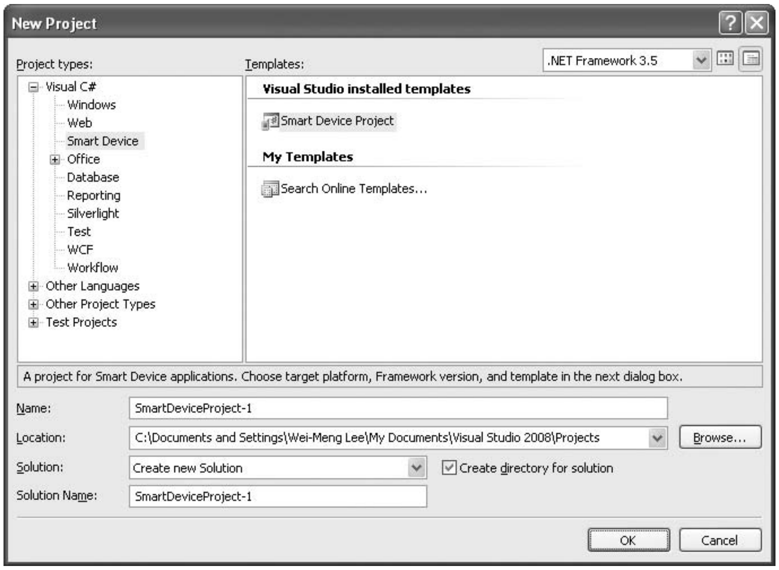
To develop Windows Mobile applications using the .NET CF, you need to download the SDK for each platform. Here are the SDKs you need:

- Windows Mobile 5.0 SDK for Pocket PC
- Windows Mobile 5.0 SDK for Smartphone
- Windows Mobile 6 Professional and Standard Software Development Kits Refresh

You can download the SDKs from Microsoft's web site (`http:// microsoft.com/downloads`) at no cost. The best tool to develop Windows Mobile applications using the .NET CF is to use the Visual Studio IDE, using Visual Studio 2005 Professional or above.

If you are using Visual Studio 2005, you need to download the Windows Mobile 5.0 SDK for Pocket PC and Smartphone (as described earlier). If you are using Visual Studio 2008, the Windows Mobile 5.0 SDKs for Pocket PC and Smartphone are already installed by default. For both versions, you need to download the Windows Mobile 6 SDKs to develop applications for Windows Mobile 6 devices.

With the relevant SDKs installed, the first step toward Windows Mobile development is to launch Visual Studio 2008 and create a new project. Select the Smart Device project type, and then select the Smart Device Project template (see Figure 18-5).

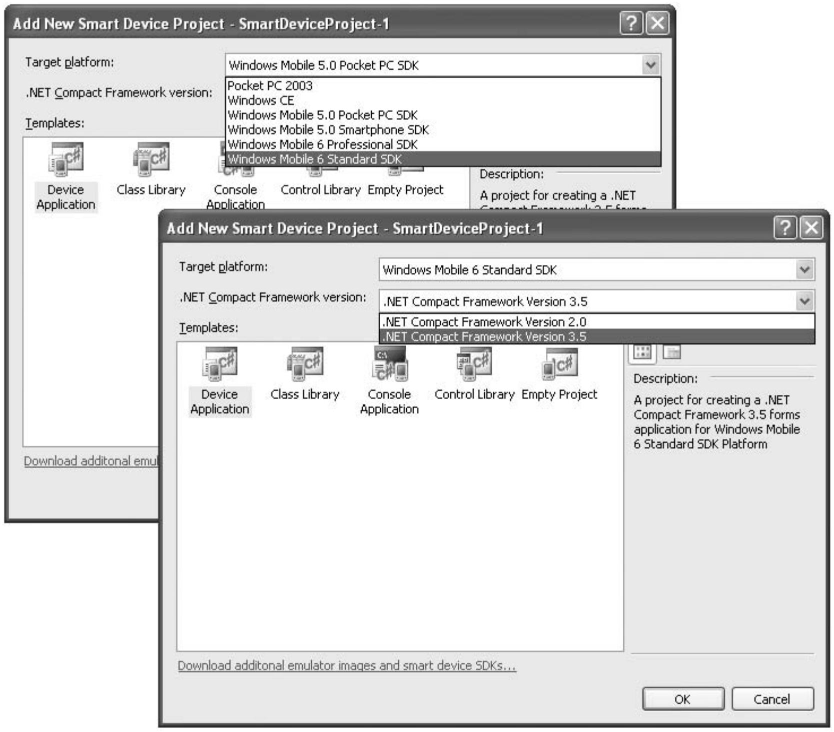


A project for Smart Device applications. Choose target platform, Framework version, and template in the next dialog box.

Name: SmartDeviceProject-1
Location: C:\Documents and Settings\Wei-Meng Lee\My Documents\Visual Studio 2008\Projects
Solution: Create new Solution
[✓] Create directory for solution
Solution Name: SmartDeviceProject-1

Figure 18-5

The Add New Smart Device Project dialog opens. You can select the target platform as well as the version of the .NET CF you want to use (see Figure 18-6).


Add New Smart Device Project - SmartDeviceProject-1

Target platform: Windows Mobile 5.0 Pocket PC SDK

.NET Compact Framework version:

Templates:

Pocket PC 2003
Windows CE
Windows Mobile 5.0 Pocket PC SDK
Windows Mobile 5.0 Smartphone SDK
Windows Mobile 6 Professional SDK
Windows Mobile 6 Standard SDK

Add New Smart Device Project - SmartDeviceProject-1

Target platform: Windows Mobile 6 Standard SDK

.NET Compact Framework version: .NET Compact Framework Version 3.5

.NET Compact Framework Version 2.0
.NET Compact Framework Version 3.5

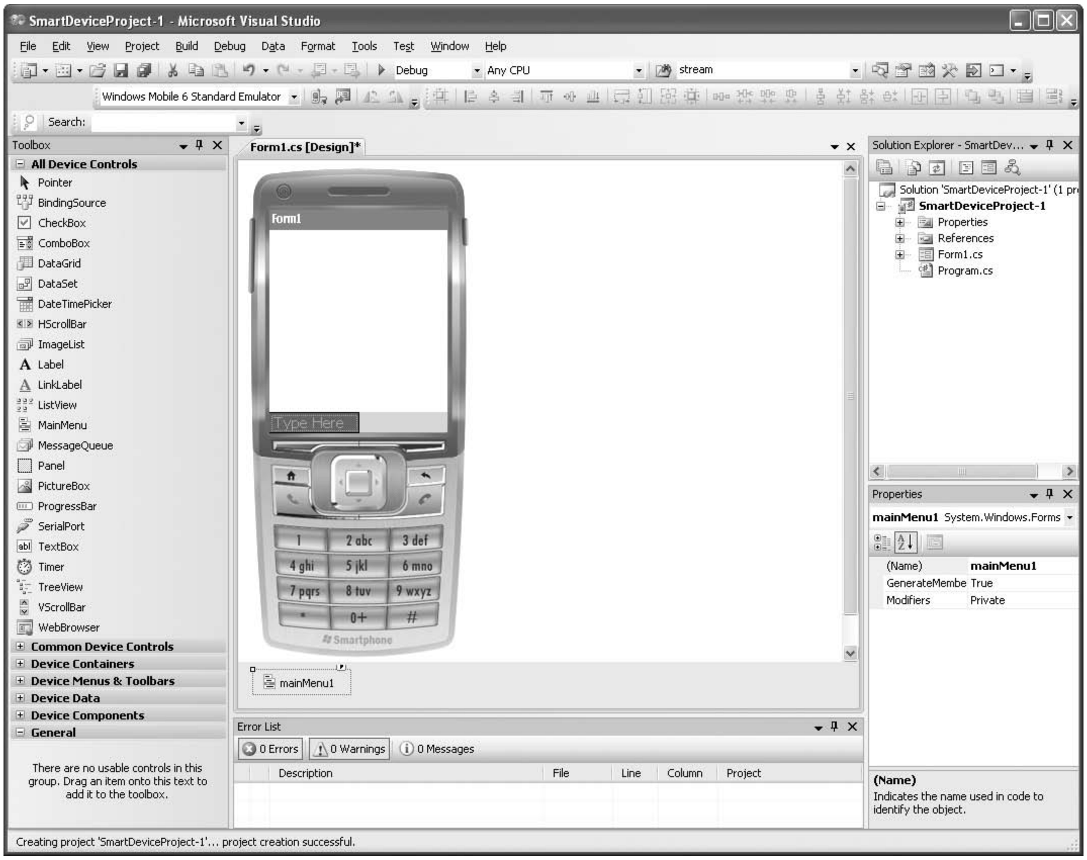
Templates:

Description:

A project for creating a .NET Compact Framework 3.5 forms application for Windows Mobile 6 Standard SDK Platform

Figure 18-6

You are now ready to start developing for Windows Mobile. Figure 18-7 shows the design view of a Windows Mobile Form in Visual Studio 2008 designer.



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

(Name)	mainMenu1
GenerateMembe	True
Modifiers	Private

(Name)
Indicates the name used in code to identify the object.

Creating project 'SmartDeviceProject-1'... project creation successful.

Figure 18-7

Building the RSS Reader Application

With the recent introduction of the Windows Mobile 6 platforms, we are now beginning to see a proliferation of new devices supporting Windows Mobile 6 Standard (aka Smartphone). As Windows Mobile 6 Standard devices do not have touch screens, they pose certain challenges when developing applications to run on them. Hence, in this section you will learn how to develop a Windows Mobile 6 Standard application that allows users to subscribe to RSS feeds.

The RSS Reader application has the following capabilities:

- ❑ Can subscribe to RSS feeds as well as unsubscribe from feeds
- ❑ Can cache the feeds as XML files on the device so that if the device goes offline the feeds are still available
- ❑ Uses a web browser to view the content of a post

Building the User Interface

To get started, launch Visual Studio 2008 and create a new Windows Mobile 6 Standard application using .NET CF 3.5. Name the application `RSSReader`.

Don't forget to download the free Windows Mobile 6 Standard SDK (`http://microsoft.com/downloads`*). You need it to create the application detailed in this chapter.*

The default Form1 uses the standard form factor of 176x180 pixels. As this application is targeted at users with wide-screen devices, change the `FormFactor` property of `Form1` to `Windows Mobile 6 Landscape QVGA`.

Populate the default `Form1` with the following controls (see also Figure 18-8):

- ❑ One `TreeView` control
- ❑ Four `MenuItem` controls

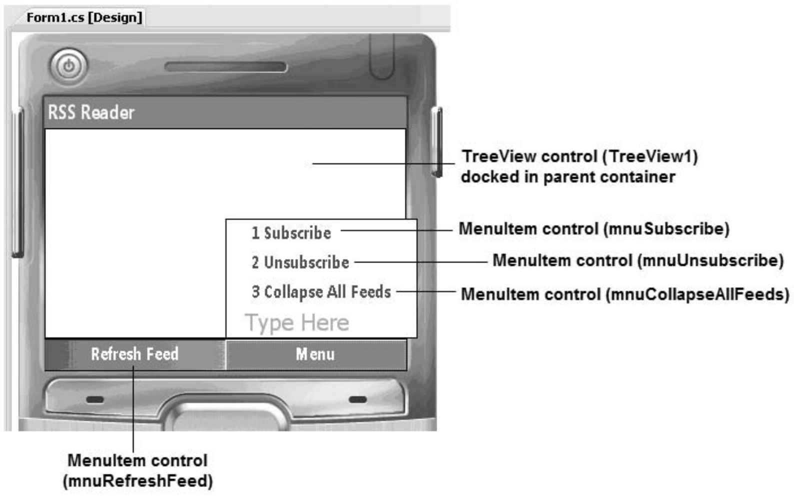

Form1.cs [Design]
RSS Reader
1 Subscribe
2 Unsubscribe
3 Collapse All Feeds
Type Here
Refresh Feed
Menu
TreeView control (TreeView1) docked in parent container
MenuItem control (mnuSubscribe)
MenuItem control (mnuUnsubscribe)
MenuItem control (mnuCollapseAllFeeds)
MenuItem control (mnuRefreshFeed)

Figure 18-8

Add an `ImageList` control to `Form1` and add three images to its `Images` property (see Figure 18-9).

You can download the images from this book's source code at its Wrox web site.

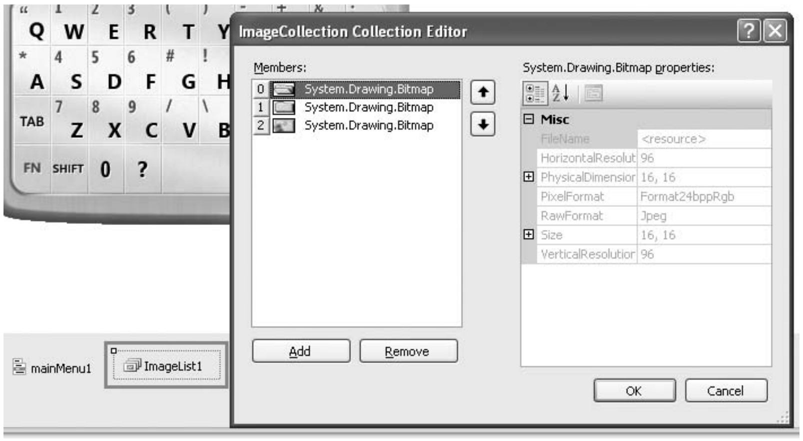


Figure 18-9

These images will be used by the `TreeView` control to display its content when the tree is expanded or closed. Hence, associate the `ImageList` control to the `TreeView` control by setting the `ImageList` property of the `TreeView` control to `ImageList1`.

Add a new Windows Form to the project, and populate it with a `WebBrowser` and `MenuItem` control (see Figure 18-10). The `WebBrowser` control will be used to view the content of a posting.

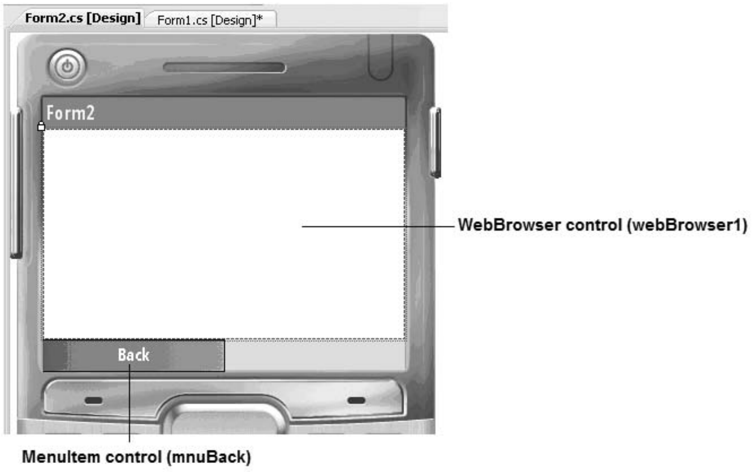


Figure 18-10

Set the `Modifiers` property of the `WebBrowser` control to `Internal` so that the control is accessible from other forms. Specifically, you want to set the content of the control from within `Form1`.

Switch to the code behind of `Form1`, and import the following namespaces:

```
using System.IO;
using System.Net;
using System.Xml;
using System.Text.RegularExpressions;
```

Declare the following constants and variable:

```
namespace RSSReader
{
    public partial class Form1 : Form
    {
        //---constants for icons---
        const int ICO_OPEN = 0;
        const int ICO_CLOSE = 1;
        const int ICO_POST = 2;

        //---file containing the list of subscribed feeds---
        string feedsList = @"\Feeds.txt";

        //---app's current path---
        string appPath = string.Empty;

        //---the last URL entered (subscribe)---
        string lastURLEntered = string.Empty;

        //---used for displaying a wait message panel---
        Panel displayPanel;

        //---for displaying individual post---
        Form2 frm2 = new Form2();
```

Creating the Helper Methods

When RSS feeds are being downloaded, you want to display a message on the screen to notify the user that the application is downloading the feed (see Figure 18-11).

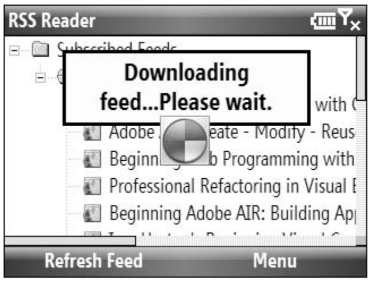

RSS Reader
Subscribed Feeds
Downloading feed...Please wait.
with
Adobe eate - Modify - Reus
Beginn b Programming with
Professional Refactoring in Visual
Beginning Adobe AIR: Building Ap
Refresh Feed
Menu

Figure 18-11

For this purpose, you can improvise with the aid of the `Panel` and `Label` controls. Define the `CreatePanel()` function so that you can dynamically create the message panel using a couple of `Panel` controls and a `Label` control:

```
//---create a Panel control to display a message---
private Panel CreatePanel(string str)
{
    //---background panel---
    Panel panel1 = new Panel()
    {
        BackColor = Color.Black,
        Location = new Point(52, 13),
        Size = new Size(219, 67),
        Visible = false,
    };
    panel1.BringToFront();

    //---foreground panel---
    Panel panel2 = new Panel()
    {
        BackColor = Color.LightYellow,
        Location = new Point(3, 3),
        Size = new Size(panel1.Size.Width - 6, panel1.Size.Height - 6)
    };

    //---add the label to display text---
    Label label = new Label()
    {
        Font = new Font(FontFamily.GenericSansSerif, 12, FontStyle.Bold),
        TextAlign = ContentAlignment.TopCenter,
        Location = new Point(3, 3),
        Size = new Size(panel2.Size.Width - 6, panel2.Size.Height - 6),
        Text = str
    };

    //---adds the label to Panel2---
    panel2.Controls.Add(label);

    //---adds the Panel2 to Panel1---
    panel1.Controls.Add(panel2);
    return panel1;
}
```

For simplicity, you are hardcoding the location of `panel1` (assuming that this application is running on a wide-screen device). Figure 18-12 shows the various controls forming the display panel.

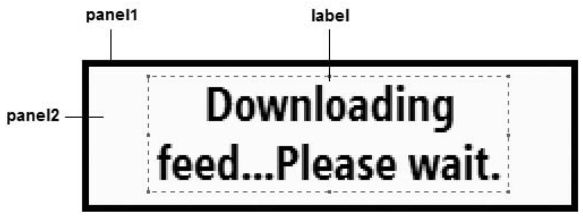


Figure 18-12

Next, define the `IsConnected()` function to test whether the user is connected to the Internet:

```
//---check if you are connected to the Internet---
private bool IsConnected()
{
    try
    {
        string hostName = Dns.GetHostName();
        IPHostEntry curhost = Dns.GetHostEntry(hostName);
        return (curhost.AddressList[0].ToString() !=
            IPAddress.Loopback.ToString());
    }
    catch (Exception)
    {
        return false;
    }
}
```

`Dns` is a static class that provides simple domain name resolution. The `GetHostName()` method gets the host name of the local computer, which is then passed to the `GetHostEntry()` method of the `Dns` class to obtain an `IPHostEntry` object. `IPHostEntry` is a container class for Internet host address information. Using this object, you can access its `AddressList` property to obtain the list of IP addresses associated with it. If the first member of the `AddressList` property array is not the loopback address (`127.0.0.1`; represented by `IPAddress.Loopback`), it is assumed that there is Internet connectivity.

Next, define the `DownloadFeed()` function, which takes in the URL for the feed you want to download and a title argument (to return the title of the feed). Each post title and its corresponding description is appended to a string and returned to the calling function:

```
//---download feed and extract Title and Description for each post---
private string DownloadFeed(string feedURL, ref string title)
{
    XmlDocument xml = new XmlDocument();

    //---always load from storage first---
    string FileName =
        appPath + @"\" + RemoveSpecialChars(feedURL) + ".xml";

    if (File.Exists(FileName))
    {
        xml.Load(FileName);
    }
    else
    {
        //---check if there is network connectivity---
        if (IsConnected())
        {
            WebRequest ftpReq = null;
            WebResponse ftpResp = null;
            Stream ftpRespStream = null;
            StreamReader reader = null;
            bool getRSSFeedFailed = false;
```

(continued)

(continued)

```
                try
                {
                    //---download the RSS document---
                    ftpReq = WebRequest.Create(feedURL);
                    ftpResp = ftpReq.GetResponse();
                    ftpRespStream = ftpResp.GetResponseStream();
                    reader = new StreamReader(ftpRespStream,
                        System.Text.Encoding.UTF8);

                    //---load the RSS document into an XMLDocument object---
                    xml.Load(reader);

                    //---save a local copy of the feed document---
                    xml.Save(FileName);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                    getRSSFeedFailed = true;
                }
                finally
                {
                    if (ftpRespStream != null)
                    {
                        ftpRespStream.Dispose();
                        ftpRespStream.Close();
                    };
                    if (ftpResp != null) ftpResp.Close();
                }
                if (getRSSFeedFailed) return String.Empty;
            }
            else
            {
                return String.Empty;
            }
        }

        //---get the title of the feed---
        XmlNode titleNode = xml.SelectSingleNode(@"rss/channel/title");
        title = titleNode.InnerText;

        //---select all <rss><channel><item> elements---
        XmlNodeList nodes = xml.SelectNodes("rss/channel/item");

        string result = String.Empty;
        foreach (XmlNode node in nodes)
        {
            //---select each post's <title> and <description> elements---
            result += node.SelectSingleNode("title").InnerText + ((char)3);
            result += node.SelectSingleNode("description").InnerText +
                ((char)12);
        }
        return result;
    }
```

To download the RSS feed XML documents, you use the WebRequest and WebResponse classes. The document is then read using a StreamReader object and loaded into an XmlDocument object. Each post title and its description are separated by the ASCII character 3, and each posting is separated by the ASCII character 12, like this:

```
Post_Title<3>Post_Description<12>Post_Title<3>Post_Description<12>
Post_Title<3>Post_Description<12>Post_Title<3>Post_Description<12>
Post_Title<3>Post_Description<12>...
```

Notice that after the XML feed for an URL is downloaded, it is saved onto storage. This ensures that the application continues to work in offline mode (when user disconnects from the Internet). The URL of the feed is used as the filename, minus all the special characters within the URL, with the .xml extension appended. For example, if the feed URL is http://www.wrox.com/WileyCDA/feed/RSS_WROX_ALLNEW.xml, then the filename would be httpwwwwroxcomWileyCDAfeedRSSWROXALLNEWxml.xml. To strip off all the special characters in the URL, define the RemoveSpecialChars() function as follows:

```
        //---removes special chars from an URL string---
        private string RemoveSpecialChars(string str)
        {
            string NewString = String.Empty;
            Regex reg = new Regex("[A-Z]|[a-z]");

            MatchCollection coll = reg.Matches(str);
            for (int i = 0; i <= coll.Count - 1; i++)
                NewString = NewString + coll[i].Value;

            return NewString;
        }
```

You use the Regex (regular expression) class to extract all the alphabets from the URL and append them into a string, which will be returned to the calling function to use as a filename.

Next, define the SubscribeFeed() function to subscribe to a feed, and then add each post to the TreeView control (see Figure 18-13):

```
        //---returns true if subscription is successful---
        private bool SubscribeFeed(string URL)
        {
            bool succeed = false;
            try
            {
                //---display the wait message panel---
                if (displayPanel == null)
                {
                    displayPanel = CreatePanel("Downloading feed...Please wait.");
                    this.Controls.Add(displayPanel);
                }
                else
                {
                    displayPanel.BringToFront();
                    displayPanel.Visible = true;
```

(continued)

(continued)

```
                Cursor.Current = Cursors.WaitCursor;
                //---update the UI---
                Application.DoEvents();
            }

            //---download feed---
            string title = String.Empty;
            string[] posts = DownloadFeed(URL, ref title).Split((char)12);
            if (posts.Length > 0 && posts[0] != String.Empty)
            {
                //---always add to the root node---
                TreeNode FeedTitleNode = new TreeNode()
                {
                    Text = title,
                    Tag = URL,  //---stores the Feed URL---
                    ImageIndex = ICO_CLOSE,
                    SelectedImageIndex = ICO_OPEN
                };

                //---add the feed title---
                TreeView1.Nodes[0].Nodes.Add(FeedTitleNode);

                //---add individual elements (posts)---
                for (int i = 0; i <= posts.Length - 2; i++)
                {
                    //---extract each post as "title:description"---
                    string[] str = posts[i].Split((char)3);

                    TreeNode PostNode = new TreeNode()
                    {
                        Text = str[0], //---title---
                        Tag = str[1],  //---description---
                        ImageIndex = ICO_POST,
                        SelectedImageIndex = ICO_POST
                    };

                    //---add the posts to the tree---
                    TreeView1.Nodes[0].Nodes
                        [TreeView1.Nodes[0].Nodes.Count - 1].
                        Nodes.Add(PostNode);
                }
                //---subscription is successful---
                succeed = true;

                //---highlight the new feed and expand its post---
                TreeView1.SelectedNode = FeedTitleNode;
            }
            else
                succeed = false;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
```

```
                //---subscription is not successful---
                succeed = false;
            }
            finally
            {
                //---clears the panel and cursor---
                Cursor.Current = Cursors.Default;
                displayPanel.Visible = false;

                //---update the UI---
                Application.DoEvents();
            }
            return succeed;
        }
```

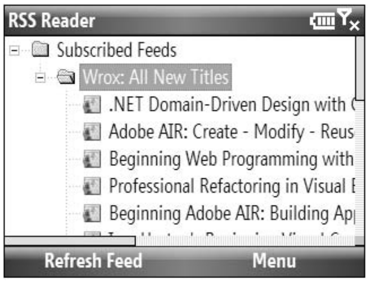

RSS Reader
Subscribed Feeds
Wrox: All New Titles
.NET Domain-Driven Design with
Adobe AIR: Create - Modify - Reus
Beginning Web Programming with
Professional Refactoring in Visual
Beginning Adobe AIR: Building Ap
Refresh Feed
Menu

Figure 18-13

For each TreeView node representing a feed title (such as Wrox: All New Titles), the `Text` property is set to the feed's title and its URL is stored in the `Tag` property of the node. For each node representing a posting (.NET Domain-Driven Design and so forth), the `Text` property is set to the posting's title and its description is stored in the `Tag` property.

Wiring All the Event Handlers

With the helper functions defined, let's wire up all the event handlers for the various controls. First, code the `Form1_Load` event handler as follows:

```
        private void Form1_Load(object sender, EventArgs e)
        {
            //---find out the app's path---
            appPath = Path.GetDirectoryName(
                 System.Reflection.Assembly.GetExecutingAssembly().
                 GetName().CodeBase);

            //---set the feed list to be stored in the app's folder---
            feedsList = appPath + feedsList;

            try
            {
                //---create the root node---
```

(continued)

(continued)

```
                TreeNode node = new TreeNode()
                {
                    ImageIndex = ICO_CLOSE,
                    SelectedImageIndex = ICO_OPEN,
                    Text = "Subscribed Feeds"
                };

                //---add the node to the tree---
                TreeView1.Nodes.Add(node);
                TreeView1.SelectedNode = node;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                return;
            }

            try
            {
                //---load all subscribed feeds---
                if (File.Exists(feedsList))
                {
                    TextReader textreader = File.OpenText(feedsList);

                    //---read URLs of all the subscribed feeds---
                    string[] feeds = textreader.ReadToEnd().Split('|');
                    textreader.Close();

                    //---add all the feeds to the tree---
                    for (int i = 0; i <= feeds.Length - 2; i++)
                        SubscribeFeed(feeds[i]);
                }
                else
                {
                    //---pre-subscribe to a few feed(s)---
                    SubscribeFeed(
                        "http://www.wrox.com/WileyCDA/feed/RSS_WROX_ALLNEW.xml");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

When the form is first loaded, you have to create a root node for the `TreeView` control and load all the existing feeds. All subscribed feeds are saved in a plain text file (`Feeds.txt`), in the following format:

```
Feed URL|Feed URL|Feed URL|
```

An example is:

```
http://news.google.com/?output=rss|http://rss.cnn.com/rss/cnn_topstories.rss|
```

If there are no existing feeds (that is, if `Feeds.txt` does not exist), subscribe to at least one feed.

In the `Click` event handler of the `Subscribe MenuItem` control, prompt the user to input a feed's URL, and then subscribe to the feed. If the subscription is successful, save the feed URL to file:

```
        private void mnuSubscribe_Click(object sender, EventArgs e)
        {
            if (!IsConnected())
            {
                MessageBox.Show("You are not connected to the Internet.");
                return;
            }

            //---add a reference to Microsoft.VisualBasic.dll---
            string URL = Microsoft.VisualBasic.Interaction.InputBox(
                "Please enter the feed URL", "Feed URL", lastURLEntered, 0, 0);

            if (URL != String.Empty)
            {
                lastURLEntered = URL;

                //---if feed is subscribed successfully---
                if (SubscribeFeed(URL))
                {
                    //---save in feed list---
                    TextWriter textwriter = File.AppendText(feedsList);
                    textwriter.Write(URL + "|");
                    textwriter.Close();
                }
                else
                {
                    MessageBox.Show("Feed not subscribed. " +
                    "Please check that you have entered " +
                    "the correct URL and that you have " +
                    "Internet access.");
                }
            }
        }
```

C# does not include the `InputBox()` function that is available in VB.NET to get user's input (see Figure 18-14). Hence, it is a good idea to add a reference to the `Microsoft.VisualBasic.dll` library and use it as shown in the preceding code.

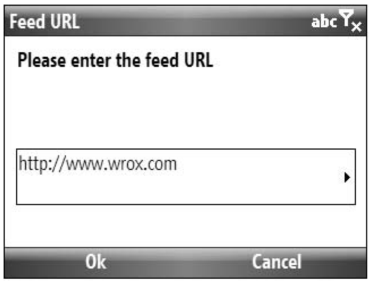


Figure 18-14

Whenever a node in the `TreeView` control is selected, you should perform a check to see if it is a posting node and enable/disable the MenuItem controls appropriately (see Figure 18-15):

```
//---fired after a node in the TreeView control is selected---
private void TreeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    //---if a feed node is selected---
    if (e.Node.ImageIndex != ICO_POST && e.Node.Parent != null)
    {
        mnuUnsubscribe.Enabled = true;
        mnuRefreshFeed.Enabled = true;
    }
    else
    {  //---if a post node is selected---
        mnuUnsubscribe.Enabled = false;
        mnuRefreshFeed.Enabled = false;
    }
}
```

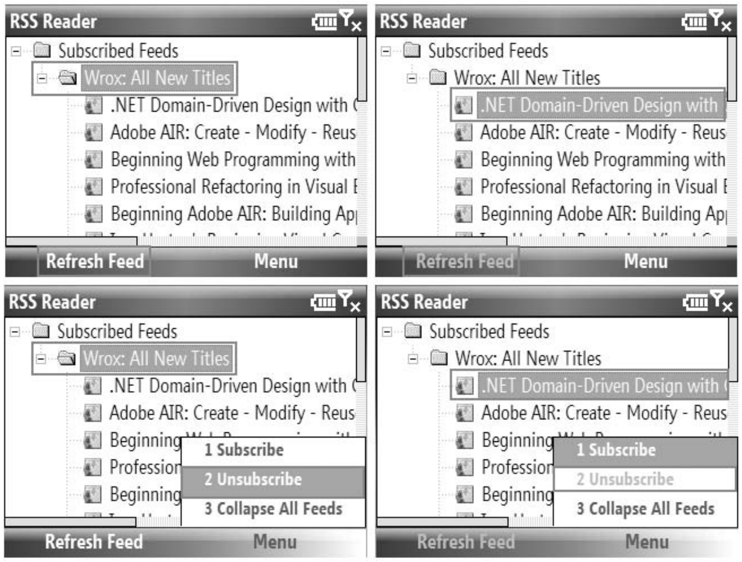

RSS Reader
Subscribed Feeds
Wrox: All New Titles
.NET Domain-Driven Design with
Adobe AIR: Create - Modify - Reus
Beginning Web Programming with
Professional Refactoring in Visual
Beginning Adobe AIR: Building Ap
Refresh Feed
Menu
1 Subscribe
2 Unsubscribe
3 Collapse All Feeds

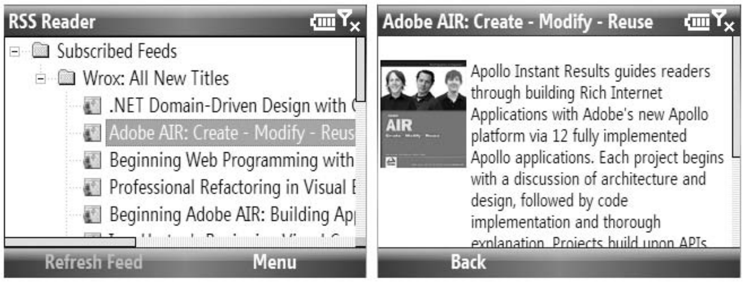
Figure 18-15

When the user selects a post using the Select button on the navigation pad, `Form2` containing the `WebBrowser` control is loaded and its content set accordingly (see Figure 18-16). This is handled by the `KeyDown` event handler of the TreeView control:

```
//---fired when a node in the TreeView is selected
// and the Enter key pressed---
private void TreeView1_KeyDown(object sender, KeyEventArgs e)
{
    TreeNode node = TreeView1.SelectedNode;
    //---if the Enter key was pressed---
    if (e.KeyCode == System.Windows.Forms.Keys.Enter)
    {
        //---if this is a post node---
        if (node.ImageIndex == ICO_POST)
        {
            //---set the title of Form2 to title of post---
            frm2.Text = node.Text;

            //---modifier for webBrowser1 in Form2 must be set to
            // Internal---
            //---set the webbrowser control to display the post content---
            frm2.webBrowser1.DocumentText = node.Tag.ToString();

            //---show Form2---
            frm2.Show();
        }
    }
}
```


RSS Reader
Subscribed Feeds
Wrox: All New Titles
.NET Domain-Driven Design with C
Adobe AIR: Create - Modify - Reus
Beginning Web Programming with
Professional Refactoring in Visual E
Beginning Adobe AIR: Building Ap
Refresh Feed
Menu
Adobe AIR: Create - Modify - Reuse
AIR
Back

Apollo Instant Results guides readers through building Rich Internet Applications with Adobe's new Apollo platform via 12 fully implemented Apollo applications. Each project begins with a discussion of architecture and design, followed by code implementation and thorough explanation. Projects build upon APIs

Figure 18-16

To unsubscribe a feed, you remove the feed's URL from the text file and then remove the feed node from the `TreeView` control. This is handled by the `Unsubscribe MenuItem` control:

```
//---Unsubscribe a feed---
private void mnuUnsubscribe_Click(object sender, EventArgs e)
{
    //---get the node to unsubscribe---
    TreeNode CurrentSelectedNode = TreeView1.SelectedNode;

    //---confirm the deletion with the user---
    DialogResult result =
       MessageBox.Show("Remove " + CurrentSelectedNode.Text + "?",
       "Unsubscribe", MessageBoxButtons.YesNo,
       MessageBoxIcon.Question,
       MessageBoxDefaultButton.Button1);

    try
    {
        if (result == DialogResult.Yes)
        {
            //---URL To unsubscribe---
            string urlToUnsubscribe = CurrentSelectedNode.Tag.ToString();

            //---load all the feeds from feeds list---
            TextReader textreader = File.OpenText(feedsList);
            string[] feeds = textreader.ReadToEnd().Split('|');
            textreader.Close();

            //---rewrite the feeds list omitting the one to be
            // unsubscribed---
            TextWriter textwriter = File.CreateText(feedsList);
            for (int i = 0; i <= feeds.Length - 2; i++)
            {
                if (feeds[i] != urlToUnsubscribe)
                {
                    textwriter.Write(feeds[i] + "|");
                }
            }
            textwriter.Close();

            //---remove the node from the TreeView control---
            CurrentSelectedNode.Remove();
            MessageBox.Show("Feed unsubscribed!");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

When the user needs to refresh a feed, first make a backup copy of the feed XML document and proceed to subscribe to the same feed again. If the subscription is successful, remove the node containing the old feed. If the subscription is not successful (for example, when a device is disconnected from the Internet), restore the backup feed XML document. This is handled by the `Refresh Feed MenuItem` control:

```
//---refresh the current feed---
private void mnuRefreshFeed_Click(object sender, EventArgs e)
{
    //---if no Internet connectivity---
    if (!IsConnected())
    {
        MessageBox.Show("You are not connected to the Internet.");
        return;
    }

    //---get the node to be refreshed---
    TreeNode CurrentSelectedNode = TreeView1.SelectedNode;
    string url = CurrentSelectedNode.Tag.ToString();

    //---get the filename of the feed---
    string FileName =
       appPath + @"\" + RemoveSpecialChars(url) + ".xml";

    try
    {
        //---make a backup copy of the current feed---
        File.Copy(FileName, FileName + "_Copy", true);

        //---delete feed from local storage---
        File.Delete(FileName);

        //---load the same feed again---
        if (SubscribeFeed(url))
        {
            //---remove the node to be refreshed---
            CurrentSelectedNode.Remove();
        }
        else //---the subscription(refresh) failed---
        {
            //---restore the deleted feed file---
            File.Copy(FileName + "_Copy", FileName, true);
            MessageBox.Show("Refresh not successful. Please try again.");
        }

        //---delete the backup file---
        File.Delete(FileName + "_Copy");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Refresh failed (" + ex.Message + ")");
    }
}
```

In the `Click` event handler for the `Collapse All Feeds MenuItem` control, use the `CollapseAll()` method from the `TreeView` control to collapse all the nodes:

```
private void mnuCollapseAllFeeds_Click(object sender, EventArgs e)
{
    TreeView1.CollapseAll();
}
```

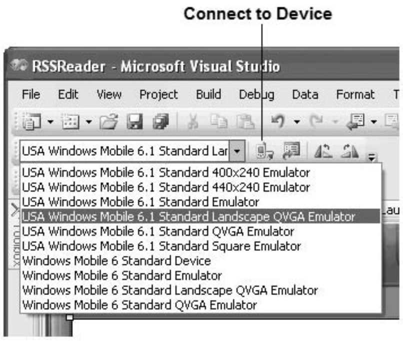
Finally, code the `Click` event handler in the `Back MenuItem` control in `Form2` as follows:

```
private void mnuBack_Click(object sender, EventArgs e)
{
    this.Hide();
}
```

That's it! You are now ready to test the application.

Testing Using Emulators

The SDKs for the various platforms include various emulators for you to test your Windows Mobile applications without needing to use a real device. For example, if your project is targeting the Windows Mobile 6 platform, you would see a list of emulators available for your testing (see Figure 18-17).


Connect to Device
RSSReader - Microsoft Visual Studio
File
Edit
View
Project
Build
Debug
Data
Format
USA Windows Mobile 6.1 Standard Lar

USA Windows Mobile 6.1 Standard 400x240 Emulator
USA Windows Mobile 6.1 Standard 440x240 Emulator
USA Windows Mobile 6.1 Standard Emulator
USA Windows Mobile 6.1 Standard Landscape QVGA Emulator
USA Windows Mobile 6.1 Standard QVGA Emulator
USA Windows Mobile 6.1 Standard Square Emulator
Windows Mobile 6 Standard Device
Windows Mobile 6 Standard Emulator
Windows Mobile 6 Standard Landscape QVGA Emulator
Windows Mobile 6 Standard QVGA Emulator

Figure 18-17

Once you have selected an emulator to use, click the Connect to Device button to launch it. To test your application, cradle the emulator to ActiveSync first so that you have Internet connectivity on the emulator. To cradle the emulator to ActiveSync, select Tools ⇨ Device Emulator Manager in Visual Studio 2008; right-click the emulator that has been launched (the one with the green arrow next to it); and select Cradle (see Figure 18-18).

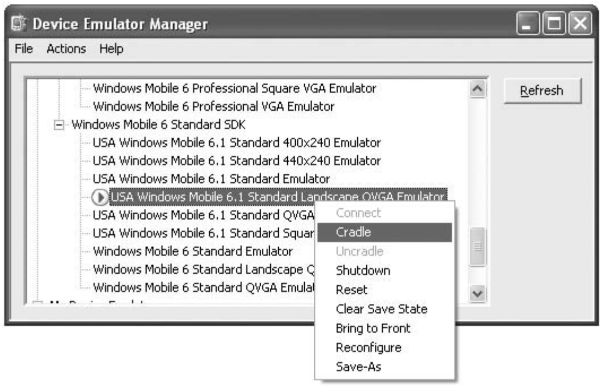

Device Emulator Manager
File Actions Help
Windows Mobile 6 Professional Square VGA Emulator
Windows Mobile 6 Professional VGA Emulator
Windows Mobile 6 Standard SDK
USA Windows Mobile 6.1 Standard 400x240 Emulator
USA Windows Mobile 6.1 Standard 440x240 Emulator
USA Windows Mobile 6.1 Standard Emulator
USA Windows Mobile 6.1 Standard Landscape QVGA Emulator
USA Windows Mobile 6.1 Standard QVGA
USA Windows Mobile 6.1 Standard Squar
Windows Mobile 6 Standard Emulator
Windows Mobile 6 Standard Landscape Q
Windows Mobile 6 Standard QVGA Emulat
Refresh
Connect
Cradle
Uncradle
Shutdown
Reset
Clear Save State
Bring to Front
Reconfigure
Save-As

Figure 18-18

Now press F5 in Visual Studio 2008 to deploy the application onto the emulator for testing.

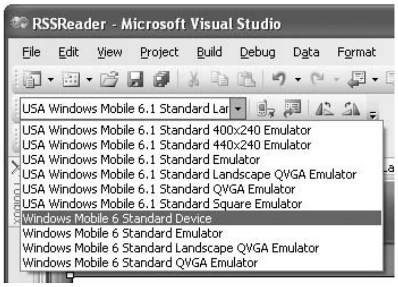
Testing Using Real Devices

While most of the testing can be performed on the emulators, it is always helpful to use a real device to fully test your application. For example, you will find out the true usability of your application when users have to type using the small keypad on the phone (versus typing using a keyboard when testing on an emulator). For this purpose, you can test your application on some of the devices running the Windows Mobile 6 Standard platform, such as the Samsung Black II (see Figure 18-19).

Figure 18-19

Testing your Windows Mobile application on real devices could not be easier. All you need is to:

1. Connect your device to your development machine using ActiveSync.
2. Select Windows Mobile 6 Standard Device (see Figure 18-20) in Visual Studio 2008.


RSSReader - Microsoft Visual Studio
File Edit View Project Build Debug Data Format
USA Windows Mobile 6.1 Standard Lar

USA Windows Mobile 6.1 Standard 400x240 Emulator
USA Windows Mobile 6.1 Standard 440x240 Emulator
USA Windows Mobile 6.1 Standard Emulator
USA Windows Mobile 6.1 Standard Landscape QVGA Emulator
USA Windows Mobile 6.1 Standard QVGA Emulator
USA Windows Mobile 6.1 Standard Square Emulator
Windows Mobile 6 Standard Device
Windows Mobile 6 Standard Emulator
Windows Mobile 6 Standard Landscape QVGA Emulator
Windows Mobile 6 Standard QVGA Emulator

Figure 18-20

3. Press F5.

The application is now deployed onto the device.

Deploying the Application

Once the testing and debugging process is over, you need to package the application nicely so that you have a way to get it installed on your users' devices.

The following sections show how to create a CAB (cabinet) file — a library of compressed files stored as a single file — so that you can easily distribute your application. Subsequent sections explain how to create an MSI (Microsoft Installer) file to automate the installation process.

Creating a CAB File

An easy way to package your Windows Mobile application is to create a CAB file so that you can transfer it onto the end user's device (using emails, web browser, memory card, and so on). The following steps show you how:

1. Add a new project to the current solution in Visual Studio 2008 (see Figure 18-21).

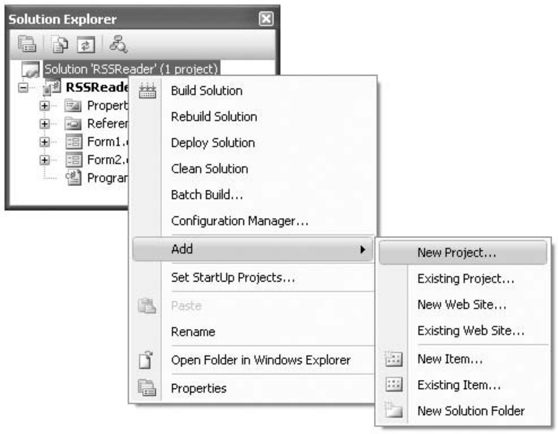
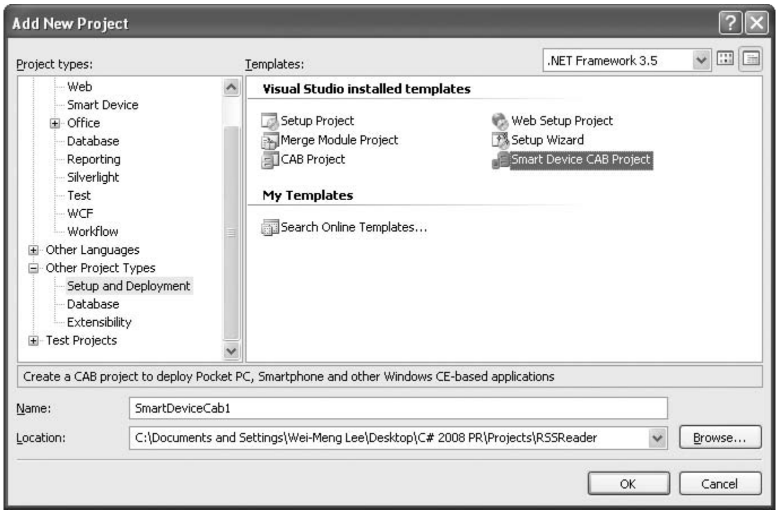


Figure 18-21

2. Choose the Setup and Deployment project type, and select the Smart Device CAB Project template (see Figure 18-22). Use the default name of SmartDeviceCab1, and click OK.



Create a CAB project to deploy Pocket PC, Smartphone and other Windows CE-based applications

Name: SmartDeviceCab1

Location: C:\Documents and Settings\Wei-Meng Lee\Desktop\C# 2008 PR\Projects\RSSReader

Figure 18-22

3. In the File System tab, right-click on Application Folder, and select Add ➪ Project Output (see Figure 18-23).

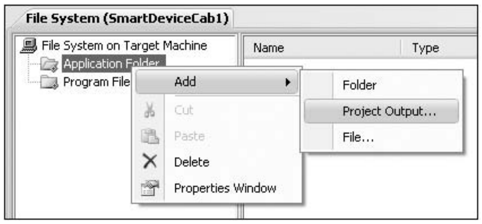


Figure 18-23

4. Select the `RSSReader` project, and click Primary output (see Figure 18-24). Click OK. This adds the output of the `RSSReader` project (which is your executable application) to the current project.

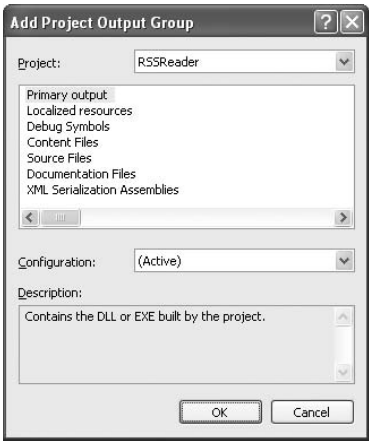


Figure 18-24

5. Right-click on the output item shown on the right-side of the File System tab, and create a shortcut to it (see Figure 18-25). Name the shortcut `RSSReader`.

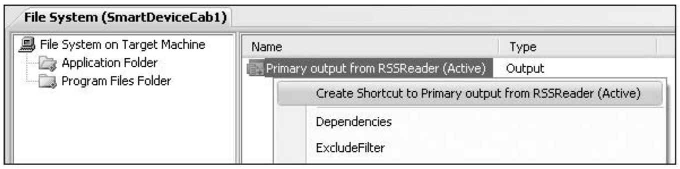

File System (SmartDeviceCab1)
File System on Target Machine
Application Folder
Program Files Folder
Name
Type
Primary output from RSSReader (Active)
Output
Create Shortcut to Primary output from RSSReader (Active)
Dependencies
ExcludeFilter

Figure 18-25

6. Right-click the File System on Target Machine item, and select Add Special Folder ➪ Start Menu Folder (see Figure 18-26).

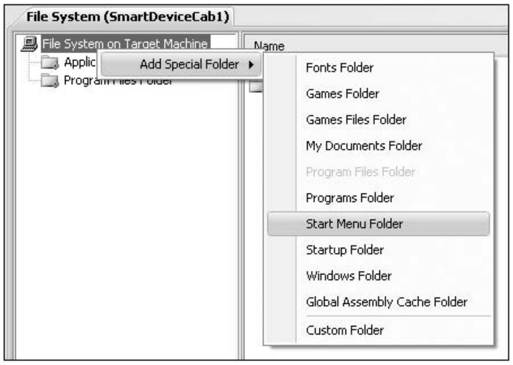


Figure 18-26

7. Drag and drop the RSSReader shortcut onto the newly added Start Menu Folder (see Figure 18-27). This ensures that when the CAB file is installed on the device, a shortcut named RSS Reader appears in the Start menu.

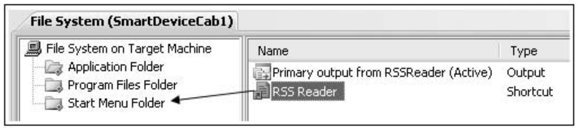
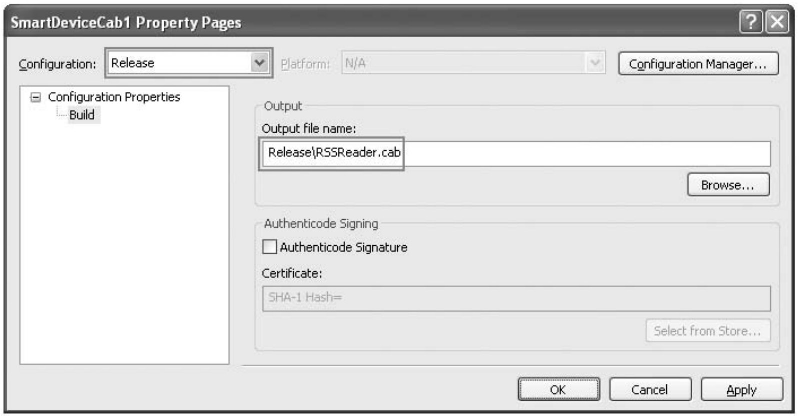

File System (SmartDeviceCab1)
File System on Target Machine
Application Folder
Program Files Folder
Start Menu Folder
Name
Type
Primary output from RSSReader (Active)
Output
RSS Reader
Shortcut

Figure 18-27

8. Right-click on the SmartDeviceCab1 project name in Solution Explorer, and select Properties. Change the Configuration from `Debug` to `Release`. Also, name the output file `Release\RSSReader.cab` (see Figure 18-28).


SmartDeviceCab1 Property Pages

Configuration: Release
Platform: N/A
Configuration Properties
Build
Output
Output file name: Release\RSSReader.cab
Authenticode Signing
Authenticode Signature
Certificate: SHA-1 Hash=

Figure 18-28

9. In Visual Studio 2008, change the configuration from `Debug` to `Release` (see Figure 18-29).

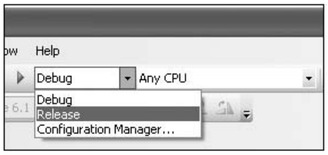
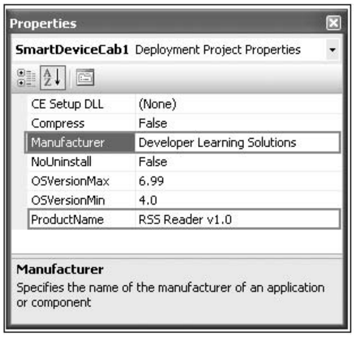


Figure 18-29

10. Finally, set the properties of the `SmartDeviceCab1` project as shown in the following table (see Figure 18-30).

Property	Value
Manufacturer	Developer Learning Solutions
ProductName	RSS Reader v1.0

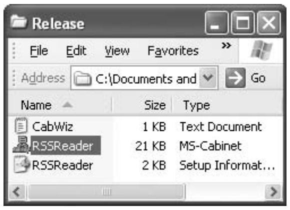


CE Setup DLL	(None)
Compress	False
Manufacturer	Developer Learning Solutions
NoUninstall	False
OSVersionMax	6.99
OSVersionMin	4.0
ProductName	RSS Reader v1.0

Manufacturer
Specifies the name of the manufacturer of an application or component

Figure 18-30

That's it! Right-click on the `SmartDeviceCab1` project name in Solution Explorer and select Build. You can find the CAB file located in the `\Release` folder of the `SmartDeviceCab1` project (see Figure 18-31).



Name	Size	Type
CabWiz	1 KB	Text Document
RSSReader	21 KB	MS-Cabinet
RSSReader	2 KB	Setup Informat...

Figure 18-31

Now you can distribute the CAB file to your customers using various media such as FTP, web hosting, email, and so on. When the user clicks on the RSSReader CAB file in File Explorer (on the device; see Figure 18-32), the application will ask if he wants to install it onto the device, or onto the storage card (if available).

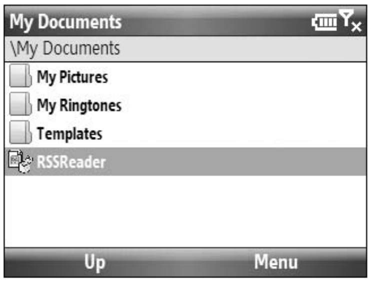


Figure 18-32

When the application is installed, the RSS Reader shortcut is in the Start menu (see Figure 18-33).

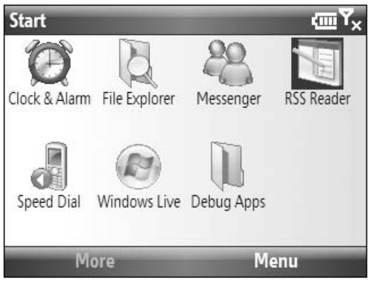


Figure 18-33

Creating a Setup Application

Although you can deploy CAB files directly to your users, you might want to use a more user-friendly way using the traditional setup application that most Windows users are familiar with — users simply connect their devices to their computers and then run a setup application, which then installs the application automatically on their devices through ActiveSync.

Creating a setup application for a Windows Mobile application is more involved than for a conventional Windows application because you have to activate ActiveSync to install it. Figure 18-34 shows the steps in the installation process.

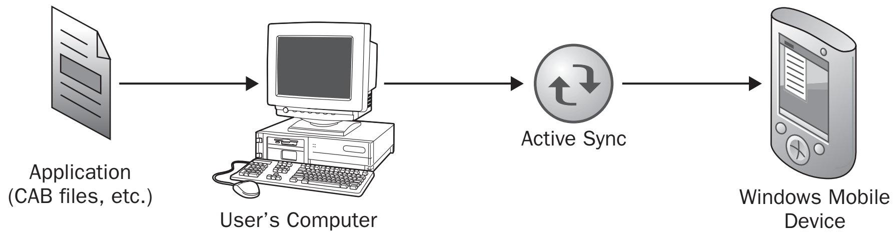


Figure 18-34

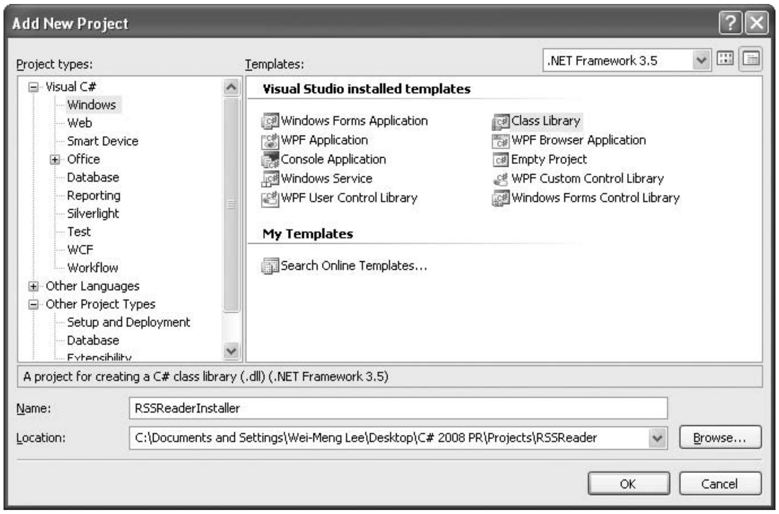
First, the application containing the CAB files (and other relevant files) must be installed on the user's computer. Then ActiveSync needs to install the application onto the user's device.

The following sections detail how to create an MSI file to install the application onto the user's computer and then onto the device.

Creating the Custom Installer

The first component you will build is the custom installer that will invoke ActiveSync to install the application onto the user's device. For this, you will use a Class Library project.

Add a new project to your current solution by going to File ➪ Add ➪ New Project. Select the Windows project type and select the Class Library template. Name the project `RSSReaderInstaller` (see Figure 18-35). Click OK.



A project for creating a C# class library (.dll) (.NET Framework 3.5)

Name: RSSReaderInstaller

Location: C:\Documents and Settings\Wei-Meng Lee\Desktop\C# 2008 PR\Projects\RSSReader

Figure 18-35

Delete the default `Class1.cs` file and add a new item to the project. In the Add New Item dialog, select the Installer Class template, and name the file `RSSReaderInstaller.cs` (see Figure 18-36).

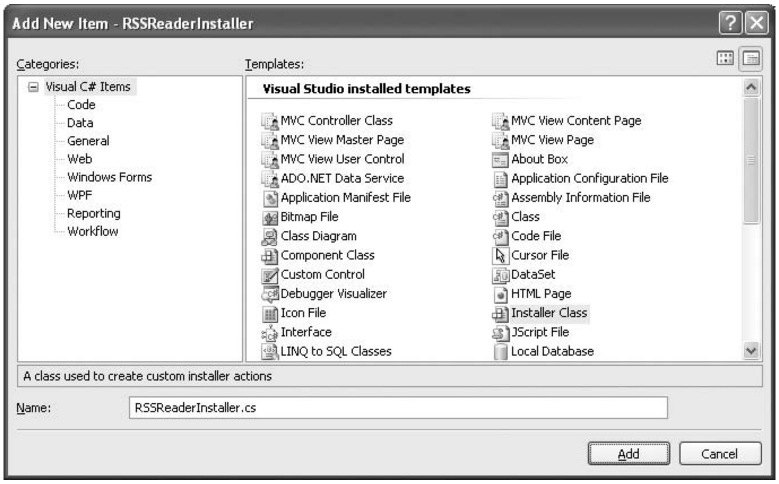


Figure 18-36

Add two references to the project: `System.Configuration.Install` and `System.Windows.Forms` (see Figure 18-37).

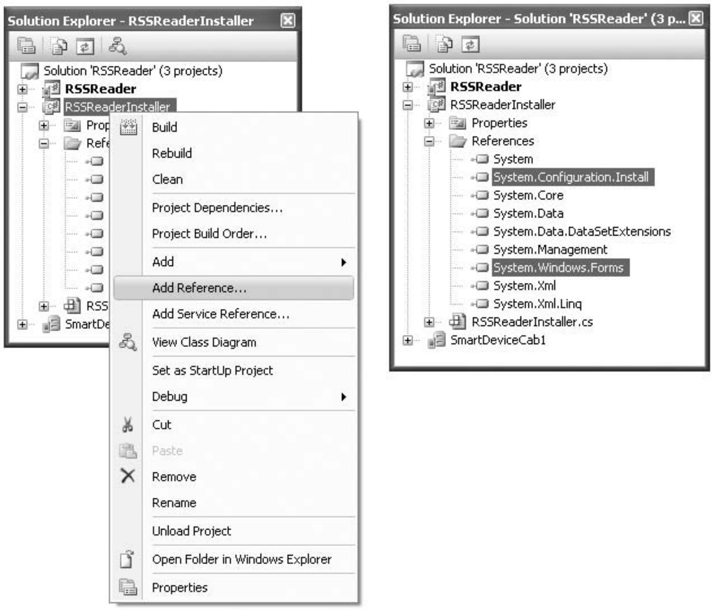


Figure 18-37

Switch to the code view of the `RSSReaderInstaller.cs` file and import the following namespaces:

```
using Microsoft.Win32;
using System.IO;
using System.Diagnostics;
using System.Windows.Forms;
```

Within the `RSSReaderInstaller` class, define the `INI_FILE` constant. This constant holds the name of the `.ini` file that will be used by ActiveSync for installing the CAB file onto the target device.

```
namespace RSSReaderInstaller
{
    [RunInstaller(true)]
    public partial class RSSReaderInstaller : Installer
    {

        const string INI_FILE = @"setup.ini";
```

In the constructor of the `RSSReaderInstaller` class, wire the `AfterInstall` and `Uninstall` events to their corresponding event handlers:

```
        public RSSReaderInstaller()
        {
            InitializeComponent();

            this.AfterInstall += new
                InstallEventHandler(RSSReaderInstaller_AfterInstall);
            this.AfterUninstall += new
                InstallEventHandler(RSSReaderInstaller_AfterUninstall);

        }

        void RSSReaderInstaller_AfterInstall(object sender, InstallEventArgs e)
        {
        }

        void RSSReaderInstaller_AfterUninstall(object sender, InstallEventArgs e)
        {
        }
```

The `AfterInstall` event is fired when the application (CAB file) has been installed onto the user's computer. Similarly, the `AfterUninstall` event fires when the application has been uninstalled from the user's computer.

When the application is installed on the user's computer, you use Windows CE Application Manager (`CEAPPMGR.EXE`) to install the application onto the user's device.

The Windows CE Application Manager is installed automatically when you install ActiveSync on your computer.

To locate the Windows CE Application Manager, define the following function named `GetWindowsCeApplicationManager()`:

```
private string GetWindowsCeApplicationManager()
{
    //---check if the Windows CE Application Manager is installed---
    string ceAppPath = KeyExists();
    if (ceAppPath == String.Empty)
    {
        MessageBox.Show("Windows CE App Manager not installed",
        "Setup", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        return String.Empty;
    }
    else
        return ceAppPath;
}
```

This function locates the Windows CE Application Manager by checking the registry of the computer using the `KeyExists()` function, which is defined as follows:

```
private string KeyExists()
{
    //---get the path to the Windows CE App Manager from the registry---
    RegistryKey key =
      Registry.LocalMachine.OpenSubKey(
      @"SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\CEAPPMGR.EXE");
    if (key == null)
        return String.Empty;
    else
        return key.GetValue(String.Empty, String.Empty).ToString();
}
```

The location of the Windows CE Application Manager can be obtained via the registry key: `"SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\CEAPPMGR.EXE"`, so querying the value of this key provides the location of this application.

The next function to define is `GetIniPath()`, which returns the location of the `.ini` file that is needed by the Windows CE Application Manager:

```
private string GetIniPath()
{
    //---get the path of the .ini file---
    return "\"" +
       Path.Combine(Path.GetDirectoryName(
       System.Reflection.Assembly.
       GetExecutingAssembly().Location), INI_FILE) + "\"";
}
```

By default, the `.ini` file is saved in the same location as the application (you will learn how to accomplish this in the next section). The `GetIniPath()` function uses reflection to find the location of the custom installer, and then return the path of the `.ini` file as a string, enclosed by a pair of double quotation marks (the Windows CE Application requires the path of the `.ini` file to be enclosed by a pair of double quotation marks).

Finally, you can now code the `AfterInstall` event handler, like this:

```
void RSSReaderInstaller_AfterInstall(object sender, InstallEventArgs e)
{
    //---to be executed when the application is installed---
    string ceAppPath = GetWindowsCeApplicationManager();
    if (ceAppPath == String.Empty)
        return;
    Process.Start(ceAppPath, GetIniPath());
}
```

Here, you get the location of the Windows CE Application Manager and then use the `Process.Start()` method to invoke the Windows CE Application Manager, passing it the path of the `.ini` file.

Likewise, when the application has been uninstalled, you simply invoke the Windows CE Application Manager and let the user choose the application to remove from the device. This is done in the `AfterUninstall` event handler:

```
void RSSReaderInstaller_AfterUninstall(object sender, InstallEventArgs e)
{
    //---to be executed when the application is uninstalled---
    string ceAppPath = GetWindowsCeApplicationManager();
    if (ceAppPath == String.Empty)
        return;
    Process.Start(ceAppPath, String.Empty);
}
```

The last step in this section is to add the `setup.ini` file that the Windows CE Application Manager needs to install the application onto the device. Add a text file to the project and name it `setup.ini`. Populate the file with the following:

```
[CEAppManager]
Version      = 1.0
Component    = RSSReader

[RSSReader]
Description  = RSSReader Application
Uninstall    = RSSReader
CabFiles     = RSSReader.cab
```

For more information about the various components in an `.ini` *file, refer to the documentation at* `http://msdn.microsoft.com/en-us/library/ms889558.aspx`.

To build the project, right-click on RSSReaderInstaller in Solution Explorer and select Build.

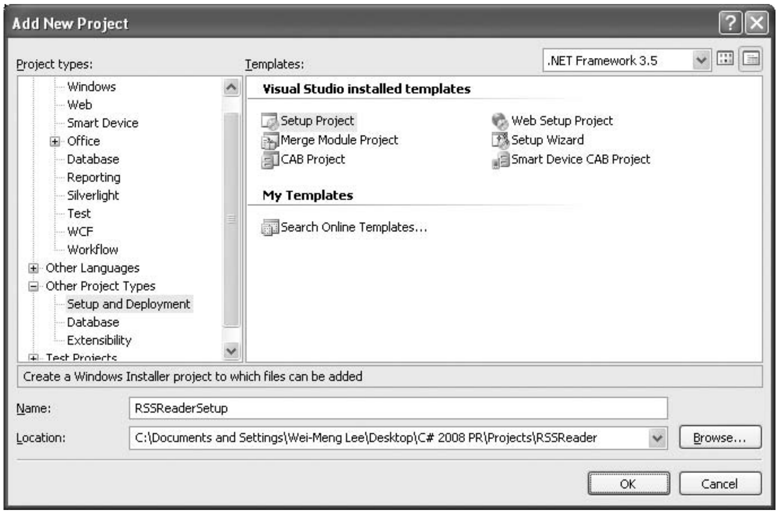
Set the `SmartDeviceCab1` project's properties as shown in the following table.

Property	Value
Manufacturer	Developer Learning Solutions
ProductName	RSS Reader v1.0

Creating a MSI File

You can now create the MSI installer to install the application onto the user's computer and then invoke the custom installer built in the previous section to instruct the Windows CE Application Manager to install the application onto the device.

Using the same solution, add a new Setup Project (see Figure 18-38). Name the project `RSSReaderSetup`.



Create a Windows Installer project to which files can be added

Name: RSSReaderSetup

Location: C:\Documents and Settings\Wei-Meng Lee\Desktop\C# 2008 PR\Projects\RSSReader

Figure 18-38

Using the newly created project, you can now add the various components and files that you have been building in the past few sections. Right-click on the `RSSReaderSetup` project in Solution Explorer, and select Add ⇨ File (see Figure 18-39).

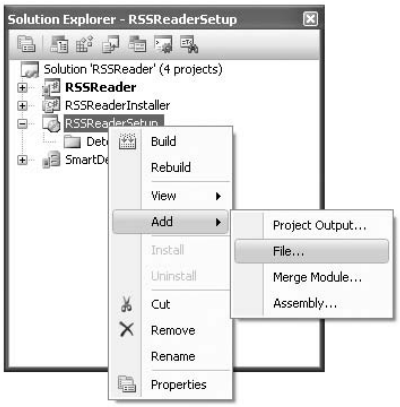


Figure 18-39

Add the following files (see Figure 18-40):

- `SmartDeviceCab1\Release\RSSReader.CAB`
- `RSSReaderInstaller\bin\Release\RSSReaderInstaller.dll`
- `RSSReaderInstaller\setup.ini`

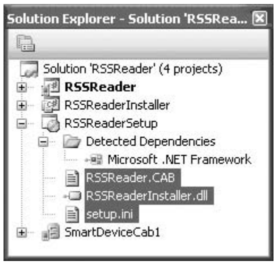


Figure 18-40

These three files will be copied to the user's computer during the installation.

The next step is to configure the MSI installer to perform some custom actions during the installation stage. Right-click the `RSSReaderSetup` project in Solution Explorer, and select View ⇨ Custom Actions (see Figure 18-41).

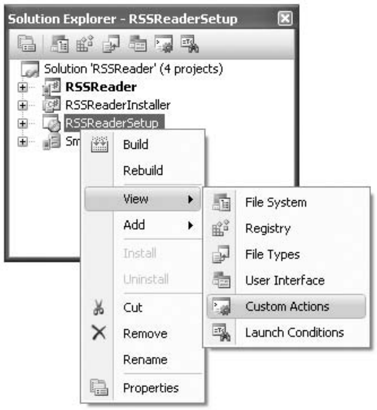


Figure 18-41

The Custom Actions tab displays. Right-click on Custom Actions, and select Add Custom Action (see Figure 18-42).

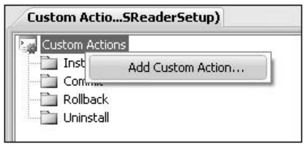


Figure 18-42

Select Application Folder, select the `RSSReaderInstall.dll` file (see Figure 18-43), and click OK.

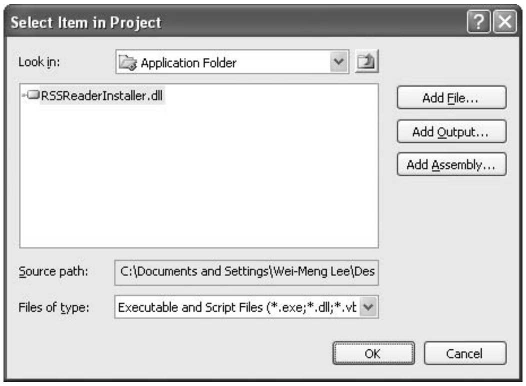


Figure 18-43

The Custom Actions tab should now look like Figure 18-44.

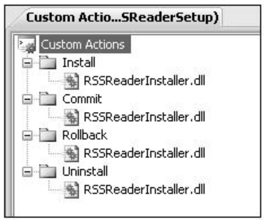
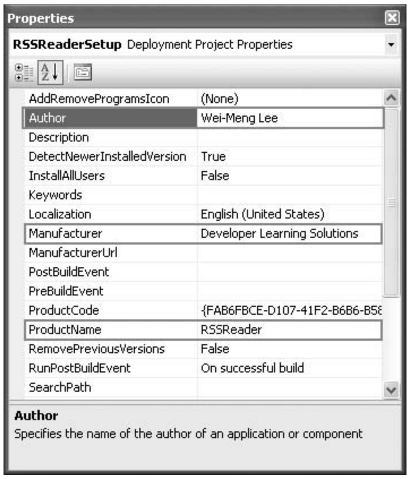


Figure 18-44

Set the various properties of the `RSSReaderSetup` project as shown in the following table (see Figure 18-45).

Property	Value
Author	Wei-Meng Lee
Manufacturer	Developer Learning Solutions
ProductName	RSSReader



Properties

RSSReaderSetup Deployment Project Properties

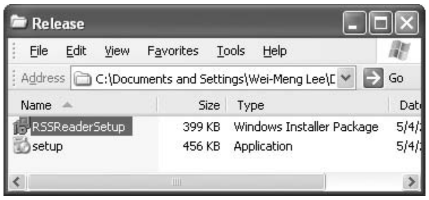
AddRemoveProgramsIcon	(None)
Author	Wei-Meng Lee
Description	
DetectNewerInstalledVersion	True
InstallAllUsers	False
Keywords	
Localization	English (United States)
Manufacturer	Developer Learning Solutions
ManufacturerUrl	
PostBuildEvent	
PreBuildEvent	
ProductCode	{FAB6FBCE-D107-41F2-B6B6-B58
ProductName	RSSReader
RemovePreviousVersions	False
RunPostBuildEvent	On successful build
SearchPath	

Author
Specifies the name of the author of an application or component

Figure 18-45

The last step is to build the project. Right-click on the `RSSReaderSetup` project in Solution Explorer, and select Build.

The MSI installer is now in the `\Release` subfolder of the folder containing the `RSSReaderSetup` project (see Figure 18-46).



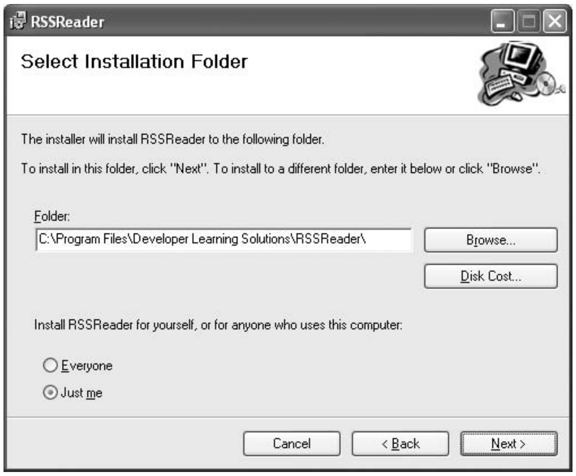
Release
File Edit View Favorites Tools Help
Address C:\Documents and Settings\Wei-Meng Lee\D
Go

Name	Size	Type	Dat
RSSReaderSetup	399 KB	Windows Installer Package	5/4/
setup	456 KB	Application	5/4/

Figure 18-46

Testing the Setup

To test the MSI installer, ensure that your emulator (or real device) is connected to ActiveSync. Double-click the `RSSReaderSeup.msi` application, and the installation process begins (see Figure 18-47).



The installer will install RSSReader to the following folder.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

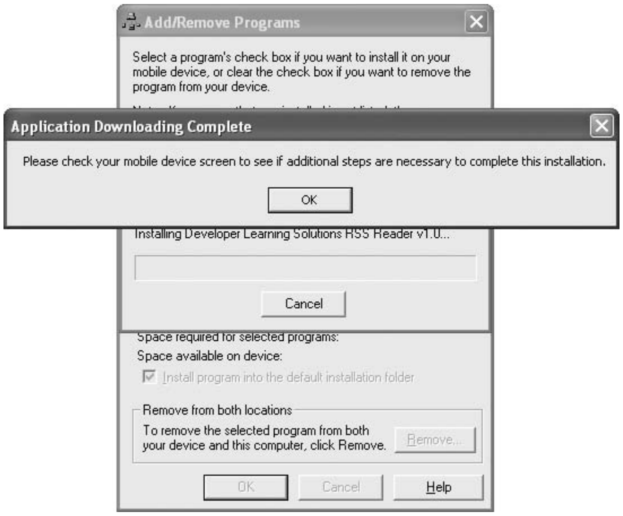
Folder: C:\Program Files\Developer Learning Solutions\RSSReader\

Install RSSReader for yourself, or for anyone who uses this computer:

( ) Everyone
(•) Just me

Figure 18-47

Follow the instructions on the dialog. At the end, an Application Downloading Complete message displays (see Figure 18-48).



Select a program's check box if you want to install it on your mobile device, or clear the check box if you want to remove the program from your device.

Please check your mobile device screen to see if additional steps are necessary to complete this installation.

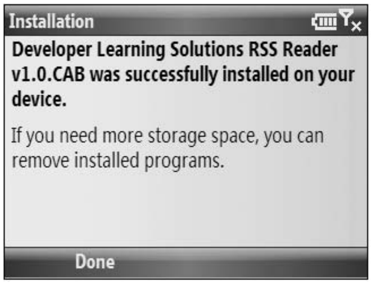
Installing Developer Learning Solutions RSS Reader v1.0...

Space required for selected programs:
Space available on device:
☑ Install program into the default installation folder

Remove from both locations
To remove the selected program from both your device and this computer, click Remove.

Figure 18-48

Check your emulator (or real device) to verify that the application is successfully installed (see Figure 18-49).

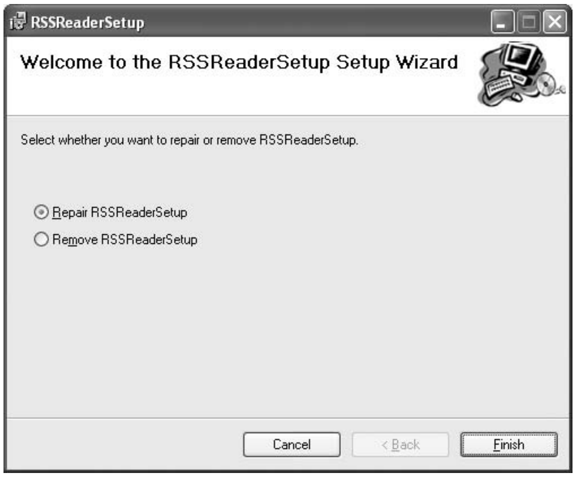

Installation

Developer Learning Solutions RSS Reader v1.0.CAB was successfully installed on your device.

If you need more storage space, you can remove installed programs.

Figure 18-49

To uninstall the application, double-click the `RSSReaderSeup.msi` application again. This time, you see the dialog shown in Figure 18-50.


RSSReaderSetup

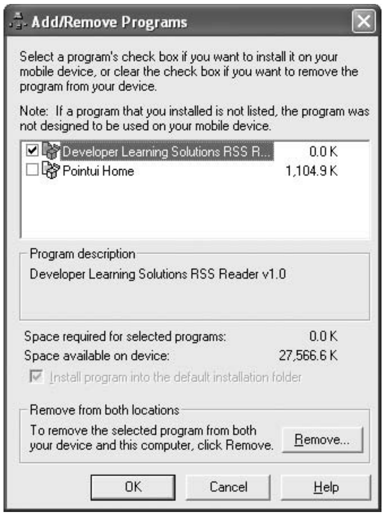
Welcome to the RSSReaderSetup Setup Wizard

Select whether you want to repair or remove RSSReaderSetup.

[✓ Repair RSSReaderSetup]
Remove RSSReaderSetup

Figure 18-50

If you choose to remove the application, the Windows CE Application Manager displays the list of programs that you have installed through ActiveSync (see Figure 18-51). To uninstall the RSS Reader application, uncheck the application and click OK. The application is removed.



Select a program's check box if you want to install it on your mobile device, or clear the check box if you want to remove the program from your device.

Note: If a program that you installed is not listed, the program was not designed to be used on your mobile device.

☑	Developer Learning Solutions RSS R...	0.0 K
☐	Pointui Home	1,104.9 K

Program description

Developer Learning Solutions RSS Reader v1.0

Space required for selected programs: 0.0 K

Space available on device: 27,566.6 K

☑ Install program into the default installation folder

Remove from both locations

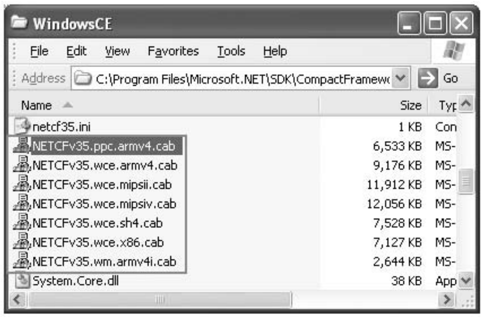
To remove the selected program from both your device and this computer, click Remove.

Figure 18-51

Installing the Prerequisites — .NET Compact Framework 3.5

One problem you will likely face when deploying your application to real devices is that the target device does not have the required version of the .NET Compact Framework (version 3.5 is needed). Hence, you need to ensure that the user has a means to install the right version of the .NET Compact Framework. There are two ways of doing this:

- Distribute a copy of the .NET Compact Framework 3.5 Redistributable to your client. You can download a copy from `http://microsoft.com/downloads`. Users can install the .NET Compact Framework before or after installing your application. This is the easiest approach, but requires the user to perform the extra step of installing the .NET Compact Framework.
- Programmatically install the .NET Compact Framework during installation, using the custom installer. Earlier, you saw how you can invoke the Windows CE Application Manager from within the custom installer class by using the `.ini` file. In this case, you simply need to create another `.ini` file, this time to install the CAB file containing the .NET Compact Framework. The various CAB files for the .NET Compact Framework 3.5 can be found on your local drive in the following directory: `C:\Program Files\Microsoft.NET\SDK\CompactFramework\v3.5\WindowsCE`. Figure 18-52 shows the various CAB files for each processor type (ARM, MIPS, SH4, X86, and so on). To install the .NET Compact Framework 3.5 on Windows Mobile 6 Standard devices, you just need to add the `NETCFv35.wm.armv4i.cab` file to the `RSSReaderInstaller` project, together with its associated `.ini` file.


WindowsCE
File Edit View Favorites Tools Help
Address C:\Program Files\Microsoft.NET\SDK\CompactFramew
Go

Name	Size	Typ
netcf35.ini	1 KB	Con
NETCFv35.ppc.armv4.cab	6,533 KB	MS-
NETCFv35.wce.armv4.cab	9,176 KB	MS-
NETCFv35.wce.mipsii.cab	11,912 KB	MS-
NETCFv35.wce.mipsiv.cab	12,056 KB	MS-
NETCFv35.wce.sh4.cab	7,528 KB	MS-
NETCFv35.wce.x86.cab	7,127 KB	MS-
NETCFv35.wm.armv4i.cab	2,644 KB	MS-
System.Core.dll	38 KB	App

Figure 18-52

Summary

This chapter explored developing applications for the Windows Mobile 6 platform, using the .NET Compact Framework. Using that framework, you can leverage your familiarity with the .NET Framework to develop compelling mobile applications. The RSS application is an example of a useful application that you can use on a daily basis. The chapter also explained how to package an application into a CAB file and then into a MSI package so that you can distribute it to your users easily.