

# 28

## Mobile Development

We are entering an era of mobile applications. Mobile devices are getting better, faster, and cheaper every year. The bandwidth on these devices has improved significantly over the last few years and will continue to improve by leaps and bounds. Businesses are continually finding newer and more exciting ways to provide applications on mobile devices. If you haven't yet had the opportunity to program mobile Web applications, you can be sure that your time will come sooner than you might expect.

This book wouldn't be complete without a chapter showing you how to create mobile Web applications using ASP.NET 2.0. This chapter starts with the basics of mobile Web application development and goes deep enough to make you feel comfortable about starting your next mobile Web development project. The chapter first discusses how Visual Studio helps with the creation of mobile Web applications. It then shows you all available mobile Web controls and teaches you the appropriate ways of using them in your applications. Finally, you learn how to develop device-specific properties and manage ViewState and Sessions.

### Creating a NEW ASP.NET Mobile Web Application

Visual Studio provides very powerful and user-friendly tools for creating mobile Web applications. If you are familiar with how to create a typical ASP.NET application with Visual Studio, you already know a lot about creating mobile Web applications. You simply create a Web site project and add a Mobile Web Form to your project. When you open this form in the Designer window, you see a set of mobile controls inside the Toolbox.

Follow these steps to create a new Visual Basic or Visual C# mobile Web application using Visual Studio 2005:

## Chapter 28

1. Choose File→New→Web Site.
2. From the Visual Studio Installed Templates list, select ASP.NET Web Site.
3. Provide the location, language, and path, and click OK (see Figure 28-1).

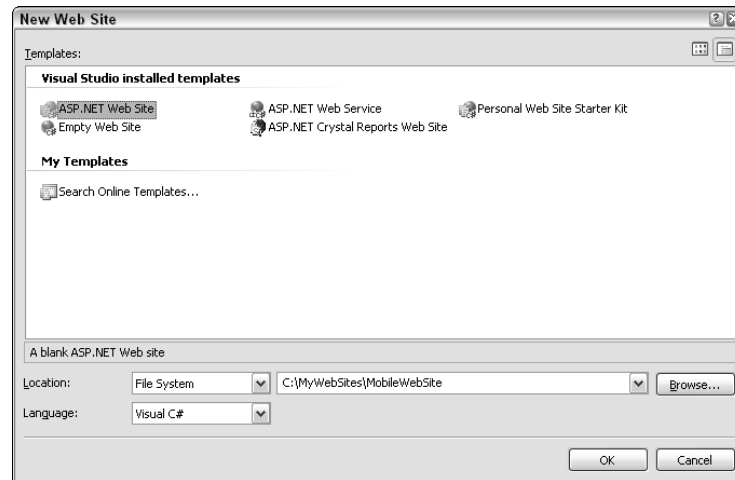


Figure 28-1

After you follow the preceding steps, you can see that Visual Studio creates a Web site project for you. Next, add a Mobile Web Form to your project by following these steps:

1. Right-click on the ASP.NET Project you just created in the Solution Explorer and select Add New Item.
2. In the Add New Item dialog, select Mobile Web Form from the Visual Studio Installed Templates list (see Figure 28-2).

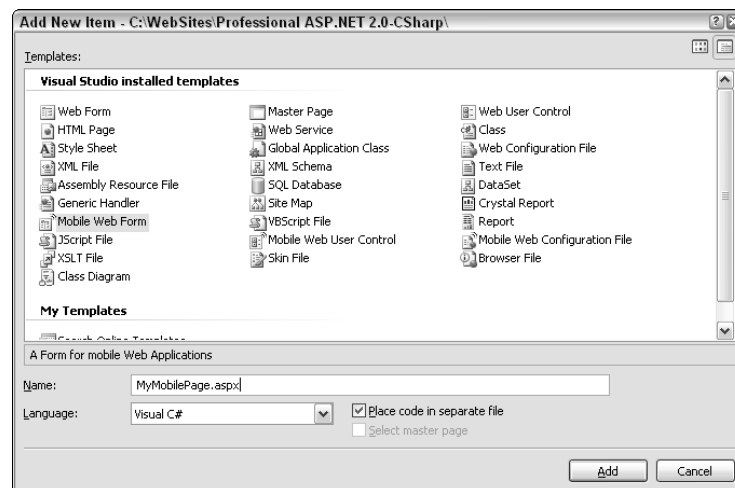


Figure 28-2

3. Provide a name for the Mobile Form, select a language, and click to check the Select Master Page check box if a master page is defined for your project. Be sure to check the Place Code in Separate File check box to use the code-behind model provided.
4. Click the Add button.

After you click the Add button, you can see that Visual Studio creates two files with the names `MyMobilePage.aspx` and `MyMobilePage.cs` (or `MyMobilePage.vb` if you selected Visual Basic). The `MyMobilePage.aspx` file contains the declarative format of the ASP.NET mobile controls. The `MyMobilePage.cs` file contains the code for handling events and performing other programmatic tasks.

After the Mobile Web Form is created, feel free to add controls from the Mobile Web Forms tab of the Toolbox. Just like other ASP.NET controls, the mobile controls provide properties and events that you can use to customize behaviors. You can also type these mobile controls directly into the Mobile Web Form source window by using the `<mobile: />` syntax shown here:

```
<mobile:TextBox ID="MyTextBox" runat="server"></mobile:TextBox>
<mobile:Label ID="MyLabel" runat="server">Label</mobile:Label>
```

Figure 28-3 shows a Mobile Web Form that contains Label, Text Box, and Command controls. This Mobile Web Form finds a customer record using the customer identifier provided in the text box.

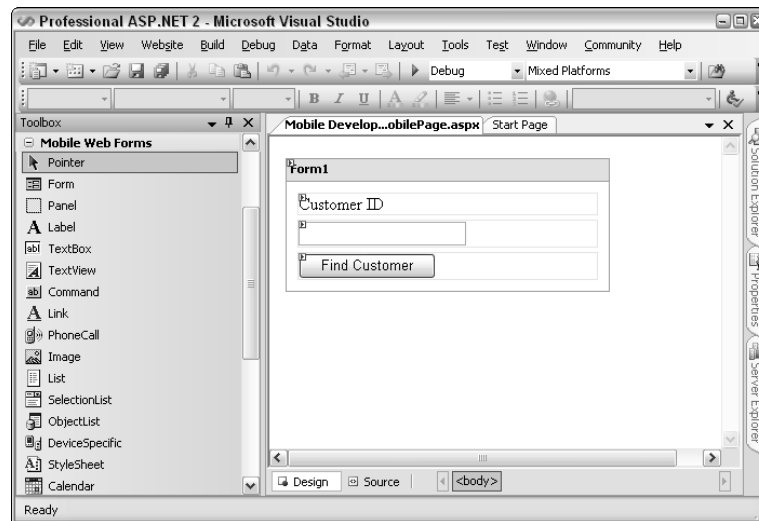


Figure 28-3

The HTML code generated by adding these controls is shown in Listing 28-1. You can see that all three mobile controls are reflected by the HTML tags that start with the `mobile:` prefix. Notice that the `mobile:` prefix is used with the form tag as well. In a typical ASP.NET page, you won't be required to treat forms as Web controls. However, the mobile Web page treats forms a little differently. The main difference is that the mobile Web page allows you to create multiple forms on one page and navigate among these forms without making a trip to the Web server. This flexibility allows you to reduce the number of roundtrips to the server, which is important because the bandwidth is typically slow on a mobile connection.

## Chapter 28

---

### Listing 28-1: The source code for a simple mobile Web page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="MyMobilePage.aspx.cs"
    Inherits="_Default" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <mobile:Label id="lblID" Runat="server">Customer ID</mobile:Label>
        <mobile:TextBox id="txtCustID" Runat="server"></mobile:TextBox>
        <mobile:Command id="cmdGetCustomer" Runat="server">
            Find Customer
        </mobile:Command>
    </mobile:Form>
</body>
</html>
```

## Views of an ASP.NET Mobile Web Form

The ASP.NET Mobile Web Forms Designer provides three views in the Microsoft Visual Studio environment: Design view, HTML view, and Code view. These views are panes in the Visual Studio main window that you can access using a variety of mechanisms.

### Design View

The Design view loads the Mobile Web Form and displays an automatic rendering of its controls using the default properties. You can add new controls by dragging and dropping them from the Toolbox. You can modify existing controls by using the Properties window or remove the controls by simply selecting them and pressing the Delete button on the keyboard.

The Design view is not a WYSIWYG editor, mainly because the actual appearance of a Mobile Web Form varies significantly from one device to another. For example, the Design view always displays one control per line, whereas some devices may be able to display multiple controls on the same line. You should also know that the ASP.NET Mobile Web Form does not support absolute positioning of Mobile Web Controls.

### HTML View

The HTML view displays the source HTML. You can edit it directly if you want to gain total control of the layout and rendering of the form. You don't have to work directly in the HTML view; it is available mostly as a convenience for those who would rather not depend on automatic HTML generation. Otherwise, the designer does a very good job of allowing you to work in a user-friendly design environment.

You can switch between the Design view and the HTML view by clicking the appropriate tab at the bottom of each view.

### Code View

The Code view manages the programming logic contained in the code-behind file. You can enter the Code view simply by right-clicking the mobile Web page in Solution Explorer and selecting the View Code menu option.

## Event Handling with Mobile Web Controls

Just like regular Web controls, Mobile Web Controls also fire events. They provide default events for handling most commonly occurring scenarios. In addition, each control also raises a number of non-default events that can be handled as needed. You already know that events are raised by certain activities performed by the user on the browser. The mobile browser behaves in the same manner. It responds to the event by sending a post back to the server where the event is processed. The resulting HTML is sent back to the browser. Handling a default event fired by a mobile control is as simple as placing the control on the page and double-clicking it. You are presented with a code window where the event handler is already prewired for you, as shown in the following code. As you will shortly see, if you are interested in handling non-default events, you simply view the list of available events in the Properties window and double-click the event you want to handle.

### VB

```
Private Sub cmdGetCustomer_Click(ByVal sender as System.Object, _  
    ByVal e as System.EventArgs) Handles Command1.Click  
  
End Sub
```

### C#

```
private void cmdGetCustomer_Click(object sender, System.EventArgs e)  
{  
  
}
```

Creating a non-default event handler is also quite easy. The steps are the same regardless of the programming language. If you have worked with previous versions of Visual Studio .NET, you may remember that wiring event handlers worked differently in Visual Basic .NET than in C#. The current version of Visual Studio unifies the steps in both languages.

Follow these steps to create a non-default event handler:

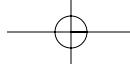
1. Select the desired control in Design view.
2. Click the Events button (the one with the lightning bolt) in the Properties window. Clicking this button shows a list of all available events for the selected control.
3. Double-click the event you want to handle. You are taken to the Code view with the event handler prewired for the selected event.

You have now created a non-default event handler.

## Using Control Containers

Two kinds of container controls are provided for mobile Web pages: the Form control and the Panel control. All mobile controls in ASP.NET exist inside one of these container controls. Other than using these container controls for grouping together Mobile Web Controls, you can also use them to apply styles consistently to all controls inside them. The container controls are added to the page with a default size that changes as new controls are added to it. You can't resize a container control to a specific size.

Mobile Web Forms only support sequential placement of controls because of the diversity among the wide range of mobile devices—especially WML devices that, for the most part, can't support



## Chapter 28

---

*side-by-side layout*—that is, having multiple controls reside next to each other sequentially. You can force ASP.NET to take advantage of side-by-side layout on devices that support it by setting the `BreakAfter` property of the mobile Web control to `False`. The ASP.NET Mobile Designer does not use the `BreakAfter` property. As a result, the Designer doesn't display controls side by side, even if `BreakAfter` is set to `False`.

The good news is that the ASP.NET Mobile Designer enables you to customize the appearance of Mobile Web Forms and controls for specific devices. This flexibility enables you to ensure that your application looks and functions as intended on the devices you specify. You can read more about customizing for specific devices a little later in this chapter in the section “Understanding Device Filters.”

### **The Form Control**

All content and all controls are contained inside a Form control. Every page is required to have at least one Form control. The page can contain multiple Form controls; however, it can display only one at a time. A default form is automatically created when you create a mobile Web page. You can add more forms by dragging and dropping them from the Toolbox.

When the page loads, it displays the first form placed inside it by default. You can write code in the `Page_Load` event to direct the user to a specific form if you want. You can also program to navigate users to other forms based on user input. Navigating between forms on the same page doesn't result in a trip to the Web server, thereby significantly improving your application's response time. Organizing pages into groups of forms also enables you to pass richer state information from one form to the next because all forms are contained inside the same physical page. All forms on a mobile page share the same code behind and, therefore, can share the same functions and member fields.

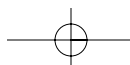
Because ASP.NET mobile Web applications usually run on devices with smaller screens, you might need to break a single ASP.NET Web page into several mobile Web pages so that each can fit on a smaller screen. Without the capability to put multiple logical groups of controls in mobile forms, you could have great difficulty maintaining a one-to-one mapping between ASP.NET Web Forms and ASP.NET Mobile Forms in the same application.

The real question to ask yourself is how to decide the appropriate groupings of forms on a mobile Web page. You should know that all forms on the page are instantiated when the page is loaded causing the page with many forms to take longer to load. However, navigating between these forms is speedier once all forms are loaded in the device's memory. Another advantage you get by using multiple forms on a page is persistence of the state information while you are switching the user from one form to another.

### **The Panel Control**

The Panel control provides an easy way to group related controls together while keeping them inside a Form control. You can easily apply styles to the panel to help you keep a consistent look-and-feel in your mobile applications. Another great benefit of panels is to keep related controls together on the same page because ASP.NET attempts to keep all controls in a panel on the screen at the same time.

You can show or hide a group of controls by keeping them inside a panel and making the panel either visible or invisible. You can optionally insert one panel inside another to create a composite group of controls.



Adding panels to your application is as simple as dragging and dropping them from the Toolbox. To add a Panel control, follow these steps:

1. Drag a Panel control from the Mobile Web Forms tab of the Toolbox.
2. Customize the Panel control by dragging other controls onto the panel from the Toolbox.

## Using StyleSheets

You can use styles to customize the appearance of controls when they are rendered. You can do so by using StyleSheet controls, defining style information, and applying it to one or more controls on the same page. As mentioned earlier, you can apply styles not only to a specific control but also to container controls that consistently apply the styles to all controls inside the container.

A StyleSheet control should be placed outside the container control. In fact, this is the only type of control that can exist outside a container. You can define only one StyleSheet control for each page or mobile user control. After adding a StyleSheet control to a Mobile Web Forms page, you can open the StyleSheet Styles Editor and Templating Options dialog boxes to define these properties.

To create, customize, and apply a StyleSheet control to a Mobile Web Form, follow these steps:

1. Drag and Drop a StyleSheet control on the Mobile Web Form.
2. Right-click the StyleSheet control you just put in place and select the Templating Options menu. The Templating Options dialog box appears, as shown in Figure 28-4. This window allows you to create and edit multiple styles and device filters. Device filters are discussed a little later in this chapter.



Figure 28-4

3. Click the Edit button next to the Style drop-down box. The Styles Editor dialog box opens, as shown in Figure 28-5. This dialog box enables you to create as many styles as you want.

## Chapter 28

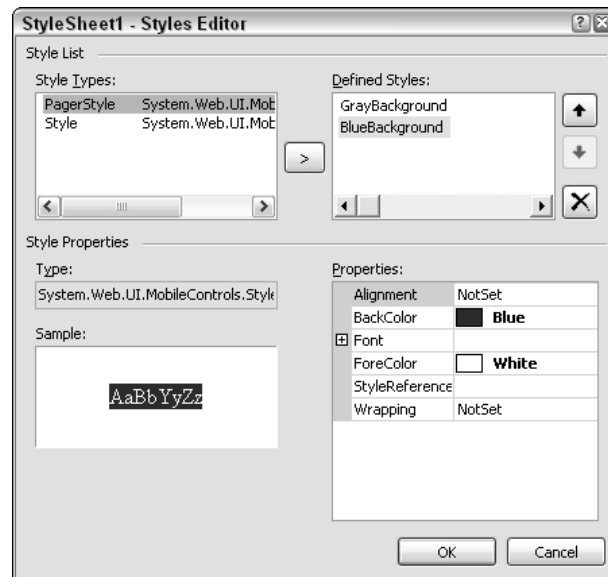


Figure 28-5

4. Click the desired style type from the left-hand list and click the > button. This action creates a new style in the right-hand list using the type you selected. You currently have two options in the Style Types list:
  - Pager Style type:** Provides styling elements for configuring pagination. This style is helpful if your Mobile Web Form has more controls than can fit on one screen. In this case, ASP.NET automatically creates pagination from viewable controls to others.
  - Style type:** Customizes styling for all mobile Web controls.
5. Right-click the styles shown in the Defined Styles list and select the Rename menu option. Rename these defined styles so that their names are more meaningful and easier to select from.
6. Click OK when you are done defining styles for your Mobile Web Form.

From Figure 28-5, you can see that your choice of style options is limited. Most of the restrictions are due to hardware limitations on mobile devices, especially WML-enabled phone devices. However, you still have a few good options to pick from. You can set background color, foreground color, alignment, font sizes, and font types. The good news is that the availability of `StyleSheet` controls makes it easy for you to apply these styles consistently throughout your mobile Web application.

After you have finished defining styles, the next step is to apply these styles to the mobile controls. You can do this by simply clicking the `StyleReference` property and selecting the desired style from the list. In Figure 28-6, the `GrayBackground` style has been applied to the `Form` control and the `BlueBackground` style to the `Label` control. You already know that applying a style to a `Form` control applies the style to all controls inside it. This is why all controls in this form have a light gray background. However, the `Label` control looks different. This is because we chose to override the style for the `Label` control by providing its own style reference.

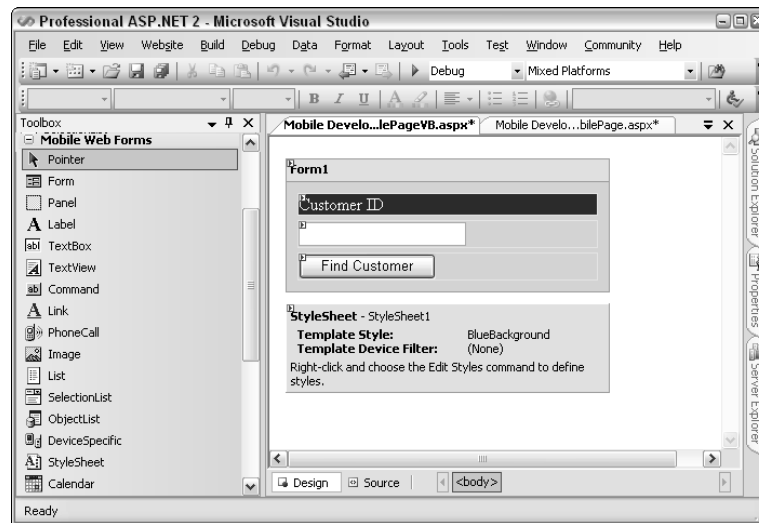


Figure 28-6

## Creating a Single StyleSheet Control for All Mobile Web Forms

It is easy for you to create a `StyleSheet` control that can be used consistently for all mobile Web pages in your mobile Web application. You simply create a Mobile User control and place a `StyleSheet` control inside it. You can reuse the `StyleSheet` placed inside a Mobile User control by dropping a `StyleSheet` control on the Mobile Web Form and setting the `ReferencePath` property to the Mobile User Control. We talk more about the Mobile User Control a little later in this chapter.

You should know that Visual Studio doesn't understand the global stylesheet reference and cannot assist you in applying these styles. Most notable is the lack of design-time style support for global styles. You also do not see the list of Styles in the `StyleReference` property of the Mobile Web control.

## Using ASP.NET Mobile Controls

The ASP.NET Mobile Designer enables you to access a rich set of interactive development tools. This section discusses various mobile controls that are available for you to reuse. These controls provide functionality suitable for all your mobile Web development needs.

### The AdRotator Control

The `AdRotator` mobile Web control is similar to the `AdRotator` control in ASP.NET Web Forms. It is capable of displaying and cycling through a random set of advertisement banners. This control automates the cycling process and changes the displayed advertisement every time the page is refreshed. You can customize this control to give more weight to certain advertisements and create a priority level for the banners. You can also provide a custom logic for cycling through the advertisements.

## Chapter 28

The AdRotator control provides a few very important properties that can be used to provide it a list of advertisements, image paths, and image links. The table that follows shows the important properties of this control.

Property	Description
AdvertisementFile	This read-write property receives a path to the advertisement file. This file should contain an XML-based definition of advertisement information, such as Image URL, Navigate URL, Number of Impressions, Start Date, and End Date.
ImageKey	This read-write property enables you to select a custom-defined tag in the advertisement XML file to find the URL for images.
NavigateUrlKey	This read-write property allows you to select a custom-defined tag in the advertisement XML file to find the URL link associated with each image.

Listing 28-2 shows the XML-based advertisement configuration file used for the AdRotator control shown in Figure 28-7. Save this file as Listing 28-02.xml. The AdRotator control selects a different image every time the page is loaded, so Figure 28-7 displays the same page showing two different images.

### Listing 28-2: The advertisement configuration file Listing 28-02.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
  <Ad>
    <ImageUrl>images/RDLogo.jpg</ImageUrl>
    <NavigateUrl>http://www.MicrosoftRegionalDirectors.com</NavigateUrl>
    <AlternateText>Microsoft Regional Directors</AlternateText>
    <Keyword>Community Leader</Keyword>
    <Impressions>2000</Impressions>
    <StartDate>5/19/05</StartDate>
    <EndDate>7/18/05</EndDate>
  </Ad>
  <Ad>
    <ImageUrl>images/tcnug_logo.gif</ImageUrl>
    <NavigateUrl>http://www.ilmservice.com/twincitiesnet</NavigateUrl>
    <AlternateText>Twin Cities .NET User Group</AlternateText>
    <Keyword>User Group</Keyword>
    <Impressions>1000</Impressions>
    <StartDate>5/30/05</StartDate>
    <EndDate>7/5/05</EndDate>
  </Ad>
</Advertisements>
```

After the XML file that will be used by the AdRotator control is in place, you can reference this file directly from your mobile ASP.NET page, as illustrated in Listing 28-3.

### Listing 28-3: Using the advertisement XML file with the AdRotator control

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Listing 28-03.aspx.cs"
  Inherits="Listing2503" %>
```

```

<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
  <mobile:Form id="Form1" runat="server">
    <mobile:AdRotator ID="AdRotator1" Runat="server"
      AdvertisementFile="Listing 28-02.xml">
    </mobile:AdRotator>
    <mobile:Label id="lblID" Runat="server">ID
    </mobile:Label>
    <mobile:TextBox id="txtCustID" Runat="server">
    </mobile:TextBox>
    <mobile:Command id="cmdGetCustomer" Runat="server"
      OnClick="cmdGetCustomer_Click">Find
    </mobile:Command>
  </mobile:Form>
</body>
</html>

```

Once in place, you get the varying results shown in Figure 28-7.

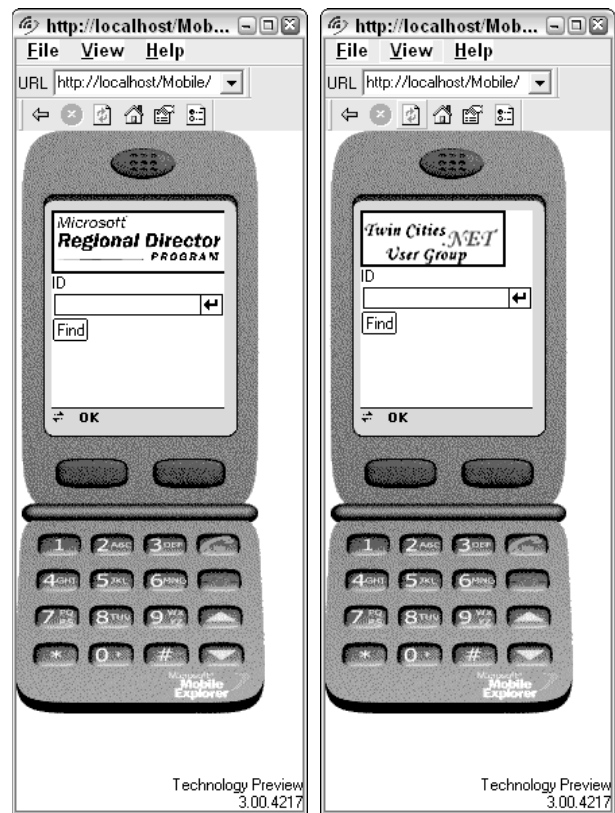


Figure 28-7

## Chapter 28

### The Calendar Control

The Calendar control offers date-picking functionality. You can add it to the Mobile Web Form by simply dragging and dropping from the Toolbox. The control starts by showing the current month by default. You can set the `VisibleDate` property to cause it to show a different month by default. The `SelectedDate` Property can be used to select a date on the calendar. The visible date can be changed irrespective of the `SelectedDate` property, causing the control to remember a selected date that may not be currently on display.

The `SelectionMode` property exposed by the Calendar control determines the manner in which the dates are selected. The default setting is `Day`, which allows the user to choose a single day. You can change this property to either `DayWeek` or `DayWeekMonth`. The `DayWeek` setting enables the user to select either a single day or a week. The `DayWeekMonth` setting allows the user to select a day, week, or month. You can change this setting at design time, or you can change it programmatically at runtime. This control raises an event with the name `SelectionChanged`. This event gets fired every time the user changes the currently selected date. The following table describes the Calendar control properties just mentioned.

Property	Description
<code>FirstDayOfWeek</code>	This read/write property enables the users to see the calendar starting from a given day.
<code>SelectionMode</code>	This read/write property allows you to configure the Calendar control to let the users select a day, a week, or the entire month. The available choices are <code>None</code> , <code>Day</code> , <code>DayWeek</code> , and <code>DayWeekMonth</code> .
<code>SelectedDate</code>	This read/write property enables you to pre-select a specific date. We can either set this property at design time, or alter its value at runtime. You should also know that if the selected date is set to a date that isn't currently visible on the screen, the Calendar control does not automatically change its appearance to show you the selected date.
<code>VisibleDate</code>	This read/write property allows you to set the date that should be visible on the screen when the calendar is displayed. This date doesn't have to be the same as the selected date.

Using the Calendar control is easy. You simply drag this control from the Toolbox, position it on the Mobile Web Form, and set the appropriate property to display the calendar in your favorite style. Interacting with the calendar is easy as well. For example, when a user selects a date on the calendar, you can read this date and populate a text box with it by writing just a single line of code in the Calendar control's `SelectionChange` event, as shown in Listing 28-4.

#### Listing 28-4: An example of using the Calendar control

##### Mobile page

```
<%@ Page Language="VB" AutoEventWireup="true" CodeFile="Calendar.aspx.vb"
    Inherits="Calendar" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>
```

```

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
  <mobile:Form id="Form1" runat="server">
    Event Date:
    <mobile:TextBox id="txtEventDate" runat="server">
  </mobile:TextBox>
  <mobile:Calendar id="EventCalendar"
    FirstDayOfWeek="Sunday" Runat="server"
    OnSelectionChanged="EventCalendar_SelectionChanged">
  </mobile:Calendar>
  </mobile:Form>
</body>
</html>

```

**VB**

```

Partial Class Calendar
  Inherits System.Web.UI.MobileControls.MobilePage

  Protected Sub EventCalendar_SelectionChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles EventCalendar.SelectionChanged

    txtEventDate.Text = EventCalendar.SelectedDate.ToShortDateString()
  End Sub
End Class

```

**C#**

```

using System;
using System.Web.Mobile;
using System.Web.UI.MobileControls;

public partial class Calendar : System.Web.UI.MobileControls.MobilePage
{
  protected void EventCalendar_SelectionChanged(object sender, EventArgs e)
  {
    txtEventDate.Text = EventCalendar.SelectedDate.ToShortDateString();
  }
}

```

Figure 28-8 shows the Calendar control where the user can click on a certain date and then populate a text box.

## The Label Control

The Label control is used to display read-only, text-based information on the screen. You can set the string displayed by this control using the `Text` property in the Properties window, or you can change it programmatically. If the string is too long to fit entirely on the screen, be sure to set the `Wrapping` property to `Wrap`. This causes the label to display the information on multiple lines. The following table shows a few of the commonly used properties of the Label control.

## Chapter 28

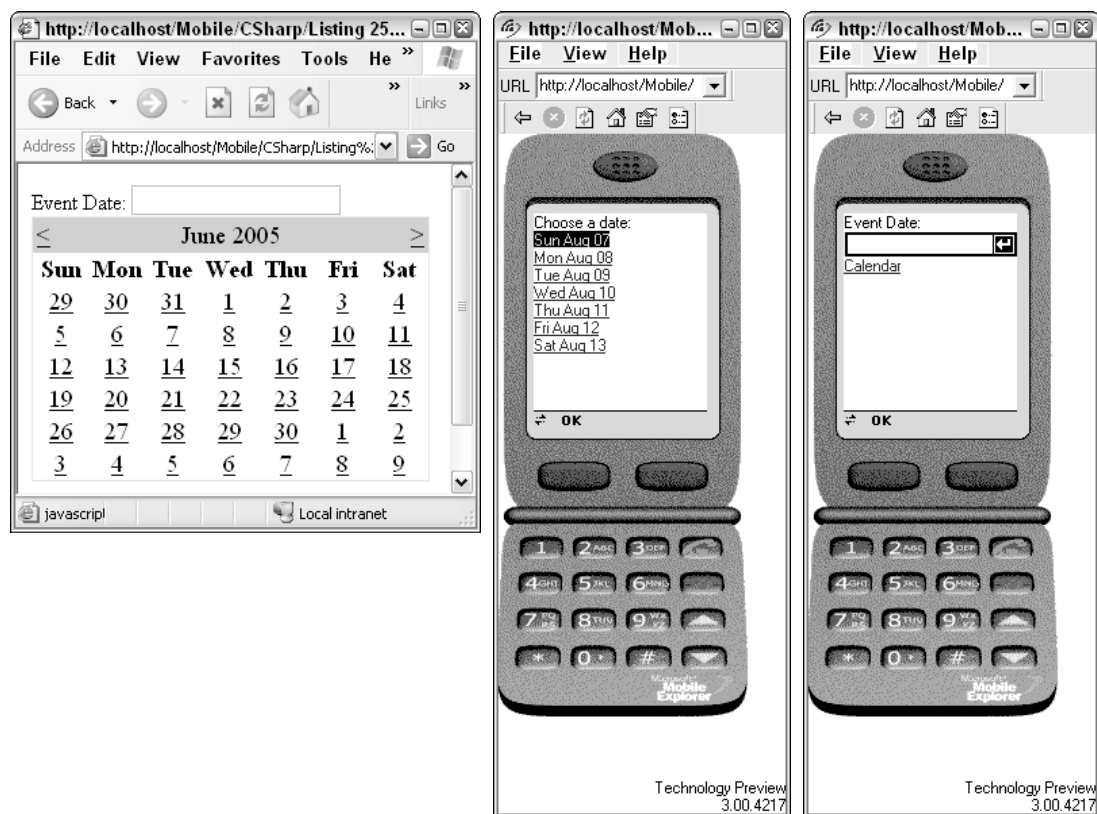


Figure 28-8

Property	Description
Wrapping	This read/write property causes the control to display the text on multiple lines if the content is too large to display entirely on one line. The possible values are <code>NotSet</code> , <code>Wrap</code> , and <code>NoWrap</code> .
Alignment	This read/write property allows you to align the control on the screen. The possible values are <code>NotSet</code> , <code>Left</code> , <code>Right</code> , and <code>Center</code> .
BreakAfter	This read/write property allows you to specify whether you want to force a line break after this control when the control is rendered. This is useful on mobile browsers that are capable of displaying more than one control on a single line. The possible values are <code>True</code> and <code>False</code> .

### The TextBox Control

You can use the `TextBox` control when you want to enable users to enter textual information. This information can also be programmatically set or retrieved using the `Text` property. You can prevent users from seeing sensitive information, such as a password, by setting the `Password` property to `True`. The following table shows a list of the `TextBox` control's common properties.

Property	Description
Text	This read/write property is used for reading and writing text-based information to and from this control. This property can either be set at design time or during runtime.
Password	This read/write property is very useful for the cases where the information is sensitive, such as a password, and shouldn't be displayed on the screen. The possible values are <code>True</code> and <code>False</code> .
Size	This read/write property allows you to specify the width of the control.
MaxLength	This read/write property allows you to specify the maximum length of the string that the text box should accept.
Alignment	This read/write property allows you to position the control on the screen. The possible values are <code>NotSet</code> , <code>Left</code> , <code>Right</code> , and <code>Center</code> .

The code example in Listing 28-5 shows a simple calculator that uses two text boxes and a label. Users can enter two numbers using the `TextBox` controls and click the `Add` button. The result will be displayed using a `Label` control.

#### Listing 28-5: An example of using `TextBox` and `Label` controls

##### Mobile page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="LabelAndTextBoxVB.aspx.vb"
    Inherits="LabelAndTextBox" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">&nbsp;
        Add Two Numbers:&nbsp;
        <mobile:TextBox ID="txtNumber1" Runat="server">
        </mobile:TextBox>
        <mobile:TextBox ID="txtNumber2" Runat="server">
        </mobile:TextBox>
        <mobile:Label ID="lblResult" Runat="server">Label
        </mobile:Label>
        <mobile:Command ID="cmdAdd" Runat="server" OnClick="cmdAdd_Click">Add
        </mobile:Command>
    </mobile:Form>
</body>
</html>
```

##### VB

```
Partial Class LabelAndTextBox
    Inherits System.Web.UI.MobileControls.MobilePage

    Protected Sub cmdAdd_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles cmdAdd.Click
```

(continued)

## Chapter 28

---

**Listing 28-5:** *(continued)*

```
Dim Number1 As Integer
Dim Number2 As Integer

Number1 = Convert.ToInt32(txtNumber1.Text)
Number2 = Convert.ToInt32(txtNumber2.Text)

lblResult.Text = Convert.ToString(Number1 + Number2)
End Sub
End Class
```

**C#**

```
using System;
using System.Web;
using System.Web.Mobile;
using System.Web.UI.MobileControls;

public partial class LabelAndTextBoxCSharp :
    System.Web.UI.MobileControls.MobilePage
{
    protected void cmdAdd_Click(object sender, EventArgs e)
    {
        int Number1;
        int Number2;

        Number1 = Convert.ToInt32(txtNumber1.Text);
        Number2 = Convert.ToInt32(txtNumber2.Text);

        lblResult.Text = Convert.ToString(Number1 + Number2);
    }
}
```

Figure 28-9 shows the calculator from Listing 28-5.

### The TextView Control

The TextView control works much like the Label control except that it can display large fields of textual data. You can style the text to appear with normal, bold, and italic formatting; you can also use line breaks, paragraph markers, and hyperlinks.

In reality, this control doesn't offer anything different or better than the Label control. The earlier versions of the .NET Framework didn't allow the Label control to display its content on multiple lines by wrapping text. However, the 2.0 version of the .NET Framework enables the Label control to wrap, thereby causing the TextView control to be redundant with the Label control.

## Mobile Development

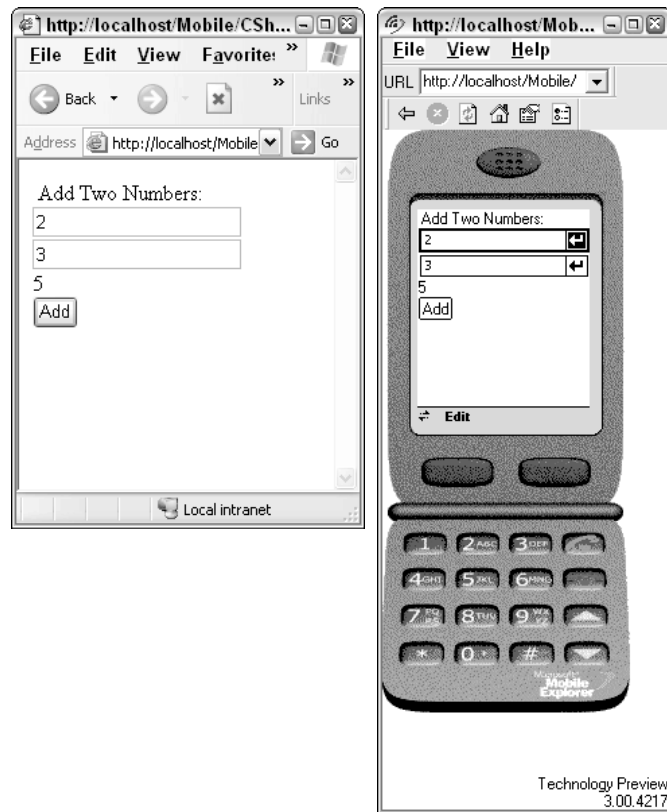


Figure 28-9

The following table shows the commonly used properties of the TextView control.

Property	Description
Wrapping	This read/write property causes the control to display the text on multiple lines if the content is too large to display entirely on one line. The possible values are <code>NotSet</code> , <code>Wrap</code> , and <code>NoWrap</code> .
Alignment	This read/write property allows you to align the control on the screen. The possible values are <code>NotSet</code> , <code>Left</code> , <code>Right</code> , and <code>Center</code> .
BreakAfter	This read/write property allows you to specify whether you want to force a line break after this control when the control is rendered. This is useful on mobile browsers that are capable of displaying more than one control on a single line. The possible values are <code>True</code> and <code>False</code> .

## Chapter 28

### The Command Control

The Command control displays a button on the screen and is used for capturing user input and processing it on the server. When the user clicks on the button, this control automatically fires two events on the server with the names `Click` and `ItemCommand`. Both events can be handled in the same page. However, if this control is contained inside a container, the `ItemCommand` event is also propagated to the parent control.

You should know that the `CausesValidation` property of this control is set to `True` by default, which causes the validation controls to activate on the Mobile Web Form when the user clicks this control. You can disable this behavior by setting the `CausesValidation` property to `False`.

The following table shows the commonly used properties of the Command control.

Property	Description
<code>Text</code>	This read/write property displays textual information on the screen. The information is shown as a read-only caption on the control.
<code>ImageUrl</code>	This read/write property can be used to provide a link to an image that will be rendered instead of the default button.
<code>CausesValidation</code>	This property decides whether the validation controls on the Mobile Web Form should fire or not when the user clicks the control.
<code>BreakAfter</code>	This read/write property allows you to specify whether you want to force a line break after this control when the control is rendered. This is useful on mobile browsers that are capable of displaying more than one control on a single line. The possible values are <code>True</code> and <code>False</code> .

### The Image Control

The Image control is useful for displaying an image on the screen by specifying the location of a bitmap file using the `ImageUrl` property. You can also cause this control to act as a hyperlink by setting the `NavigateUrl` property to a valid URL. The following table shows the Image control's commonly used properties.

Property	Description
<code>NavigateUrl</code>	This read/write property allows you to provide a link so that users can click on the image and get redirected to another Mobile Web Page or to another Mobile Web Form on the same page.
<code>ImageUrl</code>	This read/write property can be used to provide a link to an image that will be rendered when the Mobile Web Form is rendered.
<code>AlternateText</code>	This read/write property allows you to provide a textual description of the image. This description automatically renders if the Mobile Browser can't display the image.

Property	Description
Alignment	This read/write property allows you to position this control on the Mobile Web Form. The possible values are <code>NotSet</code> , <code>Left</code> , <code>Right</code> , and <code>Center</code> .
BreakAfter	This read/write property allows you to specify whether you want to force a line break after this control when the control is rendered. This is useful on mobile browsers that are capable of displaying more than one control on a single line. The possible values are <code>True</code> and <code>False</code> .

Listing 28-6 shows an example of how you can use this control,

### Listing 28-6: An Image control on a Mobile Web Form

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ImageCSharp.aspx.cs"
    Inherits="ImageCSharp" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <mobile:Image ID="Image1" Runat="server"
            AlternateText="Microsoft Regional Director"
            ImageUrl="~/Mobile Development/Images/RDLogo.JPG">
        </mobile:Image>
    </mobile:Form>
</body>
</html>
```

You can imagine the complexity of displaying an image on a wide variety of devices. The widely varying capabilities of mobile devices make it nearly impossible to display the same image on all devices. However, this control provides a powerful set of tools for overcoming this limitation. The device filters, for example, allow you to select an image to display from a group of images. Each image in the group can be targeted toward specific types of devices, such as a color image on handheld computers or a simplified monochrome image more suitable to the phone's display. The control chooses the most appropriate image to display by overriding its property values for specific hardware. Device filters are discussed in more detail in the section "Understanding Device Filters" a little later in this chapter.

## The PhoneCall Control

The PhoneCall control is useful for those mobile devices that can originate a phone call, such as mobile phone devices. This control displays to the user a string that appears as a command the user can select. You can set the contents of the string with the Text property and use the PhoneNumber property to enter the number for the device to call.

The devices that can't originate a phone call simply display a text value according to the format string specified in the AlternateFormat property. By default, the AlternateFormat property contains {0} {1}

## Chapter 28

as its formatting string. The control replaces the {0} with the string in the `Text` property and replaces the {1} with the contents of the `PhoneNumber` property.

The following table shows the commonly used properties of the `PhoneCall` control.

Property	Description
<code>AlternateFormat</code>	This read/write property allows you to format the way the phone number should appear in case the mobile device isn't capable of initiating a voice communication.
<code>AlternateUrl</code>	This read/write property allows you to provide a link in case the mobile device isn't capable of initiating a voice communication. This link redirects the user to another Mobile Web Page or to another Mobile Web Form on the same page.
<code>Text</code>	This read/write property allows you to provide a textual description of the image. This description automatically renders if the Mobile Browser can't display the image.
<code>PhoneNumber</code>	This read/write property allows you to position this control on the Mobile Web Form. The possible values are <code>NotSet</code> , <code>Left</code> , <code>Right</code> , and <code>Center</code> .
<code>BreakAfter</code>	This read/write property allows you to specify whether you want to force a line break after this control when the control is rendered. This is useful on mobile browsers that are capable of displaying more than one control on a single line. The possible values are <code>True</code> and <code>False</code> .

An example of using the `PhoneCall` control is shown in Listing 28-7.

### Listing 28-7: A `PhoneCall` control

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="PhoneCallCSharp.aspx.cs"
    Inherits="PhoneCallCSharp" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <mobile:PhoneCall ID="PhoneCall1" Runat="server" Alignment="Left"
            AlternateUrl="~/MyMS.aspx" PhoneNumber="1-800-555-1212">
            Call Microsoft
        </mobile:PhoneCall>
    </mobile:Form>
</body>
</html>
```

Figure 28-10 shows the `PhoneCall` control.

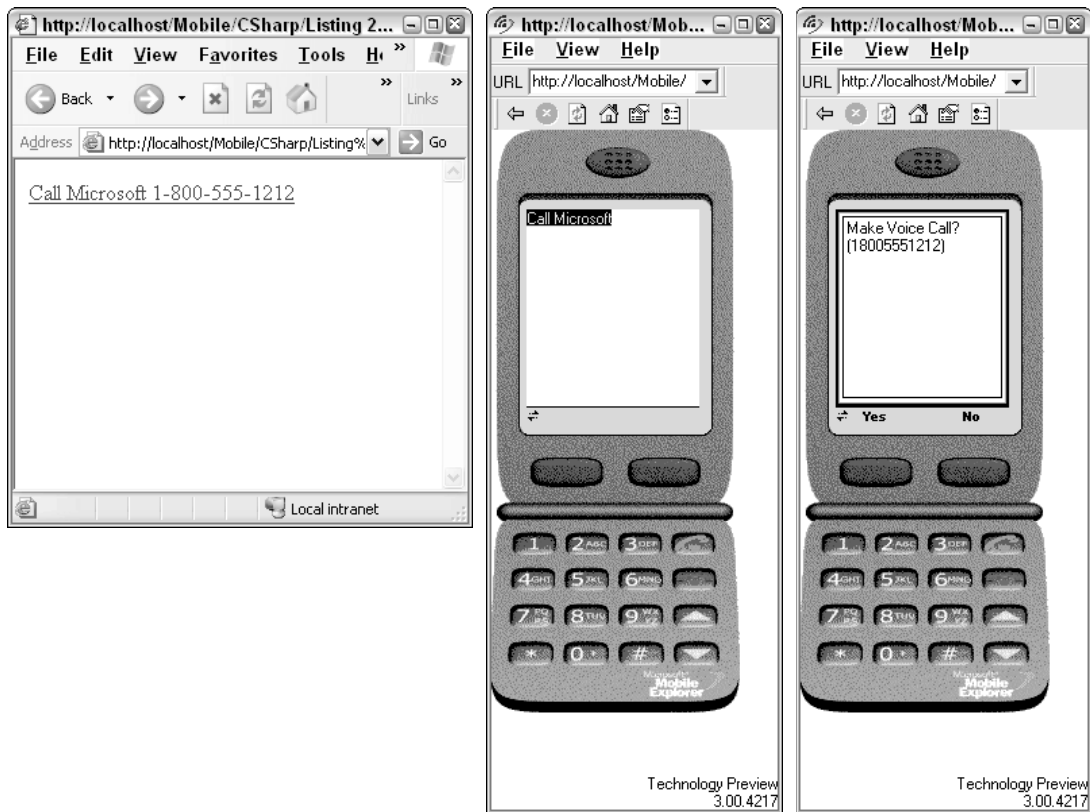


Figure 28-10

### The Link Control

The Link control displays a text string as a hyperlink that can lead to another form on the same Mobile Web Forms page or to any other URL. You can take advantage of the devices that support softkeys by specifying the `SoftKeyLabel` property and entering the link's text into the `Text` property. (A *softkey* is a key on a mobile device that lets a user execute a function; these keys can be used for multiple links. Softkeys generally correspond to a value that appears on the screen above the button.) You can customize the appearance of the Link control by setting the `Alignment`, `ForeColor`, `Font`, `StyleReference`, and `Wrapping` properties. The following table shows the Link control's commonly used properties.

Property	Description
<code>NavigateUrl</code>	This read/write property allows you to provide a link to another Mobile Web Page or another Mobile Web Form on the same page. The user is taken to this URL when the Link control is clicked.
<code>Text</code>	This read/write property allows you to provide a textual description of the link that is displayed for users.

Table continued on following page

## Chapter 28

Property	Description
SoftKeyLabel	This read/write property allows you to provide a label for the softkey. This property is applicable only for mobile devices that provide softkey functionality. You can programmatically configure this key to be associated with any link control you prefer.
BreakAfter	This read/write property allows you to specify whether you want to force a line break after this control when the control is rendered. This is useful on mobile browsers that are capable of displaying more than one control on a single line. The possible values are <code>True</code> and <code>False</code> .

Link controls are very useful for creating menus on mobile devices. Mobile devices have very restricted input capabilities, especially phone devices, so you can make your applications much more user friendly by providing a menu of options for users to select from. The example shown in Listing 28-8 provides a short menu. Users can click either on the Contact, Candidate, or Hours link to navigate to other Mobile Web Forms.

### Listing 28-8: A menu using Link controls

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="LinkMenuCSharp.aspx.cs"
    Inherits="LinkMenuCSharp" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Shajar" runat="server">
        <mobile:Link ID="lnkContacts" Runat="server"
            NavigateUrl="/Contacts.aspx" SoftkeyLabel="Contacts">
            Search for Contacts
        </mobile:Link>
        <mobile:Link ID="lnkCandidates" Runat="server"
            NavigateUrl="/Candidates.aspx" SoftkeyLabel="Candidates">
            Search for Candidates
        </mobile:Link>
        <mobile:Link ID="lnkHours" Runat="server" NavigateUrl="/Hours.aspx"
            SoftkeyLabel="Hours">
            Hours Report
        </mobile:Link>
    </mobile:Form>
</body>
</html>
```

Figure 28-11 shows a menu built with Link controls.

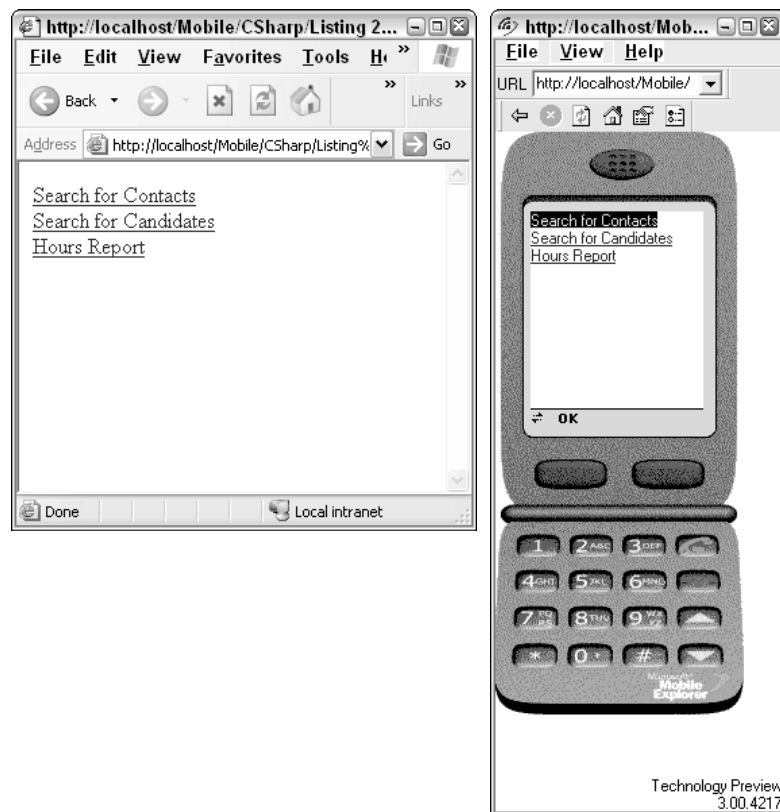


Figure 28-11

## The List Control

The List control is very useful in displaying a list of items either as bullets, numbers, or a plain list. You can provide a static list of items or bind this control to a list retrieved from the database. The static list is provided by clicking the ellipsis (...) button next to the `Items` property in the Properties window. The list might paginate on some devices depending on the viewable area available. You can set the `ItemsPerPage` property to the preferred number of list items to display on each page. You should know that the Visual Studio Mobile Designer doesn't use this property when showing you a default rendering in the Design view.

Data binding this control to a list obtained from the database is quite easy. You simply specify the data source using the `DataSource` and `DataMember` properties. Be sure to provide appropriate values to the `DataTextField` and `DataValueField` properties so that the control manages the viewable column from the data source.

## Chapter 28

You can customize the appearance of this control by using the `Alignment`, `ForeColor`, `Font`, `StyleReference`, and `Wrapping` properties. You can cause the control to display the items as either bullets or numbers by providing the appropriate value for the `Decoration` property.

The following table shows the List control's commonly used properties.

Property	Description
<code>DataSource</code>	This property lets you provide a data source object that the control can use to obtain the list.
<code>DataMember</code>	This property allows you to provide the <code>DataTable</code> name if you choose to use the <code>DataSet</code> object as the data source.
<code>DataTextField</code>	This property allows you to select a field from the data source to be displayed on the screen.
<code>DataValueField</code>	This property allows you to select a field from the data source to be used as values for each item. The values aren't displayed on the screen. Instead, they are used to store identifiers for each displayed item.
<code>Decoration</code>	This property allows you to select the list style. The available choices are <code>None</code> , <code>Bulleted</code> , and <code>Numbered</code> .
<code>Wrapping</code>	This property allows you to display a list item on multiple lines if the content is too big to fit on one line.

Listing 28-9 shows how to bind the List control to a list obtained from the database. We chose to display the Company Name field from the data source in a bulleted-list format.

### Listing 28-9: Binding to a List control

```
<%@ Page Language="VB" AutoEventWireup="true" CodeFile="List.aspx.vb"
    Inherits="List" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <mobile:List ID="LResult" Runat="server" DataTextField="CompanyName"
            DataValueField="CustomerID" Decoration="Bulleted">
        </mobile:List>
    </mobile:Form>
</body>
</html>
```

#### VB

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Configuration

Partial Class List
```

```

Inherits System.Web.UI.MobileControls.MobilePage

Protected Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load

    If Not Page.IsPostBack Then
        Dim MyConnection As SqlConnection
        Dim MyCommand As SqlCommand
        Dim MyReader As SqlDataReader

        MyConnection = New SqlConnection()
        MyConnection.ConnectionString = _
            ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

        MyCommand = New SqlCommand()
        MyCommand.CommandText = " SELECT TOP 3 * FROM CUSTOMERS "
        MyCommand.CommandType = CommandType.Text
        MyCommand.Connection = MyConnection

        MyCommand.Connection.Open()
        MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection)

        ListControl.DataSource = MyReader
        ListControl.DataBind()

        MyCommand.Dispose()
        MyConnection.Dispose()
    End If

End Sub
End Class

```

**C#**

```

using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.Mobile;
using System.Web.UI.MobileControls;
using System.Configuration;

public partial class List : System.Web.UI.MobileControls.MobilePage
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            string conn;
            SqlConnection MyConnection;
            SqlCommand MyCommand;
            SqlDataReader MyReader;

            MyConnection = new SqlConnection();
            conn =

            ConfigurationManager.ConnectionStrings["DSN_Northwind"].ConnectionString;

```

*(continued)*

## Chapter 28

**Listing 28-9:** *(continued)*

```
MyConnection.ConnectionString = conn;
MyCommand = new SqlCommand();
MyCommand.CommandText = " SELECT TOP 3 * FROM CUSTOMERS ";
MyCommand.CommandType = CommandType.Text;
MyCommand.Connection = MyConnection;

MyCommand.Connection.Open();
MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection);

LResult.DataSource = MyReader;
LResult.DataBind();

MyCommand.Dispose();
MyConnection.Dispose();

}
}
```

Figure 28-12 shows the result of compiling and executing the code in Listing 28-9.

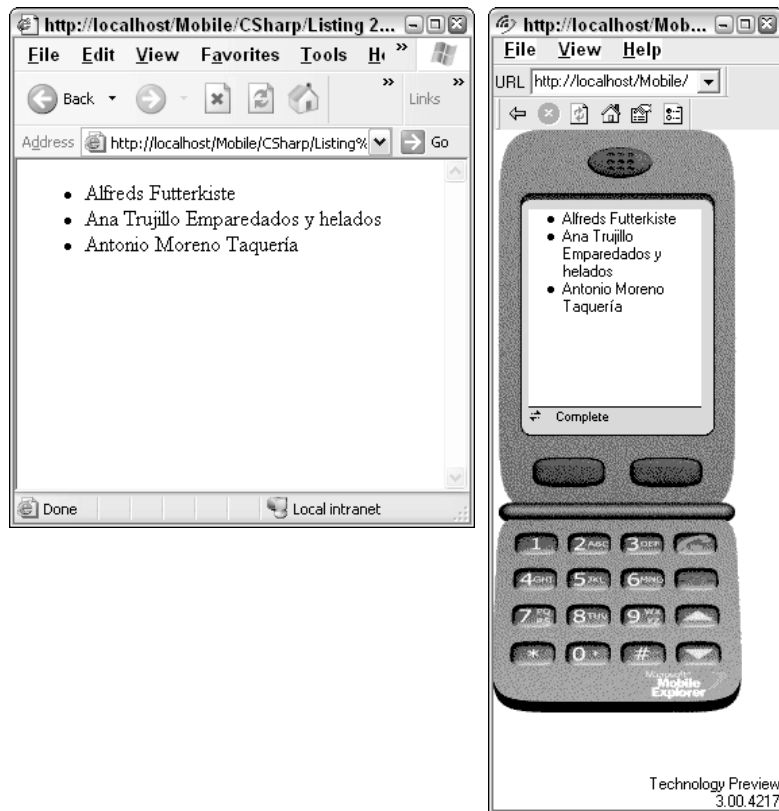


Figure 28-12

## The ObjectList Control

The ObjectList control provides an easy-to-use way of viewing tabular information from a database. You can bind a list of records retrieved from the database. The control starts by showing you just a single column from the data source. You can select a record from the list to cause this control to post back to the server and display all columns for the selected record in a vertical list. The control also automatically provides a Back button, which takes you back to the screen showing a single column for all records. You can select the column that you want on the first screen by setting the LabelField property. Leaving this property alone causes this control to select the first column from the record to display on the first screen.

You can reuse the code shown in Listing 28-9 and modify it slightly to bind to an ObjectList control. The result of using this control is shown in Figure 28-13. Clicking the link in the screen on the left brings up the screen shown on the right.

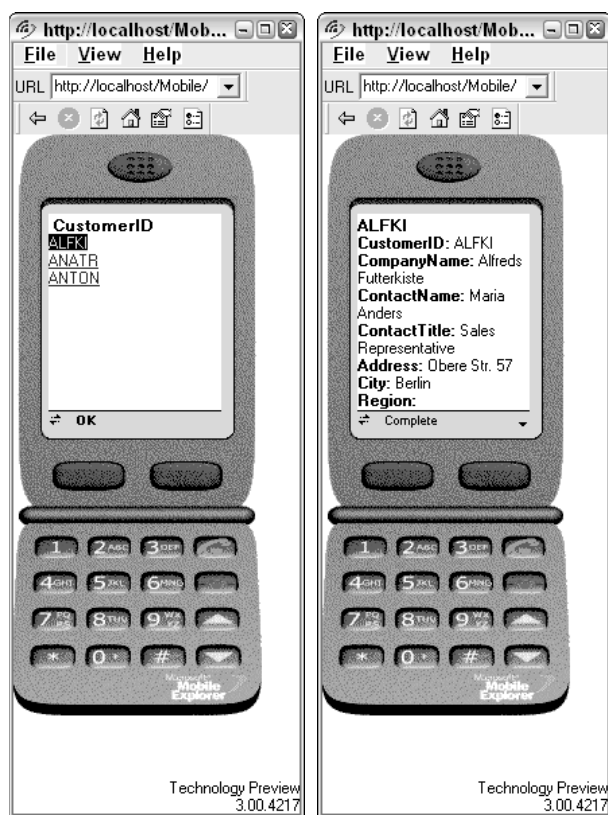


Figure 28-13

## The SelectionList Control

The SelectionList control shows a list of items in the form of either a drop-down list, list box, check box list, or radio buttons list, enabling the user to select one or more items from the list. This control doesn't support pagination and is, therefore, better suited for short lists.

## Chapter 28

Unlike its counterpart in the ASP.NET Web Forms, this control doesn't provide an auto post back property. Be sure to add a Command control, which causes the postback and fires a server-side event. This control, however, also fires a `SelectedIndexChanged` event on the server after the command object has initiated the postback process. The `SelectedIndexChanged` event, of course, fires only when the user changes the selected value prior to pushing the Command control.

Specifying that this control be rendered as a drop-down list, a list box, a check box list, or a radio-button list is as simple as setting the `SelectType` property. You also have the capability to provide static content to this control by using the `Items` property.

Property	Description
<code>SelectType</code>	This property lets you specify the type of selection list you desire. The available options are <code>DropDown</code> , <code>ListBox</code> , <code>Radio</code> , <code>MultiSelectListBox</code> , and <code>CheckBox</code> .
<code>DataSource</code>	This property lets you provide a data source object that the control can use to obtain the list.
<code>DataMember</code>	This property lets you provide the <code>DataTable</code> name if you choose to use the <code>DataSet</code> object as the data source.
<code>DataTextField</code>	This property allows you to select a field from the data source to be displayed on the screen.
<code>DataValueField</code>	This property allows you to select a field from the data source to be used as values for each item. The values aren't displayed on the screen. Instead, they are used to store identifiers for each displayed item.
<code>Wrapping</code>	This property allows you to display a list item on multiple lines if the content is too big to fit on one line.

The code shown in Listing 28-10 illustrates four different `SelectionList` controls; each control renders a different look, even though all of these controls are bound to the same data source.

### Listing 28-10: Examples of `SelectionList` controls

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="SelectionList.aspx.vb"
    Inherits="SelectionListVB" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <mobile:SelectionList ID="slistDropDown" Runat="server"
            DataTextField="CompanyName" DataValueField="CustomerID">
        </mobile:SelectionList>
        <mobile:SelectionList ID="slistRadioButton" Runat="server"
            DataTextField="CompanyName" DataValueField="CustomerID"
            SelectType="Radio">
        </mobile:SelectionList>
```

```

<mobile:SelectionList ID="slistCheckBoxes" Runat="server"
    DataTextField="CompanyName" DataValueField="CustomerID"
    SelectType="CheckBox">
</mobile:SelectionList>
<mobile:SelectionList ID="slistListBox" Runat="server"
    DataTextField="CompanyName" DataValueField="CustomerID"
    SelectType="ListBox">
</mobile:SelectionList>

</mobile:Form>
</html>

```

**VB**

```

Imports System.Data
Imports System.Data.SqlClient
Imports System.Configuration

Partial Class SelectionList
    Inherits System.Web.UI.MobileControls.MobilePage

    Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load

        If Not Page.IsPostBack Then
            Dim MyConnection As SqlConnection
            Dim MyCommand As SqlCommand
            Dim MyAdapter As SqlDataAdapter

            Dim MyDS As DataSet
            MyDS = New DataSet()

            MyConnection = New SqlConnection()
            MyConnection.ConnectionString = _
                ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

            MyCommand = New SqlCommand()
            MyCommand.CommandText = "SELECT TOP 3 * FROM CUSTOMERS"
            MyCommand.CommandType = CommandType.Text
            MyCommand.Connection = MyConnection

            MyAdapter = New SqlDataAdapter()
            MyAdapter.SelectCommand = MyCommand

            MyAdapter.Fill(MyDS)
            MyCommand.Dispose()

            slistDropDown.DataSource = MyDS.Tables(0).DefaultView
            slistListBox.DataSource = MyDS.Tables(0).DefaultView
            slistRadioButton.DataSource = MyDS.Tables(0).DefaultView
            slistCheckBoxes.DataSource = MyDS.Tables(0).DefaultView

            slistDropDown.DataBind()
            slistListBox.DataBind()

```

*(continued)*

## Chapter 28

---

**Listing 28-10:** *(continued)*

```
        slistRadioButton.DataBind()  
        slistCheckBoxes.DataBind()  
    End If  
  
End Sub  
End Class
```

**C#**

```
using System;  
using System.Configuration;  
using System.Data;  
using System.Data.SqlClient;  
using System.Web.Mobile;  
using System.Web.UI.MobileControls;  
  
public partial class SelectionList : System.Web.UI.MobileControls.MobilePage  
{  
    protected void Page_Load(object sender, EventArgs e)  
    {  
        if (!Page.IsPostBack)  
        {  
            SqlConnection MyConnection;  
            SqlCommand MyCommand;  
            SqlDataAdapter MyAdapter;  
  
            DataSet MyDS = new DataSet();  
  
            MyConnection = new SqlConnection();  
            MyConnection.ConnectionString =  
ConfigurationManager.ConnectionStrings["DSN_Northwind"].ConnectionString;  
  
            MyCommand = new SqlCommand();  
            MyCommand.CommandText = "SELECT TOP 3 * FROM CUSTOMERS";  
            MyCommand.CommandType = CommandType.Text;  
            MyCommand.Connection = MyConnection;  
  
            MyAdapter = new SqlDataAdapter();  
            MyAdapter.SelectCommand = MyCommand;  
  
            MyAdapter.Fill(MyDS);  
            MyCommand.Dispose();  
  
            slistDropDown.DataSource = MyDS.Tables[0].DefaultView;  
            slistListBox.DataSource = MyDS.Tables[0].DefaultView;  
            slistRadioButton.DataSource = MyDS.Tables[0].DefaultView;  
            slistCheckBoxes.DataSource = MyDS.Tables[0].DefaultView;  
  
            slistDropDown.DataBind();  
            slistListBox.DataBind();  
            slistRadioButton.DataBind();  
            slistCheckBoxes.DataBind();  
        }  
    }  
}
```

```

    }
}
}

```

Running the code shown in Listing 28-10 displays four SelectionList controls, each with a different style, as shown in Figure 28-14.

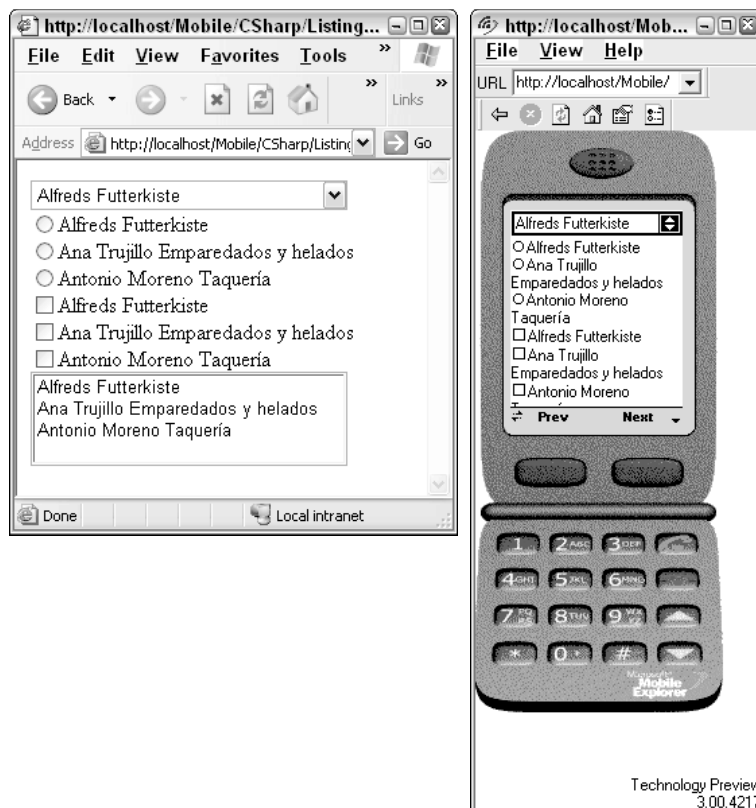


Figure 28-14

## Using Validation Controls

The validation controls in mobile Web applications work similar to the way they function in standard ASP.NET Web applications. The `RequiredFieldValidator`, for instance, is meant to ensure the user provides information in the entry fields. The `CompareValidator` is used to compare the values of two fields. The `RangeValidator` is used to ensure that entry fields contain information within an acceptable range. The `RegularExpressionValidator` validates entry fields by using a custom format string, and the `CustomValidator` validates entry fields by using custom code.

## Chapter 28

There are, however, a few differences in the way the validation controls function on Mobile Web Forms. For example, the following properties are not supported for the ValidationSummary control:

- DisplayMode
- EnableClientScript
- ShowMessageBox
- ShowSummary

On the other hand, the ValidationSummary control supports two new properties that are not supported in the ASP.NET Web Forms. These properties are `BackLabel` and `FormToValidate`. Because of the small screen size, when the validation summary is displayed on a mobile device, it is often shown on a new screen. Users have to click a Back button to get back to the form they were using before the validation error occurred. The `BackLabel` property allows you to provide a custom label for this Back button.

Mobile Web pages are also capable of using multiple Mobile Web Forms. This feature requires you to assign the ValidationSummary control to a certain Mobile Web Form. The `FormToValidate` property of the ValidationSummary Control serves this purpose.

The code shown in Listing 28-11 shows a TextBox control and two validation controls. The RequiredFieldValidator is meant to ensure the user doesn't leave the text box empty, and the RegularExpressionValidator ensures that the input value is a valid telephone number.

### Listing 28-11: Validation controls in action

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Validation.aspx.vb"
    Inherits="Validation" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <mobile:Label ID="lblPhoneNumber" Runat="server">
            Enter Phone Number:
        </mobile:Label>
        <mobile:TextBox ID="txtPhoneNumber" Runat="server">
        </mobile:TextBox>
        <mobile:RequiredFieldValidator ID="rfvPhone" Runat="server"
            ControlToValidate="txtPhoneNumber"
            ErrorMessage="Phone number must be provided">*
        </mobile:RequiredFieldValidator>
        <mobile:RegularExpressionValidator ID="revPhone" Runat="server"
            ControlToValidate="txtPhoneNumber"
            ErrorMessage="Invalid Phone Format"
            ValidationExpression="((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}">*
        </mobile:RegularExpressionValidator>
        <mobile:Command ID="cmdPhoneNumber" Runat="server">OK</mobile:Command>
        <mobile:ValidationSummary ID="ValidationSummary1"
            Runat="server" BackLabel="Return to Entering Phone Number"
```

```
FormToValidate="Form1">
  </mobile:ValidationSummary>

  </mobile:Form>
</body>
</html>
```

Figure 28-15 shows the result of executing the code shown in Listing 28-11.

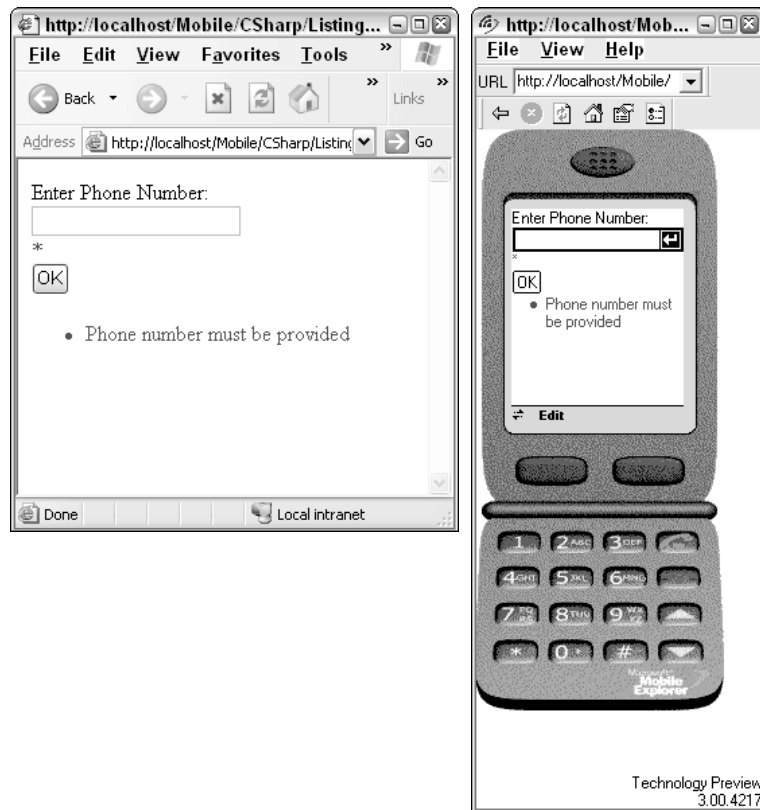


Figure 28-15

## Navigating between Mobile Web Forms

A Mobile Web page can contain more than one Mobile Web Form, so you need to learn how to navigate between these forms. There are two ways you can navigate between Mobile Web Forms. You can either set the `ActiveForm` property of the Mobile Web page, or you can redirect using the Link control. If you are redirecting the Link control, you can create a navigation link to another Mobile Web Form on the same Mobile Web page by naming the form and prefixing it with the # symbol.

## Chapter 28

---

You can configure both these ways programmatically. While you are inside the class for the Mobile Web Form, you always have access to the `ActiveForm` property, which you can set at any time. You can also programmatically access the `NavigateUrl` property of the Link control and change its value to point to a different Mobile Web Form.

## The Mobile Web User Control

A mobile user control is very similar to the ASP.NET Mobile Web Form. You create it in the same manner, fill it with controls and content, and then use it as a control in a page. Mobile Web user controls are stored in `.ascx` files and usually have an associated code-behind file. These controls inherit from the base class `MobileUserControl` in the namespace `System.Web.UI.MobileControls`.

To create a Mobile Web user control, follow these steps:

1. Right-click the project and select Add New Item.
2. In the Add New Item dialog, select Mobile Web User Control from the Visual Studio Installed Templates section.
3. Give the user control a name and select the programming language of your choice. Click the Add button.

The code example shown in Listing 28-13 displays a Mobile Web user control that asks the user to provide a Customer ID. Using the input Customer ID, this user control searches the database and displays a list of matching orders using an `ObjectList` control. The `ObjectList` control initially shows only a list of `OrderID` values. You can click a specific `OrderID` to see the details of that order.

Before that, though, Listing 28-12 shows a mobile page that is using this user control.

---

### Listing 28-12: Utilizing a mobile user control

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Listing 28-12.aspx.vb"
    Inherits="UserControlSample" %>

<%@ Register Src="OrdersList.ascx" TagName="OrdersList" TagPrefix="uc1" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <uc1:OrdersList ID="OrdersList1" runat="server" />
    </mobile:Form>
</body>
</html>
```

From this listing, you can see that first the user control is registered on the mobile page using the `@Register` page directive. You can also see that by using the attributes `TagName` and `TagPrefix`, you can define how you will declare the user control in your code. Finally, the `Src` attribute points to the location of the user control. You should review how this user control is constructed so you can actually use it in this simple page (see Listing 28-13).

**Listing 28-13: Constructing the user control****.ASCX**

```
<%@ Control Language="VB" AutoEventWireup="false" CodeFile="OrdersList.ascx.vb"
    Inherits="OrdersList" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

    <mobile:Label ID="lblID" Runat="server">Customer ID</mobile:Label>
    <mobile:TextBox ID="txtCustID" Runat="server">
    </mobile:TextBox>
    <mobile:Command ID="cmdShowOrders" Runat="server"
        OnClick="cmdGetCustomer_Click">
        Show Orders
    </mobile:Command>
    <mobile:ObjectList ID="OLOrders" Runat="server"
        CommandStyle-StyleReference="subcommand"
        LabelStyle-StyleReference="title">
    </mobile:ObjectList>
```

**VB**

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Configuration

Partial Class OrdersList
    Inherits System.Web.UI.MobileControls.MobileUserControl

    Protected Sub cmdGetCustomer_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles cmdShowOrders.Click

        Dim MyConnection As SqlConnection
        Dim MyCommand As SqlCommand
        Dim CustIDParam As SqlParameter
        Dim MyReader As SqlDataReader

        MyConnection = New SqlConnection()
        MyConnection.ConnectionString = _
            ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

        MyCommand = New SqlCommand()
        MyCommand.CommandText = "SELECT * FROM ORDERS WHERE CustomerID = @CustID"
        MyCommand.CommandType = CommandType.Text
        MyCommand.Connection = MyConnection

        CustIDParam = New SqlParameter()
        CustIDParam.ParameterName = "@CustID"
        CustIDParam.SqlDbType = SqlDbType.NChar
        CustIDParam.Size = 5
        CustIDParam.Direction = ParameterDirection.Input
        CustIDParam.Value = txtCustID.Text

        MyCommand.Parameters.Add(CustIDParam)
```

*(continued)*

## Chapter 28

---

**Listing 28-13:** *(continued)*

```
MyCommand.Connection.Open()
MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection)

OLOrders.DataSource = MyReader
OLOrders.DataBind()

MyCommand.Dispose()
MyConnection.Dispose()

End Sub
End Class
```

**C#**

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.Mobile;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.MobileControls;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

public partial class OrdersListCSharp :
System.Web.UI.MobileControls.MobileUserControl
{
    protected void cmdGetCustomer_Click(object sender, EventArgs e)
    {
        SqlConnection MyConnection ;
        SqlCommand MyCommand;
        SqlParameter CustIDParam;
        SqlDataReader MyReader;

        MyConnection = new SqlConnection();
        MyConnection.ConnectionString =
            ConfigurationManager.ConnectionStrings["DSN_Northwind"].ConnectionString;

        MyCommand = new SqlCommand();
        MyCommand.CommandText = "SELECT * FROM ORDERS WHERE CustomerID = @CustID";
        MyCommand.CommandType = CommandType.Text;
        MyCommand.Connection = MyConnection;

        CustIDParam = new SqlParameter();
        CustIDParam.ParameterName = "@CustID";
        CustIDParam.SqlDbType = SqlDbType.NChar;
        CustIDParam.Size = 5;
```

## Mobile Development

```

CustIDParam.Direction = ParameterDirection.Input;
CustIDParam.Value = txtCustID.Text;

MyCommand.Parameters.Add(CustIDParam);

MyCommand.Connection.Open();
MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection);

OLOrders.DataSource = MyReader;
OLOrders.DataBind();

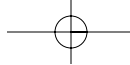
MyCommand.Dispose();
MyConnection.Dispose();
}
}

```

This Mobile Web user control is now ready to be plugged into any Mobile Web Form (such as the one from Listing 28-13). You can create a new Mobile Web Form and drag and drop this Mobile Web user control onto it. You can see how easy it is to reuse it. Figure 28-16 shows the Mobile Web Form displaying its content using a mobile user control.



Figure 28-16



## Chapter 28

---

# Using Emulators

So far, you have learned that you can develop mobile Web applications for a wide variety of mobile devices. Naturally, you want to be able to ensure that the application will look and function acceptably on the device on which you intend to use to access the application. The only sure ways of testing the application for various devices are either to deploy the application to a server with Internet access or to use a device emulator. Most developers are reluctant to deploy an untested application for Internet accessibility. Device emulators, therefore, provide an effective and practical way for testing an application's appearance and behavior on specific devices.

The manufacturers of most mobile devices provide emulators that simulate the operation of their hardware and browsers. Emulator software enables you to view your ASP.NET Mobile Web Forms application as it might appear on the manufacturer's hardware device. Viewing a Mobile Web Forms application on an emulator is simple. You compile the application and place the URL of the application's start page into the appropriate place in the emulator's browser. Your application operates on an emulator as it would on the actual hardware device.

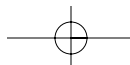
There are many online sources from which you can download mobile emulators. A few of these sources are listed here:

- ❑ **OpenWave:** Download this emulator from <http://developer.openwave.com>. The Open Wave emulator supports WML, HTML, and XHTML.
- ❑ **Ericsson:** You can download this emulator (called WAPIDE) from [www.ericsson.com/developers](http://www.ericsson.com/developers). The easiest way to find it is to search for WAPIDE on Ericsson's Web site. This emulator requires Java Runtime.
- ❑ **Nokia:** You can download this emulator from [www.forum.nokia.com](http://www.forum.nokia.com). The easiest way to find it is to search for Mobile Internet Toolkit on Nokia's Web site. This emulator requires Java Runtime to be installed on the machine.

Microsoft Visual Studio uses a built-in browser as the default application browser. You can alter this default so that Visual Studio invokes a mobile device emulator instead. Visual Studio also enables you to easily select a different current device emulator to act as the default application browser. You can obtain the device emulators from mobile device hardware manufacturers and install them on your development computer.

To use an emulator as the Visual Studio application browser, follow these steps:

1. Install and test the mobile device emulator on your development computer as instructed in the emulator's documentation.
2. Right-click on any .ASPX page within a project displayed in the Solution Explorer and choose Browse With to open the Browse With dialog.
3. Click the Add button.
4. Browse to find the executable file for the emulator and provide a friendly name for it.
5. Click OK.
6. If you want to set the emulator as the default browser, highlight it from the list in the Browse With dialog and click the Set As Default button.



## Understanding Devices Filters

Device filters (the ability to figure out which device you are dealing with through a filtering process) give you the ultimate flexibility. You can customize the appearance of controls for either a specific device or for categories of devices. You can base the customization completely on the capability of devices. It helps you ensure that your application looks attractive and functions on all targeted devices.

You can accomplish a number of tasks by using device filters. For example, you can select styles based on the type of device, and you can render a richer presentation to the devices that support it and a toned-down presentation to devices that don't. The device filter stores specific information in the `<deviceFilters>` section of the `web.config` file.

There are two types of device filters: comparison-based and evaluator. You can use the comparison-based filter to determine whether a device supports a specific capability. You can accomplish this by comparing the current value of a device capability with a specific value.

The evaluator uses a delegate-based device filter. It allows you to provide a method that is called by the device filter so that you can write custom code to process complex evaluations yourself. You have the flexibility to define this method either in a separate assembly or in the code-behind page.

After you have defined one filter, you can apply it to any number of controls in your mobile Web application. This flexibility is possible because all comparison-based filters are automatically stored in the `web.config` file. The evaluator filters, however, are not declared in the configuration file.

To declare a comparison-based device filter, follow these steps:

1. From the Design View of a page, select the control to which you want to add a comparison-based device filter.
2. Click the ellipsis (...) button next to the `AppliedDeviceFilters` property in the Properties tool window. This action launches the Applied Device Filters dialog.
3. Click the Edit button to either create or modify filters.
4. In the Device Filter Editor dialog, click the New Device Filter button, type the name, and select the filter type.
5. In the Attributes section, select the attribute the device filter will use from the Compare drop-down box, and enter the value to compare against in the Argument text box.
6. Use the Up and Down arrows to set the filter order. The filters are stored in the `web.config` file in this order, and this is the order in which they are applied.
7. Click OK to save the device filter information.

After defining device filters, you can add them to the list of applied filters for a control. You simply click the ellipsis (...) button next to the `AppliedFilters` property of the control. This opens the Applied Device Filters dialog. Select your desired filter from the list of filters and click the Add button. Be sure to keep the filters in the order you need by using the Up Arrow and Down arrow buttons. At runtime, the mobile Web application tests the filters one by one from the top to the bottom. Consequently, the first device filter that results in a successful evaluation determines which property overrides or templates the application uses.

## Chapter 28

It is also a good practice to add the (Default) filter at the end of the list. This ensures that the device that doesn't match any of the filters receives the property values defined for the (Default) filter. This filter always results in a successful evaluation and blocks all other evaluations below it in the list.

Mobile Web controls differentiate between device filters by using the name and the argument value of each filter. This means that you can give two device filters the same name as long as they have different argument values.

Figure 28-17 shows the Applied Device Filter and Device Filter Editor dialogs.

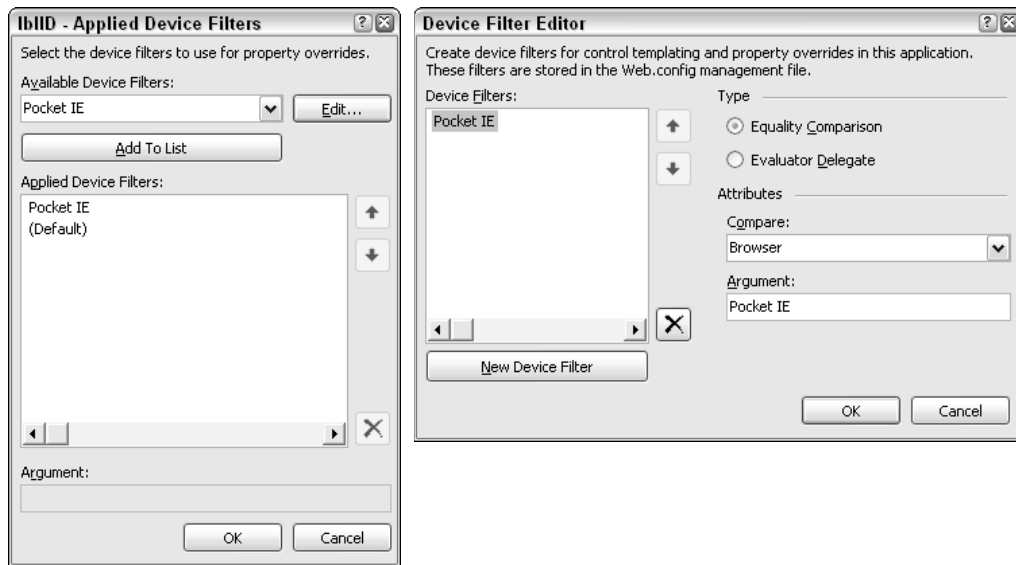


Figure 28-17

Now that you have determined the different kind of devices you want to support and have defined device filters for these devices, the next step is to define override properties for specific device filters. Defining property overrides is as simple as clicking the `PropertyOverrides` property of the Mobile Web control, selecting the device filter from the list, and providing the property value specific to the device filter.

Listing 28-13 shows a simple Mobile Web Form containing three controls. We want the Label and Command controls to display longer versions of the text if the application is accessed by an IE browser on a Pocket PC. We have already defined a device filter that checks to see if the incoming browser is Pocket IE.

Start by selecting the Label control on the Designer window and clicking the ellipsis (...) button next to the `PropertyOverrides` property. In the Property Override window, select the Pocket IE filter from the drop-down list and provide a longer version of the `Text` property to this Label control. Repeat these same steps for the Command control. Figure 28-18 shows the Property Override window for both controls. The title bars on these Windows show the selected control whose properties are being overridden.

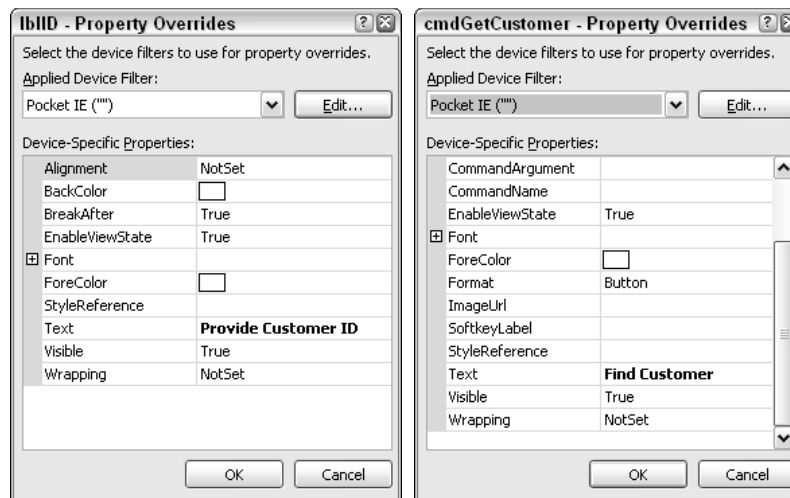


Figure 28-18

## State Management in ASP.NET Mobile Applications

Mobile Web Forms provide elegant ways for managing user and page state. These state management mechanisms work in mobile applications in several ways, as the following sections explain.

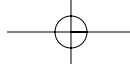
### ViewState in Mobile Web Controls

You already know that ASP.NET Web Forms are capable of maintaining their own state across multiple postbacks. They can do this because of the built-in support for ViewState. ViewState makes a Web Form's lifetime span multiple roundtrips to the server.

The ViewState in ordinary Web Forms is managed by the Web server using a hidden field in the form. This hidden field contains encoded ViewState data that gets submitted to the Web server every time the Web Form posts back.

However, ViewState in Mobile Web Forms functions in a completely different manner. The main goal for the mobile application is to avoid excessive network traffic due to limited bandwidth available on many mobile devices. As a result, ASP.NET does not send a page's ViewState to the client and, instead, stores it as part of a user's session on the server. A hidden field is still contained in the Mobile Web Form, but it contains an only identifier for the page's ViewState stored on the server.

You can imagine how easy it can be for the current ViewState to be out of synchronization with the current page displayed on the browser, especially if the user uses the Back button on the browser to go back in the history. Suppose you go to Page A, click a button to go to Page B, and then press the Back button to return to Page A. The current page displayed on your browser is now Page A, but the current state on the server is that of Page B.



## Chapter 28

---

ASP.NET Mobile Web Forms solve this problem by maintaining a history of ViewState information in the user's session. The identifier sent to the client corresponds to a position in this history. In the previous example, if you again post from Page A, the Mobile Web Form uses the identifier saved with Page A to synchronize the history.

You can configure the size of this history in order to tune your application. The default size is 6 and can be changed by adding a numeric attribute to a tag in the `web.config` file, as shown in the following code:

```
<configuration>
  <system.web>
    <mobileControls sessionStateHistorySize="8" />
  </system.web>
</configuration>
```

Because the ViewState is stored in the user's session, it is possible for it to expire if a page does not post back within the session expiration time. In such cases, the `OnViewStateExpire` method of the page is called. The default implementation of this method throws an exception indicating that the ViewState has expired. However, if you are able to restore ViewState manually after expiration, you can override this method at the page level and not call the base implementation.

Even though storing of ViewState in session has the advantage of reduced network traffic, it can also lead to poorer performance. It is usually a best practice to turn off ViewState on the controls when you don't need to retain the information. Disabling ViewState is as simple as setting the `EnableViewState` property on the control to `False`. You can also disable the ViewState for an entire page by adding the `EnableViewState="false"` attribute to the `@Page` directive. You should, however, be aware that some mobile controls save essential state information, such as active form, across client roundtrips even when the ViewState is disabled.

Mobile Web applications do not include a mobile control for writing out hidden variables. Instead, the Mobile Form provides a collection called `HiddenVariables` inside the `MobilePage` class that you can use to specify hidden variables. All name/value pairs stored in this collection are persisted as hidden variables. The `HiddenVariables` collection is automatically repopulated with these hidden variables when the form is submitted.

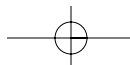
### Managing Session State

The session management in ASP.NET functions the same way for mobile applications. It is scalable, robust, and can be used across Web farms. You are provided with a session object that you can use to save information about a user session across multiple requests.

The default behavior of the session management features of ASP.NET require the server to write out a session cookie to a client. The client submits the cookie on each request during the session, and the server looks up the Session State from this information. However, many mobile browsers do not support cookies. In such cases, the session management and the ViewState management require you to configure the application to use a `cookieless` session. `Cookieless` session management automatically inserts the session key in the application's URL.

### Hidden Fields

ASP.NET 2.0 provides an elegant way of using hidden fields on a mobile Web application. Instead of providing a Mobile Web control that could be used for storing hidden information, ASP.NET 2.0 provides a collection object for every Mobile Web Form. This collection is called `HiddenVariables` and it is used to store



key/value pair information. This information is automatically stored in hidden fields when the Mobile Web page is rendered. When the page is posted back, either to the same page or to a different page, ASP.NET 2.0 retrieves the hidden information from the page and restores the `HiddenVariables` collection. You can simply access the fields from within this collection without being concerned with the behind-the-scene details.

The code in Listing 28-14 shows a simple mobile Web application that lets the user add hidden fields by simply typing them on the screen and clicking the Add Hidden Fields button. The user can also view all the hidden fields by clicking the Show Hidden Fields button.

### Listing 28-14: Ways to use hidden fields in mobile Web applications

#### ASPX Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="HiddenVariable.aspx.vb"
    Inherits="HiddenVariable" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <mobile:Label ID="lblKeyName" Runat="server">
            Provide Key Name</mobile:Label>
        <mobile:TextBox ID="txtKeyName" Runat="server">
        </mobile:TextBox>
        <mobile:Label ID="lblVariable" Runat="server">
            Provide Variable Value</mobile:Label>
        <mobile:TextBox ID="txtHiddenField" Runat="server">
        </mobile:TextBox>
        <mobile:Command ID="cmdAddHidden" Runat="server"
            OnClick="cmdAddHidden_Click">
            Add Hidden Fields</mobile:Command><br />
        <mobile:Command ID="cmdShowHidden" Runat="server"
            OnClick="cmdShowHidden_Click">
            Show Hidden Fields</mobile:Command><br />
        <mobile:List ID="lsHiddenFields" Runat="server">
        </mobile:List>
    </mobile:Form>
</body>
</html>
```

#### VB

```
Partial Class HiddenVariable
    Inherits System.Web.UI.MobileControls.MobilePage

    Protected Sub cmdAddHidden_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles cmdAddHidden.Click
        Me.HiddenVariables.Add(txtKeyName.Text, txtHiddenField.Text)
    End Sub

    Protected Sub cmdShowHidden_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles cmdShowHidden.Click
        lsHiddenFields.DataSource = Me.HiddenVariables.Values
        lsHiddenFields.DataBind()
    End Sub
End Class
```

(continued)

## Chapter 28

### Listing 28-14: (continued)

```

C#
using System;
using System.Collections;
using System.Web.Mobile;
using System.Web.SessionState;
using System.Web.UI.MobileControls;

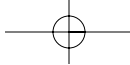
public partial class HiddenVariable : System.Web.UI.MobileControls.MobilePage
{
    protected void cmdShowHidden_Click(object sender, EventArgs e)
    {
        lsHiddenFields.DataSource = this.HiddenVariables.Values;
        lsHiddenFields.DataBind();
    }
    protected void cmdAddHidden_Click(object sender, EventArgs e)
    {
        this.HiddenVariables.Add(txtKeyName.Text, txtHiddenField.Text);
    }
}

```

Figure 28-19 shows the result of executing the code shown in Listing 28-14.



Figure 28-19



## Summary

You should now have the knowledge to be successful with your mobile Web development projects. You have learned how you can use various feature-rich mobile Web controls. These controls encapsulate all the details related with rendering to a mobile device by providing an easy-to-use programming interface. The Web Control architecture for Mobile Web Controls enables you to reuse all your business logic and data access code from the ASP.NET Web application, hence providing a highly productive programming environment.

This chapter showed you how to create device-specific displays. The fast-evolving mobile device industry produces new devices with more capability at a more rapid pace than ever before. Programmers needed the capability to customize the rendering of a Mobile Web Control to a specific device to take full advantage of that device's capability. The device filters support provided for mobile Web controls now gives you this capability.

You also learned how you can effectively manage Session State and ViewState. It's important for you to understand how Session State and ViewState are handled differently in Mobile Web applications so that you can take full advantage of these features.

