
.Developing .ASP NET .Mobile .Web .Applications

.Sudhanshu .Singh

ASP.NET Mobile Web Development Overview

Developing ASP.NET pages for mobile device browsers does not differ substantially from developing pages for desktop browsers. To help you create applications for mobile devices, ASP.NET provides a **System.Web.Mobile** namespace devoted specifically to mobile Web development.

You can create a Web page from the **MobilePage** base class and add controls from the **System.Web.Mobile** namespace. This namespace defines a suite of Web server controls and adapters that are especially useful when creating applications that need to be available to many different mobile devices, such as cell phones.

ASP.NET 2.0 also provides a control-adaptive architecture that allows custom device adapters to be created for ASP.NET 2.0 Web server controls. The adapters can create custom rendering for a control based on the requesting browser. With the adaptive architecture, you can create custom adapters for ASP.NET Web server controls to render output specific to the devices that access your application on desktop browsers.

Whether developing for desktop browsers or mobile devices, development follows the standard .NET event-driven model in which your application responds to user requests, button clicks, and so on.

Mobile Application Architecture

Although ASP.NET integrates technology to make ASP.NET mobile Web application development follow the same paradigm as traditional Web application development, the architecture's primary intent is not to allow you to create single pages that can target browsers in both desktop and mobile devices. The limitations of browsers on mobile devices often mean that pages designed for desktop browsers cannot translate to mobile device browsers.

For example, if you create an ASP.NET Web page that includes a site header, a navigation bar along the top of the page, a secondary navigation structure along the side of the page, and then content in the rest of the page, the page will render as designed in a desktop browser. In that case, there is usually ample space to render all the controls and still provide a scrollable content area. However, in many mobile device browsers, this layout would be impossible. Many mobile devices have a smaller screen area than desktop monitors, so even navigation becomes a multi-step process in which the user must click several controls to get to page content.

Presentation logic follows a similar pattern. For example, when the user fills in a Web form using a desktop browser, the user can see many controls on the screen at once. When the form is validated on the server, validation errors can be displayed next to the controls. With a mobile device, form input and validation can be much harder to display in a format that is usable. Additionally, for mobile devices you might choose to provide shortcuts that allow the user to fill in information with less typing because the device might be difficult to type on.

For these reasons, you must create separate pages in your ASP.NET Web application for use in desktop and mobile device browsers. A page developed specifically for mobile device browsers allows you to break down presentation logic into smaller pieces that work better for the device's display area and input hardware.

Mobile Web Server Controls

The ASP.NET 2.0 **System.Web.Mobile** namespace is devoted specifically to mobile Web development. You create a mobile Web page from the **MobilePage** base class and add mobile Web server controls from the **System.Web.Mobile** namespace. Mobile Web server controls have a number of specialized adapters in the .NET Framework and are therefore especially geared to developing mobile Web applications that target a wide range of mobile devices.

ASP.NET Web Server Controls and the Adapter Architecture

Most ASP.NET 2.0 Web server controls adhere to the unified adapter architecture. This means that all controls can behave differently depending on the requesting device by calling a custom adapter that provides appropriate behaviors for that device, such as creating the proper markup language. If an adapter is configured in the browser definitions file for the requesting device or browser, ASP.NET calls the adapter at each life-cycle stage of a Web server control. The adapter can then adjust rendered output for example, and handle any device-specific view state logic or device idiosyncrasies. Browser definition files can be found in the Browsers folder of the .NET Framework Config directory or in the App_Browsers folder of a Web application.

There are currently no adapters provided for the ASP.NET controls. There are, however, a rich set of adapters for the ASP.NET mobile controls to provide for a wide variety of devices and browsers. You can create custom adapters for each device and have the ASP.NET page framework use those adapters when a specific device accesses your page.

Choosing Custom Adapters or Mobile Controls

For pages targeting mobile devices, you must use mobile Web server controls and create pages that inherit from **MobilePage**. These controls support many mobile devices, such as cell phones. ASP.NET includes mobile Web server controls for a wide variety of general Web development and mobile-specific needs. Additionally, mobile-control device adapters already exist for major devices and their markup languages.

Microsoft will continue to provide adapter updates for the mobile Web server controls when major markup languages evolve. This enables you to support new markup languages with the same controls that you are already using. For example, if you are creating an e-commerce site that supports desktop browsers as well as a large array of mobile devices, you would create a set of ASP.NET pages that inherit from the **Page** class, and a separate set of pages that inherit from the **MobilePage** base class and use mobile controls.

If you need to, you can create your own custom adapters or modify existing ones where new devices dictate new behavioral requirements in mobile Web server controls.

There are scenarios where using ASP.NET Web server controls and writing custom adapters makes sense. These typically will be applications for rich desktop browsers where browser behavior variations are required, or for applications to be targeted by a constrained device class for which mobile controls and their feature set is not warranted. One example might be if you were creating an insurance-claim application that had a browser-based interface for office use and a rich-device interface for field use. Your application could

then use the same base page classes for both the regular pages and the rich-device pages. You would then need to create custom adapters only for the device that was deployed in the field.

ASP.NET Device Filtering Overview

You can use device filtering to customize certain rendering aspects of Web server controls depending on the browser or device that accesses them. When a user requests a Web page from a server, the user's browser makes a request that contains information — such as the user-agent and other headers — that identifies the browser's type and version. ASP.NET can then match the identifier to a particular device that is defined in a browser file. And then output can be filtered by device by using the identifier in Web server controls.

Device Filtering

The following declarative code example demonstrates use of a device filter to shorten the text property of a **Label** control for a Pocket PC running Pocket Internet Explorer. This is a common use of a device filter, where more succinct text is provided for a device with limited screen size. The prefix "PIE" in front of the second **Text** attribute specifies that the control should render that version of the text if the requesting browser's identifier is "PIE".

```
<asp:Label runat="server" id="title" Text="Welcome to Our Online Shopping Catalog" PIE:Text="Welcome, Shopper" />
```

Control Filters

You can filter output on controls for different devices by applying filters to the following:

- Control properties
- Custom attributes
- Templates

Device Filters for Directives

You can also apply device filters to **@ Page** directive attributes to better suit device capabilities. For instance, you can disable view state for certain devices, or use different themes based on the device that is accessing the page. Some of the **@ Page** directives you can filter are:

- **Buffer**
- **ClientTarget**
- **CodePage**
- **ContentType**
- **Culture**
- **EnableViewState**
- **EnableViewStateMac**
- **ErrorPage**
- **LCID**
- **MasterPageFile**
- **ResponseEncoding**

- **Theme**
- **UICulture**

If you are working with user controls, you can apply device filters to **@ Control** directive attributes as well. In general, the **@ Control** directive offers fewer attributes for which device filtering makes sense, but you could apply it to an attribute such as **EnableViewState**.

Finally, you can apply device-filters attributes, which are used to specify properties of a master page, in the **@ Master** directive.

Note

You cannot specify a device filter in the Web.config file.

How to: Work with Emulators and Browsers

ASP.NET mobile controls enable you to develop applications for a wide variety of mobile 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 pages as they might appear on the manufacturers' hardware devices, and to experience the interface for your Web site as users do. For example, after you see how a user must navigate through your site on a specific device, you might want to modify the interface using a **DeviceSpecific** template for that device.

Developing and testing with emulators enables you to test your mobile Web application more easily on a variety of devices before you deploy it.

There are two approaches to viewing your mobile Web pages on device emulators:

- Install and use device emulators provided by manufacturers.
- Use the emulators installed with some editions of Visual Studio. To use this option, you must have an edition of Visual Studio containing Emulator Manager, and you must have installed the ActiveSync application, which you can download from the tools page on the Microsoft Mobile Developer Center.

Adding an Emulator to Visual Studio

You can add an emulator to the list of browsers available in Visual Studio.

To add a device manufacturer's emulator to the list of available browsers:

1. Compile your application.
2. Install the mobile device emulator on your development computer. See the emulator's documentation for instructions.
3. In the **File** menu, click **Browse With**. Visual Studio displays the **Browse With** dialog box.
4. Click **Add**. Visual Studio displays the **Add Program** dialog box.
5. In the **Program name** box, enter the name of the emulator's executable program file.

6. If the emulator accepts command line arguments, enter them in the **Program name** field. For example, enter %startpage to specify where the application's start page should be substituted in the command line.
7. In the **Friendly name** box, enter the name of the browser as you want it to appear in Visual Studio.
8. Click **OK**.
9. If you want the emulator to be the default browser, click **Set as Default**.
10. Click **Close**.

Removing an Emulator

If you no longer need an emulator, you can remove it from Visual Studio.

To remove an emulator from the browser list:

1. In Solution Explorer, right-click the name of any .aspx file.
2. In the shortcut menu, click **Browse With**. The designer displays the **Browse With** dialog box.
3. Select an emulator from the browser list.
4. Click **Remove**. Visual Studio removes the emulator name from the browser list.

Note

You cannot remove a browser designated as the default browser.

Testing Pages with Emulator Manager

All editions of Visual Studio include support for mobile Web pages. If your edition of Visual Studio also includes support for mobile device applications (such as for the Pocket PC), you have Emulator Manager installed, which includes several emulators. However, the emulators available with Emulator Manager were not originally intended for testing mobile Web pages. To use Emulator Manager to test your mobile Web pages, you can install ActiveSync, which is available for download from the tools page on the Microsoft Mobile Developer Center.

To view your Web site with an emulator using Emulator Manager:

1. Compile your application.
2. In the **Tools** menu, click **Emulator Manager**.

Note

If the **Emulator Manager** command is not available, you do not have the Compact Framework installed.

3. Select a device emulator.
4. In the **Actions** menu, click **Connect**.

The emulator appears. Move the emulator so you can see Emulator Manager, and wait for it to indicate that the selected device is connected.

5. In Emulator Manager, right-click the connected device emulator in the list and in the shortcut menu, click **Cradle**. ActiveSync starts.
6. In the **Set Up a Partnership** dialog box, select **Guest partnership** and click **Next**.
7. When ActiveSync indicates the device is connected, close ActiveSync. (It will continue running in the background.)
8. In the emulator, navigate to your Web site.

 Note

An emulator might not be able to use the localhost URL to access your Web site project. If not, you can view the Web site using the intranet URL.

Debugging Web Pages in the Emulator

If Visual Studio cannot launch an emulator when debugging a Web site, you can debug your application by attaching to the ASP.NET worker process.

To debug your Web site application by attaching to the worker process:

1. Set a breakpoint in the code you want to debug.
2. Compile your application.
3. On the **Tools** menu, select **Attach to Process**.
4. In the **Available Processes** list, select the Web site worker process (w3wp.exe or aspnet_wp.exe).
5. Click **Attach**. The Visual Studio debugger starts.
6. In the emulator or browser, navigate to your Web site project. The debugger stops at the first breakpoint.

For more information about using an emulator, see the manufacturer's emulator documentation.

ASP.NET Mobile Web Pages

The Microsoft Visual Studio integrated development environment (IDE) enables you to easily build ASP.NET applications that include mobile Web pages. You can include mobile Web pages in any ASP.NET Web site alongside ASP.NET Web pages. In Visual Studio, you can work with the adaptive rendering, customization, and extensibility features of ASP.NET mobile controls, using the standard IDE design tools: the page designer, the Toolbox, the debugger, Source view, Design view, and more.

To start creating an ASP.NET mobile Web page, open an existing ASP.NET Web site project in Visual Studio 2005, or create a new Web site project. Create a new mobile Web page (Web Form) and drag a mobile control from the **Mobile Web Forms** tab in the Toolbox. You can specify properties and event handlers for the control using the Properties window. Use standard Visual Studio functionality to build and test your application.

Because ASP.NET automatically adapts the rendering of your mobile page to different devices, you build your application by logically grouping controls and arranging them to match the desired user experience. A difference from designing ASP.NET pages for desktop browsers is that you cannot resize mobile controls manually. Instead, ASP.NET resizes controls when it generates the appropriate markup. To see how the application renders on a specific device, view it on an emulator for the device or on the actual device.

Design view displays a representation of pages. It does not emulate the rendering of any specific device. As you develop your pages, Design view provides you with visual cues that indicate the current property settings of mobile controls. However, the page might not appear at run time exactly as you see it in Design view. The target device might not support every control property that you have set, or it might support the property, but not the setting you specify. In addition, some properties are provided strictly for extensibility. For example, most of the controls have a **BackColor** property, but only the **Form** control currently uses it. The mobile controls enable you to develop controls that use the **BackColor** property. Developers writing custom device adapters can use this property while rendering controls.

You can optimize the markup generated by a mobile control for a specific device. Mobile controls provide powerful tools that enable you to customize the application's output for specific devices by overriding `property_values` and by creating a specialized rendering based on device capabilities.

The extensibility model of mobile controls enables new device support to be added without requiring that the Web application be modified. You can add support for new devices by updating configuration file settings or by deploying new device adapters. This greatly increases the lifespan of your applications because they continue to work with the latest devices.

Creating ASP.NET Mobile Web Pages

ASP.NET mobile controls enable you to target a wide range of devices, including Web-enabled cell phones, pagers, and personal digital assistants (PDAs) such as the Pocket PC. ASP.NET provides the same rich Web application model for mobile devices that it does for ASP.NET applications that target desktop

browsers. This section of the documentation describes the extensions that the ASP.NET mobile controls add to ASP.NET Web pages.

ASP.NET mobile controls consist of ASP.NET server controls and device adapters that can intelligently render pages and controls. Knowledge of ASP.NET helps you use mobile controls to build mobile Web pages.

ASP.NET mobile controls also extend the schema of the **Machine.config** file and add elements to support mobile device rendering. ASP.NET provides an extensible model so that third parties can add new controls and add support for new devices.

You can build mobile Web applications using Microsoft Visual Studio or a text editor, and program them using any programming language supported by the common language runtime. The .NET Framework and the ASP.NET mobile controls together form a powerful, flexible, and extensible platform for developing and deploying mobile Web pages.

Inside the ASP.NET Mobile Controls

ASP.NET provides controls, components, and tools to help you build mobile Web pages rapidly for many types of devices — without having to write code that is targeted for a specific device. ASP.NET mobile Web pages can recognize a variety of mobile devices and render markup appropriately for them.

You can also create new mobile controls as user controls. Device manufacturers or independent software vendors (ISVs) can add support for new devices. By writing their own control adapters, developers can customize controls to take advantage of unique features on specific devices.

Extensibility

ASP.NET mobile Web pages and mobile controls offer the same extensibility features available for ASP.NET pages and server controls, but add support for working with multiple devices. Specifically, ASP.NET mobile Web pages and mobile controls provide the following extensibility:

- You can use ASP.NET user controls to write simple mobile controls declaratively.
- You can customize the output of any control for a specific device by adding a new adapter for the control.
- You can write new mobile controls and use them in ASP.NET mobile Web pages. New controls can use inheritance or composition to take advantage of existing controls.
- You can add support for an entirely new device by using adapter extensibility, with no changes to individual applications.

ASP.NET Mobile Controls

You can create ASP.NET mobile Web pages in Visual Studio or with any text editor. Many of the controls are similar to ASP.NET Web server controls. For example, the **System.Web.UI.MobileControls.Label** and **System.Web.UI.MobileControls.TextBox** controls mimic the behavior of the ASP.NET **System.Web.UI.WebControls.Label** and **System.Web.UI.WebControls.TextBox** controls.

Getting Started with ASP.NET Mobile Controls

ASP.NET mobile controls are a set of ASP.NET server controls that generate Wireless Markup Language (WML), compact HTML (cHTML), HTML, and XHTML content for different devices. This section provides information about device simulators, software requirements, and related topics.

ASP.NET Mobile Web Forms and ASP.NET Compatibility

When you create your ASP.NET mobile Web Forms pages, you can use nearly all the features of ASP.NET. However, first consider compatibility issues.

Error Handling and Reporting

When an ASP.NET application encounters an unhandled exception or other error while processing a request, it generates an error page. Exceptions can occur at any time during the processing of a request. For example, they might occur when reading a configuration file (Web.config), compiling a page, or running a page.

You can configure your application to generate default or custom error pages. If you configure your application for default error pages, ASP.NET sets an error code in the response and renders a page that describes the error in detail. However, if you configure your application for custom error pages, each error request is redirected to a custom page that you provide for it.

Many mobile devices cannot render the detailed contents of an error page. Instead, such devices usually show only a device-specific error message, or the error code. To address this situation, ASP.NET mobile Web Forms pages attempt to format the error page so that it renders on the device. However, this device-specific rendering is limited to exceptions that occur while running the page. Therefore, if you are using default error pages, you should first try your mobile Web Forms page from a desktop browser to detect potential configuration or compilation errors.

If you plan to use custom error pages in your ASP.NET mobile Web application, ASP.NET can format the error page appropriately for different mobile devices if you write your custom error pages using mobile controls.

Tracing

ASP.NET provides an easy-to-use functionality called tracing that you can use to debug your Web applications. ASP.NET provides two levels of tracing, page-level and application-level tracing. Page-level tracing provides trace information as HTML code that is appended to each traced page, whereas application-level tracing provides trace information through a special mapped URL (Trace.axd) in the application.

If you use page-level tracing in your ASP.NET mobile Web application, the HTML code appended to the rendering might prevent the output from being rendered on the mobile device. Instead, for ASP.NET mobile Web applications, you must use application-level tracing and inspect the trace output from a desktop Web browser.

Session State and Cookies

ASP.NET provides rich session management features that allow you to easily maintain state across requests. Normally, the ASP.NET session state facility uses cookies on the browser, but it can be configured to work without cookies.

In ASP.NET, you can use the **Session** to save information about a user session across multiple requests. Session management in ASP.NET is scalable and robust so that you can use it even across Web farms. By default, the ASP.NET **Session** uses a client-side cookie to store an identifier on the client's computer. You can use the identifier to locate a session across server round trips. In addition, the ASP.NET **Session** supports a cookieless session mode that initially redirects a client to a new URL that contains a session's identifier. The session identifier is then automatically parsed out of the URL.

When writing an ASP.NET mobile Web application, you must keep in mind that some mobile devices and wireless gateways do not support cookies. To add support for these devices, you must configure your application to use cookieless sessions.

Considerations When Using Session State

When writing an ASP.NET mobile Web application that uses session state management, consider the following factors:

- Using ASP.NET controls in the **System.Web.UI.WebControls** namespace on a mobile Web Forms page is not supported. Mobile Web Forms pages that use mobile Web controls in the **System.Web.UI.MobileControls** namespace support setting the **EnableSessionState** attribute of the **@ Page** directive to **false**. However, mobile Web Forms pages that use an ASP.NET control from the **System.Web.UI.WebControls** namespace with **EnableSessionState** set to **false** might lead to a compile-time error.
- Some mobile devices and gateways do not support cookies. To enable an ASP.NET mobile Web Forms page to run on these devices, set the **cookieless** attribute of the **sessionState** element to **true**.
- Some mobile devices have problems dealing with relative URLs after they have been redirected through the technique employed by cookieless session management.

For example, if an Openwave browser opens an .aspx file at `http://localhost/a.aspx`, and the Web site redirects the browser to `/12345678/a.aspx`, the browser still considers its current path as the root. The browser will request a subsequent relative reference to `b.aspx` as `/b.aspx`.

The solution is to include a rooted URL on the page, such as `/12345678/a.aspx`, instead of a relative URL when rendering after a redirect. The built-in ASP.NET mobile controls automatically do this, but any newly written controls or adapters must include code that handles rendering after a redirect. Both the **MobilePage** and the adapter base classes have methods, such as **MakePathAbsolute**, that help a mobile control developer write rooted URLs.

Using Redirects

Some devices and browsers currently require fully qualified URLs in response to an HTTP redirect. Set the **useFullQualifiedRedirectUrl** attribute of the **httpRuntime** element to **true** in the **System.Web** section of either the **Machine.config** file or **Web.config** file (at the application level).

Syntax Issues

Syntax that is valid in ASP.NET, for example, **<%=**, is not valid in ASP.NET mobile controls, and must be replaced by data-binding mechanisms.

Data-binding expressions must be delimited by **<%#** and **%>**. The following is an example of the use of data-binding expressions.

```
<%# binding expression code goes here %>
```

Comparing Web Controls and Mobile Controls

ASP.NET Mobile Web pages are based on ASP.NET Web Pages. ASP.NET mobile controls provide a flexible toolset that enables you to create content sites and Web applications intended for a wide variety of mobile devices. You can take advantage of the adaptive rendering of ASP.NET mobile controls, while having the flexibility to customize the display for specific devices or types of devices, such as a handheld computer or a mobile phone.

The following table provides a side-by-side comparison of ASP.NET Web server controls and ASP.NET mobile controls.

Web server control	Mobile control	Comments or differences
AdRotator	AdRotator	Similar functionality. Mobile control adds ImageKey and NavigateUrlKey properties.
Button, ImageButton, LinkButton	Command	Mobile control combines the functionality of the Web server Button , ImageButton and LinkButton controls.
Calendar	Calendar	Similar functionality. The mobile control does not provide HTML-specific properties directly, but exposes an underlying Web server Calendar control through the WebCalendar property.
[no equivalent control]	PhoneCall	Used to actively drop the data line and initiate the call on dial-capable devices. This is similar to the mailto: protocol for electronic mail addresses, which starts an e-mail client.
CompareValidator	CompareValidator	Same functionality.
CustomValidator	CustomValidator	Same functionality.
DataList, Repeater	List	Similar functionality. Mobile control can apply

		templates on a per-device basis.
DataGrid	ObjectList	Similar functionality. The ObjectList control provides multiple views to show the data collections
[no equivalent control]	DeviceSpecific	Used to enable property overrides and templates for mobile controls.
[no equivalent control]	Form	Similar to a page in an ASP.NET Web application. Mobile Web pages can contain multiple Form controls.
Image	Image	Similar functionality. The mobile control can select an image from a set of device-specific images.
Label	Label	Same functionality.
HyperLink	Link	ASP.NET cannot render the mobile control as an image. Use the Image control to create an image link (by specifying the NavigateUrl property on the Image control).
Panel	Panel	Mobile panel controls can contain a DeviceSpecific control to display templates of the DeviceSpecific control rather than the panel.
RangeValidator	RangeValidator	Same functionality
RegularExpressionValidator	RegularExpressionValidator	Same functionality.
RequiredFieldValidator	RequiredFieldValidator	Same functionality.
CheckBox, CheckBoxList, DropDownList, ListBox, RadioButton, RadioButtonList	SelectionList	Mobile control combines the functionality of the corresponding ASP.NET Web server controls. Use the SelectType property (and the associated ListSelectType enumeration) to define the type of selection list button to render. For example, setting the SelectionList control's SelectType property to the CheckBox enumeration corresponds to the ASP.NET Web server controls CheckBox and CheckBoxList ; DropDown is the same as DropDownList . Use the Rows property to specify the number of items shown in the list when the SelectType property is the ListBox or MultiSelectListBox control.
IStyleSheet	StyleSheet	ASP.NET Web pages use cascading style sheets rather than StyleSheet controls.
Table	[no equivalent control]	Use the List , ObjectList , and SelectionList mobile controls
TextBox	TextBox	Similar functionality. The mobile control does not provide automatic postback, read-only, or multiline functionality.
[no equivalent control]	TextView	Used to display large blocks of text. Supports basic text formatting.

ValidationSummary	ValidationSummary	Same functionality. The mobile control shows validation error messages on a separate form (through the FormToValidate property).
-------------------	-------------------	---

ASP.NET Namespaces for Mobile Controls

Microsoft ASP.NET provides three namespaces that are used to implement the run-time and design-time behavior of mobile components and controls. These namespaces include the fundamental interfaces and base classes for implementing attributes, classes, controls, and elements. Listed below are the namespaces in ASP.NET for mobile controls and the classes that constitute them:

- **System.Web.Mobile.** Core capabilities, authentication, and error-handling classes. For more information, see the **MobileCapabilities** and **MobileFormsAuthentication** classes.
- **System.Web.UI.MobileControls.** Core ASP.NET mobile control classes. For examples, see the **IObjectListFieldCollection** interface, **ITemplateable** interface, **AdRotator** class, and **DeviceSpecific** class.
- **System.Web.UI.MobileControls.Adapters.** Core adapter classes that you can implement to create adapters for targeted devices.

Device Adapter Code

To help you create adapters for new mobile devices, or to create your own modifications for the XHTML adapter set, you can download the source code for the adapter from the **ASP.NET Mobile Controls XHTML Adapter Source** page on the Microsoft Download Center.

The adapter code can be compiled into its own assembly. You can compile the adapter code by running the following command in the directory that contains the device adapter code. The example assumes that the **csc** command is in your computer's **PATH** environment variable.

```
C#  
csc /target:library /out:System.Web.UI.MobileControls.ShippedAdapterSource.dll /r:System.Web.Mobile.dll  
/debug+ /D:COMPILING_FOR_SHIPPED_SOURCE /nowarn:0679 *.cs
```

To use the custom adapter, update the **<mobileControls>** section in the root Web.config file.

Developing ASP.NET Mobile Web Pages

This section provides information about basic ASP.NET mobile controls as well as design concepts for mobile devices.

Introduction to ASP.NET Mobile Web Pages

This section of the documentation introduces you to designing ASP.NET mobile Web pages using ASP.NET mobile controls. The mobile controls are built on and extend the technology of the Microsoft .NET Framework.

Authoring Tools

To create ASP.NET mobile Web pages, you can use Microsoft Visual Studio or you can use a text editor. Visual Studio provides tools for creating mobile Web pages and its code, and for managing the application containing your mobile Web pages.

After you create your mobile pages, you can view them using a browser on a supported device.

Server-Side Applications

Mobile controls run on the server. They send markup language to the browser; the markup specifies how to display the controls and content in the current form or page.

Each mobile Web page contains at least one form element, indicated by the **<mobile:Form>** tag. Every mobile control tag must include the **runat=server** attribute.

Client Script

Like other types of Web pages, mobile Web pages can contain client script for the browser to process. If these scripts refer to specific Web server controls, they must use the identifier emitted in the markup language. These identifiers vary depending on what markup language the device supports. To obtain the exact name of the control, compile the application, browse to the page or form, and view its source markup.

Traditional "Hello World" Example

The following code example illustrates a "Hello World" mobile Web page. The example demonstrates using a Form control as a container for a Label mobile control that displays the text "Hello, world!"

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>
<mobile:Form id=Form1 runat="server">
  <mobile:Label id=Label1 runat="server"> Hello, world! </mobile:Label>
</mobile:Form>
```

International "Hello World" Example

The mobile community is worldwide. The following code example illustrates an international version of "Hello World". In this variation of the preceding example, a List control displays a list of languages as defined in individual **<Item>** elements. An event handler reads the language selected by a user and then switches to another form. A **switch** statement is used to display the correct text for the user's selected language.

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>

<script runat="server">
public void MyEventHandler(Object source, ListCommandEventArgs e)
{
    Selection.Text = "You selected " + e.ListItem.Text;
    ActiveForm = SecondForm;
    switch (e.ListItem.Text)
    {
        case "French": Selection.Text = "Bonjour le monde";
                        break;
        case "German": Selection.Text = "Hallo Welt";
                        break;
        case "Italian": Selection.Text = "Ciao il mondo";
                        break;
        case "Norwegian": Selection.Text = "Hei verden";
                        break;
        case "Portuguese": Selection.Text = "Oi mundo";
                        break;
        default: Selection.Text = "Hello World";
                break;
    }
}
</script>

<mobile:Form id="ListStuff" runat="server" BackColor="White" ForeColor="#bb7023">
    <mobile:Label runat=server id="label"> Pick a Language! </mobile:Label>
    <mobile:List runat=server id="ListMe" OnItemCommand="MyEventHandler">
        <item Text="English" />
        <item Text="French" />
        <item Text="German" />
        <item Text="Italian" />
        <item Text="Norwegian" />
        <item Text="Portuguese" />
    </mobile:List>
</mobile:Form>
<mobile:Form id="SecondForm" runat="server" BackColor="White" ForeColor="Green">
    <mobile:Label runat=server> Your "Hello World" Translation </mobile:Label>
    <mobile:Label runat=server id="Selection"></mobile:Label>
    <mobile:Link runat=server id="GoBack" NavigateURL="#ListStuff">back</mobile:Link>
</mobile:Form>
```

Changing the Text Encoding for an International Application

International applications often require you to change the character encoding from the default UTF-8 encoding. To change the text encoding, use the globalization element, as in the following example, which sets the encoding to UTF-16:

```
<globalization> requestEncoding="utf-16" responseEncoding="utf-16" />
```

You can set the encoding in either the global Machine.config file, which specifies the encoding for all of your applications, or in the application's Web.config file, which specifies the encoding for only that application.

Life Cycle of an ASP.NET Mobile Web Page

The life cycle of a Microsoft ASP.NET mobile Web page and its controls is similar to the life cycle of a standard ASP.NET Web page. The following section describes similarities and differences between them.

Note

The life cycle of an ASP.NET mobile control is the same as the life cycle of a mobile Web page.

Life Cycle Stages for a Mobile Web Page

The following table lists life-cycle stages of a mobile Web page and information about its controls. This information primarily describes the differences in the life cycle of a mobile Web page or mobile control from those described for a non-mobile ASP.NET page.

ASP.NET page life-cycle stage	Mobile page life-cycle stage	Methods or events to override
Initialize	Device adapters are chosen using the mobileControls element in the Web.config file. Device-specific customizations are applied.	OnInit method OnInit method
Load view state	Same as non-mobile ASP.NET pages.	LoadViewState method
Process postback data	Same as non-mobile ASP.NET pages.	LoadPostData method
Load	The MobileControl base class instance calls the control's OnLoad method to load device adapter-specific information.	OnLoad method OnLoad method
Send postback change notifications	Same as non-mobile ASP.NET pages.	RaisePostDataChangedEvent method
Handle postback events	Same as non-mobile ASP.NET pages.	RaisePostBackEvent method
Pre-render	Pagination is performed. The number of items on the page is determined, unless a virtual count is specified through the ItemCount property.	ItemWeight property OnPreRender method
Save state	Same as non-mobile ASP.NET pages.	SaveViewState method
Render	The adapter accesses and renders child controls in the appropriate order. The ASP.NET page framework renders each control by calling the Render method of its adapter.	Render method Render method
Unload (Dispose)	Device adapter-specific cleanup and unloading is performed.	Dispose method OnUnload method

Design and Rendering Concepts for ASP.NET Mobile Controls

Designing pages for mobile devices differs from designing pages for traditional Web sites, because pages for mobile devices break content into groups of data that can be presented in a linear manner. This section describes control containers and techniques that you can use to render content for ASP.NET mobile controls.

There are three primary container controls available for mobile controls: the **MobilePage**, **Form**, and **Panel** controls. The rules for using these containers are:

- Each ASP.NET mobile Web page creates an instance of a **MobilePage** control.
- A mobile Web page contains one or more **Form** controls.
- A **Form** control can contain zero or more **Panel** controls.

- **Panel** controls can contain other **Panel** controls.
- Any number of other controls can be in a **Form** control.
- Any number of other controls can be in a **Panel** control.

In This Section

Forms: Provides the criteria to use when organizing mobile controls on forms.

Pages: Contains information about mobile Web Forms pages and when to use them.

Pagination: Provides a detailed look at the types of pagination in mobile controls.

Panels: Describes the function of panels in relation to forms.

Styles: Suggests guidelines for designing mobile Web pages.

Forms

An HTML form is a section of a Web page containing content, markup tags, controls (check boxes, radio buttons, menus, and so on), and labels for those controls. In ASP.NET mobile Web pages, forms extend the Microsoft ASP.NET technology across diverse devices.

In an ASP.NET mobile Web page, a form provides the interface between the browser capabilities of a page object and the code that renders that page. The form is a container for controls that encapsulate page logic into reusable components. The form also enables separation of code and content on a page.

Form Activation

Every ASP.NET mobile Web page has one form that is currently active. A form is activated in the following ways:

- Accessing a page for the first time activates the first form on the page, which raises the **Activate** event.
- Setting the **ActiveForm** property of the mobile Web page activates a different form.
- Using a **Link** control that links to another form activates that form when the link is clicked.

The latter two actions first raise the **Deactivate** event of the previously active form, and then raise the **Activate** event of the current form.

There is no reliable way to gauge when a user leaves the current page; thus, the last visited form is never deactivated, and its **Deactivate** event is never raised.

Organizing Content in Forms

You can place any number of controls in an individual form. However, for usability, it is advisable to minimize the number of controls you add to a form.

ASP.NET organizes these controls into multiple units, such as a screen, as appropriate for the target device. The **Form** control then represents an individually addressable set of controls that you can navigate to from within the page. For example, if you have two forms on a page, and one form contains employee personal information and the other contains the employee's employment history, you can reference the personal information by accessing that form.

 Note

You cannot navigate to arbitrary forms in other pages, or through an external URL. In other words, no external URL can take you to a given form within a page. When you browse to a page, the first form automatically becomes active. To change to another form, you must set the **ActiveForm** property of that page.

To determine whether you want to create a new form or add more controls to an existing form, determine the functionality you require. Create a new form if you need a separately addressable set of controls. This is especially useful when the user is moving to a different part of the application. Otherwise, you can add controls to an existing form. Because individual **Form** controls are considered separate units of interaction, ASP.NET never combines multiple forms into a single display, even if there is screen area to do so.

Creating Pages versus Forms

When you create an instance of a page, instances of all forms on that page are created, regardless of the current active form. The overhead for the page therefore depends on the number of forms on a page.

A page provides view-state management over multiple requests. Because instances of all forms on a page are created, any control on any form is addressable from the page. In contrast, state management between pages is much more limited, and you must write custom code to provide additional features.

Only the first form of a page is addressable from an external page. In contrast, each page has its own URL. Thus, the more closely related two forms in an application are, the more sensible it is to place them on the same page. In addition, it is best to place less-frequently used forms in separate pages.

Pages

ASP.NET mobile Web pages are objects that inherit from the **MobilePage** class, either directly or indirectly. The inheritance chain of a mobile Web page depends on how the page is written. If a mobile Web page is self-contained, it inherits directly from the **MobilePage** class. A mobile Web page can also inherit from a custom class that in turn inherits from the **MobilePage** class.

Forms on Mobile Web Pages

Unlike ordinary ASP.NET Web pages, mobile Web pages rely on multiple forms to organize the page's content. A mobile page typically presents information to the user with a succession of forms, and if the data in one form is larger than the device can display, the form can paginate the information onto several screens.

When building a Web application, in general you use forms within a single mobile Web page instead of creating new, separate mobile Web pages. Create a new mobile Web page only when you want to:

- Present a different URL to the user.
- Increase performance.

@ Page Directive for Mobile Web Pages

For backward compatibility, every ASP.NET mobile Web page must contain the following **@ Page** directive.

C#

```
<%@ Page Inherits = "System.Web.UI.MobileControls.MobilePage" Language="C#" %>
```

Visual Basic

```
<%@ Page Inherits = "System.Web.UI.MobileControls.MobilePage" Language="VB" %>
```

Note

When you create a mobile Web page in Visual Studio, the **@ Page** directive is added for you automatically.

The directive instructs the Web page compiler to use the **MobilePage** class as the base class for the page. The **Inherits** attribute of the **@ Page** directive is required. If the page inherits directly from the **MobilePage** class, the **Inherits** attribute must be set to the **System.Web.UI.MobileControls.MobilePage** class. If the page inherits from another class, you specify the name of that class instead. The **Language** attribute is optional, and can be set to the language used on the page.

Note

If you are writing an application that targets the .NET Framework version 1.0, the page must include an **@ Register** directive with a **TagPrefix** attribute. The **@ Register** directive maps the namespace for ASP.NET mobile controls to the **mobile** prefix. This enables you to declare mobile controls on the page using the **mobile** prefix in a tag, such as in the tag **<mobile:Label>**. Although you can use any prefix, the **mobile** prefix is strongly recommended for forward compatibility and consistency within ASP.NET. The following example shows an **@ Register** directive for a mobile page:

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

Pagination

ASP.NET mobile controls provide pagination, which automatically separates the content in your form into smaller groups of rendered content. When you use pagination, these groups of content are automatically formatted to fit the target device. The form also renders user interface (UI) elements that you can use to browse to other pages.

By default, pagination is not activated for a form. To activate pagination, set the **Paginate** property of the **Form** control to **true**. If **Paginate** is **false**, setting the **Paginate** property on a control within the form has no effect. The **Form** control also provides other properties, such as **PageCount**, **CurrentPage**, and **PagerStyle**, which allow you to control pagination behavior.

You can also specify pagination for a specific control on a form using the form's **ControlToPaginate** property.

Recommendations for Using Pagination

Pagination for small interactive forms in an ASP.NET mobile Web page, such as input forms, is often redundant. However, for forms that display large amounts of text or data, pagination can be effective in displaying the information on multiple screens.

When the page contains large quantities of data that changes over time, such as in the case of e-commerce sites where data is being updated constantly, you might consider using custom pagination. For more information about using custom pagination, see the "Custom Pagination" section later in this topic.

Some devices experience memory errors when they try to display more information than their memory can handle. Not only can pagination be effective in displaying forms with large amounts of text or data, but pagination also prevents users from encountering this category of memory errors on their devices.

Device Limitations

Some HTML devices do not support JavaScript (the **JavaScript** property is **false**). If you have a form with an **Action** property set to a nonempty string, the form does not paginate on HTML devices that do not support JavaScript.

Internal Pagination

Mobile controls that are capable of automatic pagination without child controls use internal pagination. For example, a **List** control can paginate its own items, allowing a form to break the single list into multiple pages. Controls that do not support internal pagination must either have child controls or appear atomically on a single screen.

Controls that support internal pagination use the **PagedControl** base class to derive pagination properties, methods, and events for internal and custom pagination. Properties such as the **FirstVisibleItemIndex** property provide access to the individual items on a page. Other properties provide the weight of an item and the count of visible items.

The **List**, **ObjectList**, and **TextView** controls support internal pagination.

Custom Pagination

Controls that support internal pagination also support custom pagination. Normally, controls require the data for all pages to be provided at once, and then discard all but the current page. For custom pagination, controls raise an event to load only the items for the current page. You can specify the total number of items in the **ItemCount** property. If you change the **ItemCount** property from its default value of zero, the control will use custom pagination. In that case, the control raises the **LoadItems** event, which can call an application-specified event handler to provide the items for the current page. The event handler retrieves the appropriate data and binds the data to the control.

Panels

A **Panel** control is an ASP.NET mobile control that can be included within a **Form** control as a grouping mechanism to organize other controls. Use panels to perform the following:

- To define style and flow information for a group of controls and visually group a set of controls together. Panels can be used for style inheritance, allowing child controls within a panel to inherit styles from the panel. For more information, see **Styles**.
- To show, hide, enable, or disable a set of controls.
- To create a container for dynamic controls.

- To provide ASP.NET with information about how to group controls on a screen. When paginating a form, ASP.NET attempts to keep all controls for a single panel on the same screen.

Unlike forms, a panel cannot act as the target of a jump for an application. A panel is not a separate unit of interaction with an application; thus, ASP.NET can combine panels into a single display, to any level of panel depth, as allowed by the target device.

Note

Because the **OnInit** method constructs the templated UI for the control, any control inheriting from **Panel** that adds controls to the **Controls** collection during **OnInit** must call **base.OnInit** after the controls are added, not before.

Styles

Every ASP.NET mobile control provides style properties that you can use to customize how the control is rendered. Styles can be grouped for convenience so that you can apply styles consistency across different elements of a page. Use a **StyleSheet** control or a **Style** element to access properties specific to the control and device capabilities.

Note

Style property settings are not guaranteed to be honored. If a target device does not support a particular style, ASP.NET ignores the style or substitutes a different one.

Style Inheritance

Unless you explicitly specify style properties in your control (either directly, or indirectly using a style reference), a control inherits the style properties of its container. Most style properties default to null or an enumerated value of **NotSet**. This makes it easy to distinguish style properties that have been explicitly set from those that have not been set.

Explicitly Declaring Styles

There are two ways to explicitly declare a style for a control. One way is to set a style property. For example, suppose you create a form and add a **Label** control to the form. (The label is then a child control of the form.) Then, when you set that label's **Font-Name** property to "Arial", the label uses the Arial font.

The other way to explicitly set a style for a child control is to set the **StyleReference** property of the control.

Setting Styles Using a DeviceSpecific Control

You can also set style properties through a DeviceSpecific control. The following example shows a label that is displayed using italic for most devices and bold when it is displayed on a desktop device.

C#

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" %>
<script language="C#" runat="server">
// A custom method to determine if the page is displayed on a mobile device.
public bool IsDesktop(System.Web.Mobile.MobileCapabilities capabilities, String argument)
{
    return !capabilities.IsMobileDevice;
}
</script>
```

```
<Mobile:StyleSheet runat="server">
    <Style Name="ListStyle" Font-Italic="true">
        <DeviceSpecific>
            <Choice Filter="IsDesktop" Font-Italic="false" Font-Bold="true" />
        </DeviceSpecific>
    </Style>
</Mobile:StyleSheet>

<Mobile:Form runat="server">
    <Mobile:Label id="list1" runat="server" StyleReference="ListStyle"> Here is some text </Mobile:Label>
</Mobile:Form>
```

Lookup Algorithm for a Referenced Style

When a style is referenced through the **StyleReference** property, ASP.NET follows a particular sequence of checks to determine what style to apply. For a child control (a control inside a parent container control), the rules for determining the characteristics of the child are described in the following list, using **Font-Size** as an example:

1. If the **Font-Size** attribute has been explicitly set for a child control, the control uses that setting.
2. Otherwise, if the child control's **StyleReference** property is set (for example, myChild.StyleReference = someStyle), the control uses the value of the **Font-Size** attribute from the referenced Style element (for example, someStyle). The child control accesses the value by doing the following:
 - a. The child looks up the referenced style in the style sheet of the current **MobilePage** instance.
 - b. If the child does not find it on the page's style sheet, it references the system default style sheet.
 - c. If the child does not find it in either style sheet, a run-time error is generated.
3. If the style is not available directly or indirectly, the **StyleReference** property has not been set and the child control gets the value of its **Font-Size** attribute by recursively applying this procedure to the parent control.
4. If the recursion reaches the top of the control hierarchy without finding an explicit value for the **Font-Size** attribute, the control uses the default font size.

This algorithm allows for separate styles that you can reference from multiple controls. It supports inheritance from the containing controls, and it follows standard coding and logic expectations.

Exceptions to the Lookup Algorithm

There are two exceptions to the lookup algorithm:

- A background color does not receive its value from the parent object. (This is consistent with the behavior cascading style sheets.)
- **DeviceSpecific** controls do not inherit values from the parent control. A **DeviceSpecific** control is usually explicitly authored for a specific control or type of control.

Style Sheets

ASP.NET mobile controls provide a default style sheet that sets a limited number of styles for you. For more information, see **StyleSheet**. You can easily override the values in these default styles to apply your own. Any number of **<Style>** element declarations can reside in a single style sheet. Each **<Style>** element is identified by a unique **Name** property. You can set the **StyleReference** property of another control to the **Name** property, thus referencing its style. You can also use this technique to reference a style from another style.

External Style Sheets

It is possible to define an external style sheet that you can use for multiple controls. This is advantageous if you want to use the same styles across multiple pages. To create an external style sheet, create a user control in an .ascx file, and place a single **StyleSheet** control with a set of styles in it. Then, to refer to this file, place a **StyleSheet** control on a mobile page and set its **ReferencePath** property to the relative URL of the user control.

External Style Sheet Implementation

Implementing an external style sheet requires three steps:

1. Write a Microsoft ASP.NET mobile user control in an .ascx file.
2. Place a single style sheet in the .ascx file, adding the **<Style>** elements you need.
3. Declare a style sheet and set its **ReferencePath** property to the .ascx file name of the user control for each mobile page where you want to use the external style sheet.

At run time, all the styles that you declared in the external style sheet are made available to the ASP.NET page framework for the style sheet on the mobile page.

Characteristics of Style Objects and the Style Class

A **Style** object is not a true control, and it does not inherit from the base **MobileControl** class. On a page, it can be declared only within a **StyleSheet** control through using **<Style>** element.

The base **Style** class contains style characteristics common to all mobile controls. Classes that inherit from the **Style** class contain additional style characteristics specific to their associated control.

Every **MobileControl** internally contains a **Style** object. However, this **Style** object is not exposed through public members. Instead, for each style property, the **MobileControl** has a publicly accessible property that internally references the privately contained style. Therefore, a **MobileControl** directly exposes style properties such as **Font**, **ForeColor**, and **Wrapping**.

Organization of Styles

You can organize styles into a **StyleSheet** control. Within a style sheet, you can declare any number of style objects. Styles are declared the same way as any control, with the exception that a **runat=server** attribute is not required.

A style can inherit its properties from another style in the style sheet by having its **StyleReference** property set to the name of the parent style. The scope for this is the mobile page. That is, only styles on the style sheet on the same mobile page can be referenced. To have a control acquire its styles from a style object in the style sheet, set the **StyleReference** property of its style object to the name of the style, either by declaring the **StyleReference** attribute in an ASP.NET mobile Web page, or by programmatically setting the **StyleReference** property.

Literal Text inside Form Markup

For ASP.NET mobile controls, the term **literal text** refers to the text that is placed directly in a **Form**, **Panel**, or **TextView** mobile control. The following example shows text placed directly in a **Form** control:

```
<mobile:Form>This is literal text.</mobile:Form>
```

This allows you to quickly display a chunk of HTML in a form in an ASP.NET mobile Web page.

You can use a limited number of formatting elements inside form markup, as listed in the following table:

Element	Description
<code><a></code>	Converts inner text to a hyperlink. The hyperlink text cannot contain other formatting tags.
<code></code>	Converts inner text to a bold style.
<code>
</code>	Breaks to a new line.
<code><i></code>	Converts inner text to an italic style.
<code><p></code>	Begins a new paragraph or, when used with a closing tag, places inner text in a separate paragraph.

Other tags are ignored at run time. Note that at design time, unsupported tags in literal text can cause undesirable results when the page is edited in Visual Studio.

The tags used in literal text do not necessarily correspond to tags in the output. For example, `<p>` tags might be translated into `<br>` tags by a control adapter. In addition, because controls themselves can cause line breaks, line breaks that exist as the only literal text between two controls are ignored. To force a break between two controls, you can insert a non-breaking space (` `) between the controls, in addition to the appropriate break.

When nesting tags, the hyperlink tag (`<a>`) does not recognize nested tags. For example, nesting the `<b>` or `<i>` tag as literal text inside the `<a>` tag does not render a link as bold or italic. The control completely ignores all tags inside of the `<a>` tag.

During compilation, literal text is translated into **LiteralText** and **Link** controls. Because the text is static, these controls are not intended to be addressable, although they still appear in the page's control tree. (The control tree is the tree of controls on a page — the page itself, its child controls, their children, and so on.) Thus, if you are programmatically enumerating the child controls of a form, you might find a **LiteralText** control, even though you did not explicitly add one to your page.

Note

Place a `<br>` tag in footer templates to ensure that the footer template contents do not appear inline with the page.

Device-Specific Rendering

Although ASP.NET mobile Web pages can render to a variety of devices automatically, they also provide ways for you to specify content specific to a device or a class of devices. This means that mobile Web pages allow you to customize a page to take advantage of particular features of a device. For example, a common requirement is to render items differently on different devices. ASP.NET handles much of the implicit formatting appropriate for rendering on a variety of device types. However, if needed, you can have ASP.NET render a label using one string on one device and a different string on another device.

To manage such scenarios, you include code in the application to set the properties of the controls depending on the results of device capabilities queries. To customize a mobile Web page for specific types

of devices, you define a set of device filters for the page and then specify the filter for each type of device using a **<DeviceSpecific><Choice>** construct.

Note

The default Visual Studio mobile configuration file contains a variety of predefined device filters. ASP.NET 2.0 does not automatically generate mobile device filters by default. However, if you migrate your Web application from a previous version of ASP.NET to ASP.NET version 2.0, your device filters will be maintained in the Web.config file.

Using the Filter Property

Use the **Filter** property to evaluate device filters against device capabilities, or to set specific filters.

The filter name must be the name of a method on the page or associated .ascx file, or the name of a valid device filter defined in the **<deviceFilters>** element of the Web.config file. If a method name is specified with the **Filter** property, that method must match the following prototype:

C#

```
public bool methodName( System.Web.Mobile.MobileCapabilities capabilities, String optionalArgument);
```

For example, if the **Filter** property is set to myChoiceMethod, a method with the following signature must exist.

C#

```
public bool myChoiceMethod( System.Web.Mobile.MobileCapabilities capabilities, String optionalArgument );
```

When ASP.NET evaluates the **Choice** element, it checks whether a method of the appropriate signature exists in the page or user control. If it does not exist, ASP.NET checks the **<deviceFilters>** element of the Web.config file.

Using Device Filters to Extend the MobileCapabilities Class

By adding your own device filters to the Web.config file, you can extend the **MobileCapabilities** class.

Configuring the device filters provides an evaluation mechanism for two types of filters: a comparison-based filter and an evaluator-delegate-based filter.

Comparison-Based Filters

The comparison-based filter makes basic comparisons, generally based on a Boolean argument. For this type of filter, you provide the name of a capability and the value that you want the filter to compare. At run time, the evaluator succeeds if the capability value and the value that you provide are equal. The compared Boolean properties are case-insensitive, so **true** and **True** are equivalent. Other compared properties are case-sensitive.

Evaluator-Delegate-Based Filters

For more complex evaluation, you can specify an evaluator-delegate-based filter by providing the class and method name of a method. At run time, the method that you provide is called to test the evaluator. You must write and compile your own method to test the evaluator. Alternatively, you can define a method in your page or user control, and then reference it directly from the **filter** attribute, as described previously.

Web.config File Syntax

When specifying device filters in the Web.config file, you add them to the **<system.web>** section. To view the syntax, see **<deviceFilters>**. The syntax applies to both types of filters. In the following example, the first filter shows the comparison-based filter, and the second filter shows the evaluator-delegate-based filter:

```
<system.web>
  <deviceFilters>
    <filter name="capability" compare="capabilityName" argument="argument" />
    <filter name="capability" type="className" method="methodName" />
  </deviceFilters>
</system.web>
```

The **MobileCapabilities** object evaluates these filters at run time. Each device filter specifies an evaluation condition based on capabilities of the device. The sets of devices targeted by device filters are usually not discrete; for example, many devices can match both the **IsColor** and **IsPDA** filter attributes.

Note

Device filter names are case-sensitive.

Within the **<DeviceSpecific><Choice>** construct, you specify the values for these filters. For example, the following code accesses the **IsColor** filter attribute defined in the Web.config file.

```
<DeviceSpecific>
  <Choice Filter="IsColor" ImageUrl="colorImg.gif" />
</DeviceSpecific>
```

Using the Device-Specific/Choice Construct

The **<DeviceSpecific><Choice>** construct is the core construct for inserting device-specific markup in a page. To add device-specific markup for a control, you add the **<DeviceSpecific>** element as a child of the control. A control may contain only one **<DeviceSpecific>** element.

The **<DeviceSpecific>** element and the **<Choice>** element enable you to specify a set of values from which ASP.NET chooses, based on characteristics of the requesting device. The values of the choices are strings.

The **<DeviceSpecific>** element is an outer container for holding a number of choices, as shown in the following example:

```
<mobile:Image runat=server ImageURL="bw.gif">
  <DeviceSpecific>
    <Choice Filter="isColor" ImageURL="colorImg.gif" AlternateText="This device cant display the image." />
    <Choice Filter="isWML11" ImageURL="myImage.wbmp" />
    <Choice ImageURL="monoImage.gif" />
  </DeviceSpecific>
</mobile:Image>
```

Filters such as **isColor** must have corresponding entries in the Web.config file or must exist as methods on the page or user control, as described previously.

Using the Choice Element

Choices are placed inside a **<DeviceSpecific>** element, and represent device characteristic/value pairs, where the device characteristic is drawn from a number of sources. Choices are evaluated in the order that they appear in the **DeviceSpecific/Choice** construct.

Although a control can contain only one **<DeviceSpecific>** element, you can add any number of **<Choice>** elements within a **<DeviceSpecific>** element. Each **<Choice>** element can contain the following:

- A filter name, which specifies the device filter to evaluate. If the filter name is omitted, the choice is selected by default.
- Additional properties that override properties of the parent control.
- Template definitions for the control.

ASP.NET selects the **<Choice>** element to use by going through each choice in order and evaluating the filter specified by the filter name. If the filter matches the current target device (by evaluating to **true**), that choice is selected. The control then applies any property overrides specified in the choice, and can use any templates defined in rendering.

Template Sets and Templated Controls

ASP.NET provides templated controls that expose a number of template properties that define the control's content and layout. These templates are inserted in the appropriate places during the rendering of the control. For example, there are templates for the **List** control, such as header and footer templates. This feature allows you to significantly customize the appearance of controls at run time, based on the device.

ASP.NET mobile controls extend this model and introduce the notion of template sets. A template set is a collection of templates. However, a single templated control might refer to multiple sets of templates, with each template set having different device-specific criteria.

Rendering Criteria for Template Sets

Each template set of a templated control is contained in a **<Choice>** element inside a shared **<DeviceSpecific>** element. At run time, the choices in the **<DeviceSpecific>** element are evaluated in order. The first matching **<Choice>** element is used for device-specific content. If the selected choice contains templates, the control uses them. If no templates are found, or if none of the specified choices is appropriate, the control renders its default markup.

Device-Independent Templates

Any templated control can have a device-independent template. To specify device-independent templates, use a **<Choice>** element without an explicit **Filter** attribute. A device-independent choice, where specified, must always be the last choice declared, so that it is chosen if no other choices apply to the target device.

Control-Specific Rendering

The rendering behavior of controls in templated mode is specific to the control. Some controls, such as **List** and **ObjectList**, might render all their contents from the provided templates. Other controls might add the contents of specific templates to their default content. For example, if a header or footer template is selected

for the **Form** control, the markup contained in the template is added to the form contents as a header or footer, respectively.

Setting Properties for a Templated Control

To programmatically set properties of a control in a template, you must obtain a reference to the naming container and use the **FindControl** method to find the control. You can then set its properties. The following example illustrates this technique.

Note

Because style sheet information is loaded prior to when you can programmatically change the **StyleReference** property, changing it in code has no effect.

```
C#
void Page_Load(Object sender, EventArgs e)
{
    // Iterate through the controls in the form.
    foreach(Control c in Form1.Controls)
    {
        if(c.GetType() == typeof(TemplateContainer)
        {
            // Get the link control.
            Link ctrl = (Link)c.FindControl("myLink");
            if (ctrl != null)
            {
                // Set the text and url properties.
                ctrl.Text = "Home Page";
                ctrl.NavigateURL = "http://www.microsoft.com";
            }
        }
    }
}
```

Template Sets and Styles

Styles in a **StyleSheet** control might also contain template sets. Thus, you can have multiple-templated controls that reference the same style, use the same template sets, and provide the same benefits of encapsulation that a style provides.

Style Templates vs. Style Properties

You can inherit a style from a parent, you can explicitly set the **StyleReference** property, or you can inherit the style through aggregation. With templates, however, there is no cascading effect. You cannot retrieve a template from a parent style template unless you use a template within a style sheet.

Dynamically Added Templates

In some advanced scenarios, it is useful to dynamically instantiate and add templates. The following code example adds templates in an Init event handler.

```
C#
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>

<script runat="server">
private void Form_Init(object sender, System.EventArgs e)
{
    DeviceSpecific devSpecific = new DeviceSpecific();
```

```
// Create the choice tag.
DeviceSpecificChoice devChoice = new DeviceSpecificChoice();
devChoice.Filter = "isHTML32";

// Create the template.
ITemplate myTemplate = new CustomTemplate("HeaderTemplate");

// Create the templateContainer.
TemplateContainer container = new TemplateContainer();
myTemplate.InstantiateIn(container);

// Create the tree.
devChoice.Templates.Add("HeaderTemplate", myTemplate);
((IParserAccessor)devSpecific).AddParsedSubObject(devChoice);
((IParserAccessor)form1).AddParsedSubObject(devSpecific);
}

public class CustomTemplate : ITemplate
{
    String strWhichTemplate;
    public CustomTemplate(String strTemplate)
    {
        strWhichTemplate = strTemplate;
    }
    public void InstantiateIn(Control container)
    {
        if (strWhichTemplate == "HeaderTemplate")
        {
            System.Web.UI.MobileControls.Label lb = new System.Web.UI.MobileControls.Label();
            lb.Text = "Header Template";
            System.Web.UI.MobileControls.Command cmd = new System.Web.UI.MobileControls.Command();
            cmd.Text = "heLLo";
            container.Controls.Add(lb);
            container.Controls.Add(cmd);
        }
        else if (strWhichTemplate == "FooterTemplate")
        {
            System.Web.UI.MobileControls.Label lb = new System.Web.UI.MobileControls.Label();
            lb.Text = "FooterTemplate";
            container.Controls.Add(lb);
        }
    }
}
</script>

<html><body>
    <mobile:form id="form1" runat="server" OnInit="Form_Init"></mobile:form>
</body></html>
```

Special Considerations for Using Device-Specific Templates

When mixing device-specific markup languages with mobile controls, you must ensure consistency, based on what the mobile controls are rendering. Intelligent detection and adaptation for mixed device-specific and device-independent markup is not supported.

For example, note the alignment setting on the first **Panel** control and the **Label** control in the following code sample:

```
<mobile:Panel runat=server alignment="right">
<DeviceSpecific>
    <Choice Filter="isWML11">
        <ContentTemplate>
            <table columns="2" align="LR"> <tr><td>
                </ContentTemplate>
            </td><td>
                </ContentTemplate>
            </td></tr></table>
        </Choice>
    </DeviceSpecific>
</mobile:panel>
```

```
<Mobile:Label id="label1" runat=server Text="HELLO, HOW ARE YOU?" alignment="left"></Mobile:Label>

<mobile:Panel runat=server>
  <DeviceSpecific>
    <Choice Filter="isWML11">
      <ContentTemplate>
        </td><td>
      </ContentTemplate>
    </Choice>
  </DeviceSpecific>
</mobile:panel>
```

WML **<p>** elements are used to render non-default alignment. In the preceding example, the second **Label** control is within a WML **<td>** element, and the **<p>** element generated for the second **Label** control is incorrectly rendered by the browser because it is located within the **<td>** element.

In this case, the **Label** control does not inherit the alignment from the first **Panel** control, so it generates a **<p>** tag for its alignment. However, a **<p>** tag cannot be added in this situation. This is not a common situation, but you can work around the problem by marking the **Label** control as visible if the target device is not WML-based and by making the text of the **Label** control specified within the template.

Arbitrary Device Specific Markup

In some cases, you might want to insert arbitrary markup for a specific device or type of device. To enable this, ASP.NET mobile Web pages provide a content template for the **Panel** control. If a selected choice contains a content template, the control renders using the template instead of its usual contents.

Linking Between ASP.NET Mobile Web Pages

If you have a link in the format #form1 in a control contained in a user control, the **ResolveFormReference** method looks in the user control for a form whose **ID** property is set to form1. If the form is not found, the method walks up the chain of nested user controls, and then searches for the form on the page. To link a form that is contained in a user control, use the following syntax to identify the form.

```
#mc1:form4
```

mc1 is the user control identifier. The colon (:) separates the reference to the form.

Note

There is no support for element anchors (URLs of the form page.aspx#element, where page is not the current page).

Example

The following code example demonstrates navigation between forms. The example contains a mobile Web page and a mobile user control.

Formtest.aspx

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Register TagPrefix="uc1" TagName="MobileWebUserControl1" Src="formtest.ascx" %>

<script runat="server">
void Form_Activate(Object sender, EventArgs e)
```

```
{
    ((Form)sender).DataBind();
}
</script>

<html><body>
<mobile:form id="Form1" runat="server" OnActivate="Form_Activate">
    <mobile:Label ID="Label1" runat="server" Text='<%# "Current: " + ActiveForm.UniqueID %>' />
    <mobile:Link ID="Link1" NavigateUrl="#form2" runat="server">Go to Form 2</mobile:Link>
    <mobile:Link ID="Link2" NavigateUrl="#form3" runat="server">Go to Form 3</mobile:Link>
    <mobile:Link ID="Link3" NavigateUrl="#mc1:form4" runat="server">Go to Form 4</mobile:Link>
</mobile:form>

<mobile:Form ID="Form2" Runat="server" OnActivate="Form_Activate">
    <mobile:Label ID="Label2" runat="server" Text='<%# "Current: " + ActiveForm.UniqueID %>' />
    <mobile:Link ID="Link4" NavigateUrl="#form1" runat="server">Go to Form 1</mobile:Link>
    <mobile:Link ID="Link5" NavigateUrl="#form3" runat="server">Go to Form 3</mobile:Link>
    <mobile:Link ID="Link6" NavigateUrl="#mc1:form4" runat="server">Go to Form 4</mobile:Link>
</mobile:Form>

<mobile:Form ID="Form3" Runat="server" OnActivate="Form_Activate">
    <mobile:Label ID="Label3" Runat="server" Text='<%# "Current: " + ActiveForm.UniqueID %>'></mobile:Label>
    <mobile:Link ID="Link7" NavigateUrl="#form1" Runat="server" >Go to Form 1</mobile:Link>
    <mobile:Link ID="Link8" NavigateUrl="#form2" Runat="server" >Go to Form 2</mobile:Link>
    <mobile:Link ID="Link9" NavigateUrl="#mc1:form4" Runat="server" >Go to Form 4</mobile:Link>
</mobile:Form>

<uc1:MobileWebUserControl1 id="mc1" runat="server" />
</body></html>
```

Formtest.ascx

C#

```
<%@ Control Language="C#" ClassName="FormTest" Inherits="System.Web.UI.MobileControls.MobileUserControl" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
void Form_Activate(Object sender, EventArgs e)
{
    ((Form)sender).DataBind();
}
</script>

<mobile:Form ID="Form4" Runat="server" OnActivate="Form_Activate">
<mobile:Label ID="Label1" runat="server" Text='<%# "Current: " + ((MobilePage)Page).ActiveForm.UniqueID %>' />
    <mobile:Link ID="Link1" NavigateUrl="#form1" runat="server">Go to Form 1</mobile:Link>
    <mobile:Link ID="Link2" NavigateUrl="#form2" runat="server">Go to Form 2</mobile:Link>
    <mobile:Link ID="Link3" NavigateUrl="#form3" runat="server">Go to Form 3</mobile:Link>
    <mobile:Link ID="Link4" NavigateUrl="#form4a" runat="server">Go to Form 4a</mobile:Link>
</mobile:Form>

<mobile:Form ID="Form4a" Runat="server" OnActivate="Form_Activate">
<mobile:Label ID="Label" runat="server" Text='<%# "Current: " + ((MobilePage)Page).ActiveForm.UniqueID %>' />
    <mobile:Link ID="Link5" NavigateUrl="#form1" runat="server">Go to Form 1</mobile:Link>
    <mobile:Link ID="Link6" NavigateUrl="#form2" runat="server">Go to Form 2</mobile:Link>
    <mobile:Link ID="Link7" NavigateUrl="#form3" runat="server">Go to Form 3</mobile:Link>
    <mobile:Link ID="Link8" NavigateUrl="#form4" runat="server">Go to Form 4</mobile:Link>
</mobile:Form>
```

Working With Deck Size Limits

When a WML-based cell phone requests a Web page, the WML deck returned by the server is compiled by the WAP gateway into a compact representation of the Web page. The browser then receives this compiled version of the deck.

Most WML-based browsers have a limitation on the size of a compiled deck that they can receive. This is known as the deck size limit, and varies according to the individual device. Because the limit applies to content compiled at the gateway, it is difficult to determine whether a WML deck is too large for a device. This is particularly true for dynamic, data-bound pages, because the change in size due to gateway compilation depends on the exact content of the WML deck.

ASP.NET does not limit or truncate pages to make decks fit within the deck size limit of individual devices. As a general guideline, you must design pages so that the rendered WML output is approximately 1,200 characters. By doing so, you can usually avoid decks that are too large because of variations in dynamic content.

To determine rendering size

1. If controls on the page are data-bound, bind the data to a typical set of data.
2. Test the page on an emulator.
3. Determine the size of the rendered output, either by using information provided by the emulator, or by using ASP.NET application tracing.

To reduce the rendered size of a page

To reduce the size of the markup rendered by the page, follow these guidelines:

- Use property overrides to specify abbreviated text for labels, lists, and other controls.
- Enable form pagination by setting the **Paginate** property to **true**.
- Break a single form into a series of forms.
- Use a **Panel** control with a content template to provide separate content for other devices.
- If you are using a **List** or a similar control, reduce the number of items on each page by setting the **ItemsPerPage** property. You can use a property override to make the value specific to one or more WML-based devices.

If the dynamic content for a page can vary significantly, repeat the steps for testing page sizes with different amounts of data and compare the sizes to determine how to reduce the page size.

The **MaximumRenderedPageSize** property of the **MobileCapabilities** class provides the maximum deck size of the target device. By using this value with property overrides, you can further customize the content of an application to suit specific devices. For example, if the maximum deck size of a device is more than 2,000 bytes, you can adjust your page content based on a guideline of 1,500 characters instead of 1,200.

Designing an ASP.NET Web Application

When designing your ASP.NET Web application, it is beneficial to separate the definition of the user interface (UI) from the business logic and the data store. Mobile Web pages, like ASP.NET Web pages, enable you to easily separate rendering from logic. For example, you can place the UI definition in an .aspx file, with the associated event handlers and other UI code in either the same file or in a code-behind class file. You can then place business logic code in separate classes, written in the .NET Framework language of your choice.

A key advantage to separating rendering and logic is that you can reuse code for ASP.NET Web pages and ASP.NET mobile Web pages within an ASP.NET Web application. If you have written an ASP.NET Web application that supports desktop Web browsers, you can use the business logic code also with pages that support mobile devices. You must write one set of Web pages for the desktop, and one set of mobile Web pages for mobile devices, but they can share the same business logic code. To help facilitate this, mobile Web pages are capable of containing multiple forms, so that you can factor your application into similar units for both desktop and mobile UI.

By reusing business logic, you can provide a more integrated experience for your mobile users in these ways:

- Reducing the cost of development by using common code.
- Providing fast time-to-market development.
- Leveraging existing ASP.NET skills.

For example, if you allow a user to customize your Web application, you can share the user profile across your Web application. This allows you to enable scenarios in which the user can configure settings by using a desktop browser, and use those same settings using a mobile device.

Redirecting to an ASP.NET Mobile Web Page

If you create an application that has one user interface optimized for a desktop browser and a second user interface optimized for mobile applications, you need a mechanism to redirect mobile device customers to the mobile Web pages.

Because some devices do not support cookies, consider whether you want to rely on cookies for your application.

If your application does not use cookies, you can use the following code in a Microsoft ASP.NET site to redirect to an ASP.NET mobile Web application.

```
C#
<script runat="server" language="c#">
public void Page_Load(Object sender, EventArgs e)
{
    if (Request.Browser["IsMobileDevice"] == "true" )
    {
        Response.Redirect("MobileDefault.aspx");
    }
    else
    {
```

```
        Response.Redirect("DesktopDefault.aspx");  
    }  
}
```

Because some devices do not accept relative URLs, you must also set the **useFullyQualifiedRedirectUrl** attribute of the **<httpRuntime>** element to **true** in the Web.config file. This sends a fully qualified URL to the client with the session ID appended to the end of the URL. Specifying a relative URL and then converting that URL to a fully qualified URL is necessary to preserve session state. The following example shows the configuration setting.

```
<configuration>  
  <system.web> <httpRuntime useFullyQualifiedRedirectUrl = "true" /> </system.web>  
</configuration>
```

Managing Adaptive Error Reporting in ASP.NET Mobile Web Pages

During the processing of an ASP.NET page, errors are thrown as exceptions and are handled by the ASP.NET error-handling mechanism.

This section provides an overview of error reporting in ASP.NET and how this relates to ASP.NET applications that contain mobile Web pages.

Overview of ASP.NET Error Reporting

Several types of errors can occur during a request, including the following:

- Errors in the HTTP intrinsics, such as a request for a missing file.
- Errors parsing a page, user control, or similar file while compiling it into an assembly.
- Errors running a Web page within ASP.NET during the lifetime of the page.
- Errors reading a configuration file.
- System errors, such as an out-of-memory condition.

ASP.NET offers several features for customized error reporting.

Error-Reporting Mode

You can configure an application to show detailed errors with information relevant to the developer, basic errors for ordinary users, or custom errors. You can adjust the settings to show detailed errors only when the client is the local computer.

Application-Level Errors

When an error occurs during a request, ASP.NET raises the **Application_Error** event. A method in the Global.asax file can handle this event and override error-handling behavior.

Page-Level Errors

Errors that occur during the lifetime of a page raise the **Page_Error** event. A handler for this event might be able to perform notification tasks or attempt corrective action, including suppressing the error, depending on the error.

Custom Errors

In the Web.config file, you can specify a set of custom pages for exceptions that occur within the ASP.NET application. When an error occurs, ASP.NET checks whether the application is configured to show custom errors and whether an appropriate error page exists. If either of these is true, ASP.NET redirects to the error page, passing a parameter that contains the original URL.

Overview of Adaptive Error Reporting in Mobile Web Applications

Error reporting in mobile Web pages works the same way as it does for any other ASP.NET applications. The same customization techniques are available. However, some differences in behavior make error reporting in mobile Web pages more adaptive to working with devices.

Device-specific Formatting

Error messages are automatically formatted to the target device's markup. For WML devices, this is a card deck. For HTML devices, this is an HTML page.

Note

It is recommended that if you write custom error pages, you write them as ASP.NET mobile Web pages, so that your custom error pages are properly formatted for each type of device.

Limited Default Error Message Content

For all mobile devices, the built-in error messages are terse in nature, even when the application is configured to show a detailed message. An error message typically contains the type of exception raised and the method that caused the exception. When the client is desktop browser, however, the standard error message is rendered.

HTTP Response Code

When ASP.NET reports an exception, it sets the HTTP response code to reflect the nature of the error. Browsers can act on the response code or show the error details contained in the response body. However, some mobile devices, particularly WML-based phones, show the response code only if there is an error. When an error occurs for such a device, the request returns the HTTP response code 200, indicating success, but the body of the page contains the error message. On HTML devices, the request returns the actual error code so the browser can respond accordingly.

Adaptive Error Reporting Process

For mobile Web pages, ASP.NET finds and reports errors using the following process:

1. If an application-level error occurs, an HTTP module of type **ErrorHandlerModule** handles the error. (This module is automatically installed.)
2. If an exception occurs at the page level during the page life cycle, ASP.NET calls the **OnError** method of the page. Because the page is a mobile page, an override implementation in the **MobilePage** is called, which in turn calls the **HandleError** method of the assigned page adapter. The adapter method can report the error in detail and return a value indicating that the error was handled. If it does not do so, the exception continues to be processed. ASP.NET automatically uses custom error pages.

3. ASP.NET calls the error handler from step 1. If the target device is an HTML browser capable of rendering the full ASP.NET generated error message, the method ends.
4. Otherwise, ASP.NET collects information about the exception, creates an instance of an internally defined mobile page of type **ErrorFormatterPage**, binds the data to the collected data, and renders the results. This page is responsible for producing a device-specific error message. Because the exception has been handled, the event method does not return an HTTP error status code.

Viewing ASP.NET Mobile Web Pages

You need to test ASP.NET mobile Web pages on a variety of devices and emulators to assure support for the greatest number of devices. This section describes the ways you can view an ASP.NET mobile Web page.

Using Your Desktop Browser

Because ASP.NET mobile Web pages support HTML-based browsers, you can use your desktop browser to view mobile Web pages. You can debug your application with the desktop browser because it displays detailed error information, such as compilation and run-time errors. You can also enable page tracing and view the resulting trace information on the page.

Using an Emulator

You can often obtain an emulator application for a mobile device. Emulators enable you to test your application from your desktop workstation, and do not require an actual device or a wireless connection. Emulators can also include additional development tools, such as the ability to view page source or device state.

Most emulators enable you to view applications installed locally on your workstation. However, others might require additional components, such as ActiveSync or a gateway.

To determine whether a supported device has an emulator, consult the device manufacturer. To install and use an emulator, see the documentation that accompanies the emulator. The manufacturers' Web sites often have information about emulators available for their devices.

Using a Wireless Device

If your cellular phone or other mobile device has wireless Internet access, and if your Web server can be accessed from the Internet, you can use your phone or device to view your mobile Web pages. If the server is on your corporate intranet, your network might need a proxy or gateway. Products such as Microsoft Mobile Information Server 2002 can provide secure wireless access to intranet servers.

If you have a cradle for your device, you may be able to install ActiveSync on your machine and connect your device to your machine through ActiveSync.

Using a Pocket PC

Even if you do not have wireless Internet access, you can view your application using a Pocket PC if it has network connectivity to your Web server. Pocket PCs have several network connectivity options. For more information, see the documentation that accompanies your device.

To view your mobile Web page, first connect your Pocket PC to the network. Then open Microsoft Internet Explorer for the Pocket PC, and enter the URL of the application.

Note

For best viewing with a Pocket PC, make sure that the **Fit to screen** feature is on.

Testing Devices without Cookie Support

All ASP.NET applications can be configured to work with or without the use of cookies. Some mobile devices do not support cookies, and other devices might allow the user to turn off cookies. To support these devices, configure your application to use cookieless sessions, and do not rely on cookies in your own application code.

ASP.NET Mobile Controls

AdRotator Class

Provides a server control to display a randomly selected advertisement on a mobile page.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class AdRotator Inherits MobileControl

C#

public class AdRotator : MobileControl

The **AdRotator** class uses the same configuration file as the Web Forms **AdRotator** class. The **System.Web.UI.MobileControls.AdRotator** mobile control exposes many of the same properties and events as the **System.Web.UI.WebControls.AdRotator** control, plus it adds mobile capability. The displayed advertisement changes whenever the page is refreshed.

The **AdvertisementFile** property specifies the location of an XML file that contains advertisement information. This file must reside within the application domain. For security purposes, XML files should be in the App_Data folder, which is configured to not allow users direct access to XML files. If the **AdvertisementFile** property is empty, the **AdRotator** control generates a single break tag as a placeholder. This is useful when you do not want to display an advertisement.

Example

The following code example consists of two parts: an ASP.NET mobile Web Forms page in an .aspx file and an XML file. The .aspx file uses a file named ads.xml to rotate through various advertisements based on the **KeywordFilter** property. If you provide the images for the example, they will be displayed; otherwise, the **AdRotator** control displays the value of the **AlternateText** property for the advertisements. When the user refreshes the mobile Web Forms page, the page displays the next randomly selected advertisement, based on the **KeywordFilter**.

The example also shows how to use a **<Choice> Element** to override the properties if the user's browser requires WML markup, and how to map the image's src and href attributes to the data in the XML file using the **ImageKey** and **NavigateUriKey** properties, respectively.

Although the example uses a function to determine whether the browser requires WML (isWML11), you can instead use a Web.config file to define a **DeviceSpecific** element that the .NET Framework automatically uses to make the determination for you:

```
<deviceFilters>  
  <filter name="isWML11" compare="PreferredRenderingType" argument="wml11" />  
</deviceFilters>
```

Note:

The following code sample uses the single-file code model and may not work correctly if copied directly into a code-behind file. This code sample must be copied into an empty text file that has an .aspx extension.

The .aspx file:

```
Visual Basic
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" %>
<%@ Import Namespace="System.Web.Mobile" %>

<script runat="server">
Private Sub AdCreated_Event(ByVal sender As Object, ByVal e As AdCreatedEventArgs)
    Label2.Text = "Clicking the AdRotator control takes you to " + e.NavigateUrl
End Sub

'Determine whether the current browser is a WML browser.
Public Function isWml11(ByVal caps As MobileCapabilities, ByVal value As String) As Boolean
    'Determine if the browser is not a Web crawler and requires WML markup
    If Not caps.Crawler AndAlso caps.PreferredRenderingMime = _
        MobileCapabilities.PreferredRenderingTypeWml11 Then
        Return True
    Else
        Return False
    End If
End Function
</script>

C#
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" %>
<%@ Import Namespace="System.Web.Mobile" %>

<script runat="server">
private void AdCreated_Event(Object sender, AdCreatedEventArgs e)
{
    Label2.Text = "Clicking the AdRotator control takes you to " + e.NavigateUrl;
}

// Determine whether the current browser is a WML browser
public bool isWML11(MobileCapabilities caps, string optValue)
{
    // Determine if the browser is not a Web crawler and requires WML markup
    if (!caps.Crawler && caps.PreferredRenderingType == MobileCapabilities.PreferredRenderingTypeWml11)
        return true;
    else
        return false;
}
</script>

<html>
<body>
<mobile:form id="form1" runat="server">

    <!-- The AdRotator control -->
    <mobile:AdRotator id="AdControl" runat="server" ImageKey="MobileImgSrc" NavigateUrlKey="TargetUrl"
        AdvertisementFile="App_Data/ads.xml" Alignment="Left" KeywordFilter="Developer"
        OnAdCreated="AdCreated_Event">

        <DeviceSpecific>
            <Choice Filter="isWML11" NavigateUrlKey="WmlTargetUrl" ImageKey= "WmlImageSrc" />
        </DeviceSpecific>
    </mobile:AdRotator>

    <!-- The instructions label -->
    <mobile:Label id="Label1" runat="server" Text="Refresh the page to change the advertisement" />
    <!-- The URL info label -->
    <mobile:Label id="Label2" runat="server" />

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

The sample ads.xml file (which must be located in the **App_Data** folder):

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
  <Ad>
    <WebImgSrc>imgA1.gif</WebImgSrc>
    <MobileImgSrc>imgA2.gif</MobileImgSrc>
    <WmlImgSrc>imgA3.gif</WmlImgSrc>
    <TargetUrl>http://msdn.microsoft.com/</TargetUrl>
    <WmlTargetUrl>http://OurServer/MS-MSDN.wml</WmlTargetUrl>
    <AlternateText>MSDN</AlternateText>
    <Keyword>Developer</Keyword>
    <Impressions>80</Impressions>
  </Ad>
  <Ad>
    <WebImgSrc>imgB1.gif</WebImgSrc>
    <MobileImgSrc>imgB2.gif</MobileImgSrc>
    <WmlImgSrc>imgB3.gif</WmlImgSrc>
    <TargetUrl>http://www.microsoft.com/</TargetUrl>
    <WmlTargetUrl>http://OurServer/MS-Home.wml</WmlTargetUrl>
    <AlternateText>Microsoft</AlternateText>
    <Keyword>Customer</Keyword>
    <Impressions>90</Impressions>
  </Ad>
  <Ad>
    <WebImgSrc>imgC1.gif</WebImgSrc>
    <MobileImgSrc>imgC2.gif</MobileImgSrc>
    <WmlImgSrc>imgC3.gif</WmlImgSrc>
    <TargetUrl>http://www.microsoft.com/net/</TargetUrl>
    <WmlTargetUrl>http://OurServer/MS-Net.wml</WmlTargetUrl>
    <AlternateText>.NET</AlternateText>
    <Keyword>Developer</Keyword>
    <Impressions>80</Impressions>
  </Ad>
</Advertisements>
```

Calendar Class

Provides control capability to display a calendar.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class Calendar **Inherits** MobileControl **Implements** IPostBackEventHandler

C#

public class Calendar : MobileControl, IPostBackEventHandler

The calendar display is by day, week, or month. Whether an entire month is rendered on a mobile device depends on the capabilities of the device. In general, the **Calendar** control permits the selection of a date.

The mobile **Calendar** control wraps a Web Forms **Calendar** control. Although the mobile **Calendar** control mimics some properties, methods, and events of the underlying control, it does not expose other properties specific to HTML rendering. To modify these, you can access the underlying control through the **WebCalendar** property and modify the settings directly.

Example

The following code example shows how the **SelectionMode** property in the page load code block allows the user to select a day, a week, or a month block of time. This example sets the **BorderStyle** and **BackColor** properties of the **Calendar** class to distinguish the user selection.

Note:

The following code sample uses the single-file code model and may not work correctly if copied directly into a code-behind file. This code sample must be copied into an empty text file that has an .aspx extension.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Import Namespace="System.Drawing" %>
<%@ Import Namespace="System.Web.UI.WebControls" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    ' Display the day header
    Calendar1.ShowDayHeader = True
    ' Allow the user to select a week or a month at a time.
    Calendar1.SelectionMode = CalendarSelectionMode.DayWeekMonth
    ' Set the BorderStyle and BorderColor properties.
    Calendar1.WebCalendar.DayStyle.BorderStyle = BorderStyle.Solid
    Calendar1.WebCalendar.DayStyle.BorderColor = Color.Cyan
    Calendar1.CalendarEntryText = "Your birthdate"
    Calendar1.FirstDayOfWeek = System.Web.UI.WebControls.FirstDayOfWeek.Friday
    Calendar1.VisibleDate = DateTime.Parse("7/1/2004")
End Sub

Protected Sub ShowChanges(ByVal sender As Object, ByVal e As EventArgs)
    TextView1.Text = "The date you selected is " & Calendar1.SelectedDate.ToShortDateString()
    ' Distinguish the selected block using colors.
    Calendar1.WebCalendar.SelectedDayStyle.BackColor = Color.LightGreen
    Calendar1.WebCalendar.SelectedDayStyle.BorderColor = Color.Gray
    Calendar1.WebCalendar.DayStyle.BorderColor = Color.Blue
End Sub

Protected Sub Command1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim currentDay As Integer = Calendar1.VisibleDate.Day
    Dim currentMonth As Integer = Calendar1.VisibleDate.Month
    Dim currentYear As Integer = Calendar1.VisibleDate.Year
    Calendar1.SelectedDates.Clear()
    ' Loop through current month and add all Wednesdays to the collection.
    Dim i As Integer
    For i = 1 To System.DateTime.DaysInMonth(currentYear, currentMonth)
        Dim targetDate As New DateTime(currentYear, currentMonth, i)
        If targetDate.DayOfWeek = DayOfWeek.Wednesday Then
            Calendar1.SelectedDates.Add(targetDate)
        End If
    Next i
    TextView1.Text = "Selection Count = " & Calendar1.SelectedDates.Count.ToString()
End Sub
</script>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Import Namespace="System.Drawing" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<script runat="server">
protected void Page_Load(object sender, EventArgs e)
{
    // Display the day header
    Calendar1.ShowDayHeader = true;
    // This allows the user to select a week or a month at a time.
    Calendar1.SelectionMode = CalendarSelectionMode.DayWeekMonth;
    // Set the BorderStyle and BorderColor properties.
    Calendar1.WebCalendar.DayStyle.BorderStyle = BorderStyle.Solid;
    Calendar1.WebCalendar.DayStyle.BorderColor = Color.Cyan;
    Calendar1.CalendarEntryText = "Your birthdate";
    Calendar1.VisibleDate = DateTime.Parse("7/1/" + DateTime.Now.Year.ToString());
}

protected void ShowChanges(Object sender, EventArgs e)
{
    TextView1.Text = "The date you selected is " + Calendar1.SelectedDate.ToShortDateString();
    // Distinguish the selected block using colors.
    Calendar1.WebCalendar.SelectedDayStyle.BackColor = Color.LightGreen;
    Calendar1.WebCalendar.SelectedDayStyle.BorderColor = Color.Gray;
    Calendar1.WebCalendar.DayStyle.BorderColor = Color.Blue;
}

protected void Command1_Click(object sender, EventArgs e)
{
    int currentDay = Calendar1.VisibleDate.Day;
    int currentMonth = Calendar1.VisibleDate.Month;
    int currentYear = Calendar1.VisibleDate.Year;
    Calendar1.SelectedDates.Clear();

    // Add all Wednesdays to the collection.
    for (int i = 1; i <= System.DateTime.DaysInMonth(currentYear, currentMonth); i++)
    {
        DateTime targetDate = new DateTime(currentYear, currentMonth, i);
        if (targetDate.DayOfWeek == DayOfWeek.Wednesday)
            Calendar1.SelectedDates.Add(targetDate);
    }
    TextView1.Text = "Selection Count =" + Calendar1.SelectedDates.Count.ToString();
}
</script>

<html><body>
<mobile:form id="form1" runat="server">
    <mobile:Calendar id="Calendar1" runat="server" OnSelectionChanged="ShowChanges" />
    <mobile:TextView runat="server" id="TextView1" />
    <mobile:Command id="Command1" OnClick="Command1_Click" Runat="server">Select Weekdays
    </mobile:Command>
</mobile:form>
</body></html>
```

Command Class

Creates a user interface element that enables users to invoke ASP.NET event handlers and it provides a means to post user input from UI elements back to the server.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class Command **Inherits** TextControl **Implements** IPostBackEventHandler, IPostBackDataHandler

C#

public class Command : TextControl, IPostBackEventHandler, IPostBackDataHandler

The **Command** control displays as an interactive UI element on the requesting device. The label of the UI element comes from the **Text** property, which is inherited from the **TextControl** base class.

 Caution:

Avoid using special characters in ASP.NET mobile Web page URLs. The HREF tags generated for posting **Command** events back to the server are not strictly validated. For example, a URL that includes spaces results in the generation of WML that cannot be handled by some WML browsers.

Example

The following code example demonstrates how to attach command events. Clicking either of the **Command** buttons raises the **OnItemCommand** event. The user-defined function uses the **CommandEventArgs** argument to see which **Command** button was clicked.

 Note:

The following code example uses the single-file code model and may not work correctly if copied directly into a code-behind file. This code example must be copied into an empty text file that has an .aspx extension.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.Mobile" %>

<script runat="server">
Public Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim caps As System.Web.Mobile.MobileCapabilities = CType(Request.Browser, MobileCapabilities)
    If caps.MaximumSoftkeyLabelLength = 5 Then
        Command1.SoftkeyLabel = "Click"
    ElseIf caps.MaximumSoftkeyLabelLength > 5 Then
        Command1.SoftkeyLabel = "Submit"
    End If
End Sub

Private Sub Command_Click(ByVal sender As Object, ByVal e As CommandEventArgs)
    Dim txt As String = "You clicked Button{0}. ({1} points)"
    If e.CommandName.ToString() = "Command1" Then
        Label1.Text = String.Format(txt, 1, e.CommandArgument)
    ElseIf e.CommandName.ToString() = "Command2" Then
        Label1.Text = String.Format(txt, 2, e.CommandArgument)
    End If
End Sub
</script>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.Mobile" %>

<script runat="server">
public void Page_Load(Object sender, EventArgs e)
{
    MobileCapabilities caps = (MobileCapabilities)Request.Browser;
    if (caps.MaximumSoftkeyLabelLength == 5)
    {
        Command1.SoftkeyLabel = "Click";
    }
    else if (caps.MaximumSoftkeyLabelLength > 5)
    {
        Command1.SoftkeyLabel = "Submit";
    }
}

void Command_Click(object sender, CommandEventArgs e)
{
    string txt = "You clicked Button{0}. ({1} points)";
```

```
        if (e.CommandName.ToString() == "Command1")
        {
            Label1.Text = String.Format(txt, 1, e.CommandArgument);
        }
        else if (e.CommandName.ToString() == "Command2")
        {
            Label1.Text = String.Format(txt, 2, e.CommandArgument);
        }
    }
</script>

<html>
<body>
<mobile:form id="form1" runat="server">
    <mobile:Label id="Label1" runat="server"> Click a button </mobile:Label>
    <mobile:Label id="Label2" runat="server" />

    <mobile:Command id="Command1" Format="Button" OnItemCommand="Command_Click"
    CommandName="Command1" runat="server" Text="Button1" CommandArgument="70" />
    <mobile:Command id="Command2" Format="Link" OnItemCommand="Command_Click"
    CommandName="Command2" runat="server" Text="Button2" CommandArgument="50" />
</mobile:form>
</body>
</html>
```

CompareValidator Class

Determines validity by comparing a specific field in one control to a specific field in another control, using a specifiable comparison operator.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class CompareValidator **Inherits** BaseValidator

C#

public class CompareValidator : BaseValidator

Use compare validators to ensure that the values of two text boxes provide the same values, such as confirming a new password.

This class behaves identically to the ASP.NET **CompareValidator** control.

By default, **Command** controls on a form raise any validator controls on the form to perform validation when the form is submitted to the server. To disable automatic validation, set the **CausesValidation** property on the **Command** controls to **false**.

Validation succeeds if the input control is empty. Use a **RequiredFieldValidator** control to require the user to enter data into the input control.

Example

The following example code uses a **CompareValidator** control (CompareValidator1) to check whether the two text boxes have the same value and alerts the user if they are different.

Security Note:

This example has a text box that accepts user input, which is a potential security threat. By default, ASP.NET Web pages validate that user input does not include script or HTML elements.

Visual Basic
<pre><%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %> <%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %> <html> <body> <mobile:form id="form1" runat="server"> <mobile:Label id="Label1" runat="server">Enter values </mobile:Label> <mobile:TextBox id="TextBox1" runat="server" Text="abc" /> <mobile:TextBox id="TextBox2" runat="server" Text="xyz" /> <mobile:Command id="Command1" runat="server" Text="OK" /> <mobile:CompareValidator ID="CompareValidator1" Runat="server" ErrorMessage="Values are different" Operator="Equal" ControlToCompare="TextBox1" ControlToValidate="TextBox2" /> </mobile:form> </body> </html></pre>
C#
<pre><%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %> <%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %> <html> <body> <mobile:form id="form1" runat="server"> <mobile:Label id="Label1" runat="server">Enter values </mobile:Label> <mobile:TextBox id="TextBox1" runat="server" Text="abc" /> <mobile:TextBox id="TextBox2" runat="server" Text="xyz" /> <mobile:Command id="Command1" runat="server" Text="OK" /> <mobile:CompareValidator ID="CompareValidator1" Runat="server" ErrorMessage="Values are different" Operator="Equal" ControlToCompare="TextBox1" ControlToValidate="TextBox2" /> </mobile:form> </body> </html></pre>

CustomValidator Class

Provides a control that can perform custom validation against another control.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic
<pre>Public Class CustomValidator Inherits BaseValidator</pre>
C#
<pre>public class CustomValidator : BaseValidator</pre>

This class behaves identically to the **System.Web.UI.WebControls.CustomValidator** Web server control. Developers can choose their own common language runtime delegate to use for validation.

By default, **Command** controls on a form raise validator events on the form to perform validation when the form is submitted to the server. To disable automatic validation, set the **CausesValidation** property on the **Command** controls to **false**.

Example

The following example checks whether the value that a user places into the **TextBox** control is an even number. If the value is an even number, then the page is valid. If not, the page is not valid, and the **CustomValidator** displays the **ErrorMessage** property.

 Note:

The following code sample uses the single-file code model and may not work correctly if copied directly into a code-behind file. This code sample must be copied into an empty text file that has an .aspx extension.

 Security Note:

This example has a text box that accepts user input, which is a potential security threat. By default, ASP.NET Web pages validate that user input does not include script or HTML elements.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<script runat="server">
' If the page validates, go to page 2
Protected Sub Submit_Click(ByVal sender As Object, ByVal e As EventArgs)
    If (Page.IsValid) Then
        ActiveForm = Form2
    End If
End Sub

Private Sub ServerValidate(ByVal source As Object, ByVal args As ServerValidateEventArgs)
    ' Convert the text to a number
    Dim num As Integer
    Integer.TryParse(numberBox.Text, num)
    ' Test for an even number
    If (num > 0) Then
        args.IsValid = ((num Mod 2) = 0)
    Else
        args.IsValid = False
    End If
End Sub
</script>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<script runat="server">
// If the page validates, go to page 2
protected void Submit_Click(Object sender, EventArgs e)
{
    if (Page.IsValid) { ActiveForm = Form2; }
}

private void ServerValidate(object source, ServerValidateEventArgs args)
{
    // Convert the text to a number
    Int num;
    Int32.TryParse(numberBox.Text, out num);
    // Test for an even number
    if (num > 0) args.IsValid = ((num % 2) == 0); else args.IsValid = false;
}
</script>

<html><body>
<mobile:form id="Form1" runat="server">
    <mobile:Label ID="Label1" runat="server"> Please enter an even number greater than zero.</mobile:Label>
    <mobile:TextBox ID="numberBox" Runat="server" Numeric="true" MaxLength="2" />
    <mobile:CustomValidator ID="CustomValidator1" ControlToValidate="numberBox"
        OnServerValidate="ServerValidate" runat="server"> Your number is not an even number.
    </mobile:CustomValidator>
    <mobile:Command ID="Command1" runat="server" OnClick="Submit_Click"> Submit </mobile:Command>
</mobile:form>
<mobile:Form id="Form2" runat="server">
    <mobile:Label ID="Label2" runat="server"> Your number is an even number. </mobile:Label>
</mobile:Form>
</body></html>
```

Provides the capability to group controls together.

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Public Class Form **Inherits** Panel **Implements** ITemplateable, IPostBackEventHandler

```
public class Form : Panel, ITemplateable, IPostBackEventHandler
```

Example

 **Note:**

The following code example uses the single-file code model and may not work correctly if copied directly into a code-behind file. This code example must be copied into an empty text file that has an .aspx extension.

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.UI.MobileControls" %>
<%@ Import Namespace="System.Drawing" %>

<script runat="server">
Private Sub Form1_Activate(ByVal sender As Object, ByVal e As EventArgs)
    Dim viewText As String = "You have viewed this Form {0} times."
    If (count = 0) Then
        ' First viewing
        message2.Text = "Welcome to the Form Sample"

    Else
        ' subsequent viewings
        message2.Text = String.Format(viewText, (count + 1).ToString())

    End If

    ' Format the form
    Form1.Alignment = Alignment.Center
    Form1.Wrapping = Wrapping.NoWrap
End Sub
End Page

```

```
Form1.BackColor = Color.LightBlue
Form1.ForeColor = Color.Blue
Form1.Paginate = True

' Create an array and add the tasks to it.
Dim arr As ArrayList = New ArrayList()
arr.Add(New Task("Verify transactions", "Done"))
arr.Add(New Task("Check balance sheet", "Scheduled"))
arr.Add(New Task("Send report", "Pending"))

' Bind the SelectionList to the array.
SelectionList1.DataValueField = "Status"
SelectionList1.DataTextField = "TaskName"
SelectionList1.DataSource = arr
SelectionList1.DataBind()

End Sub

Private Sub Form1_Deactivate(ByVal sender As Object, ByVal e As EventArgs)
    count += 1
End Sub

Private Sub Form2_Activate(ByVal sender As Object, ByVal e As EventArgs)
    Form2.BackColor = Color.DarkGray
    Form2.ForeColor = Color.White
    Form2.Font.Bold = BooleanOption.True
End Sub

Protected Sub Command1_OnSubmit(ByVal sender As Object, ByVal e As EventArgs)
    Dim i As Integer
    message2.Text = "FORM RESULTS:"
    message2.Font.Bold = BooleanOption.True

    ' Create a string and a TextView control
    Dim txtView As TextView = New TextView()
    Dim txt As String = ""
    Dim spec As String = "{0} is {1}<br />"

    ' Display a list of selected items with values
    For i = 0 To SelectionList1.Items.Count - 1
        ' Get the ListItem
        Dim itm As MobileListItem = SelectionList1.Items(i)
        ' List the selected items and values
        If itm.Selected Then
            txt &= String.Format(spec, itm.Text, itm.Value)
        End If
    Next

    ' Put the text into the TextView
    txtView.Text = txt
    ' Add the TextView to the form
    Form1.Controls.Add(txtView)
    ' Hide unnecessary controls
    SelectionList1.Visible = False
    link1.Visible = False
    Command1.Visible = False
End Sub

' Property to persist the count between postbacks
Private Property count() As Integer
    Get
        Dim o As Object = ViewState("FormCount")
        If IsNothing(o) Then
            Return 0
        Else
            Return CType(o, Integer)
        End If
    End Get

    Set(ByVal value As Integer)
        ViewState("FormCount") = value
    End Set
End Property
```

```
End Set
End Property

' A custom class for the task array
Private Class Task
    Private _TaskName As String
    Private _Status As String
    Public Sub New(ByVal TaskName As String, ByVal Status As String)
        _TaskName = TaskName
        _Status = Status
    End Sub

    Public ReadOnly Property TaskName() As String
        Get
            Return _TaskName
        End Get
    End Property

    Public ReadOnly Property Status() As String
        Get
            Return _Status
        End Get
    End Property
End Class
</script>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.UI.MobileControls" %>
<%@ Import Namespace="System.Drawing" %>

<script runat="server">
private void Form1_Activate(object sender, EventArgs e)
{
    string viewText = "You have viewed this Form {0} times.";
    if (count == 0)
        // First viewing
        message2.Text = "Welcome to the Form Sample";
    else
        // subsequent viewings
        message2.Text = String.Format(viewText, (count + 1).ToString());

    // Format the form
    Form1.Alignment = Alignment.Center;
    Form1.Wrapping = Wrapping.NoWrap;
    Form1.BackColor = Color.LightBlue;
    Form1.ForeColor = Color.Blue;
    Form1.Paginate = true;

    // Create an array and add the tasks to it.
    ArrayList arr = new ArrayList();
    arr.Add(new Task("Verify transactions", "Done"));
    arr.Add(new Task("Check balance sheet", "Scheduled"));
    arr.Add(new Task("Send report", "Pending"));

    // Bind the SelectionList to the array.
    SelectionList1.DataValueField = "Status";
    SelectionList1.DataTextField = "TaskName";
    SelectionList1.DataSource = arr;
    SelectionList1.DataBind();
}

private void Form1_Deactivate(object sender, EventArgs e)
{
    count++;
}
```

```
private void Form2_Activate(object sender, EventArgs e)
{
    Form2.BackColor = Color.DarkGray;
    Form2.ForeColor = Color.White;
    Form2.Font.Bold = BooleanOption.True;
}

protected void Command1_OnSubmit(object sender, EventArgs e)
{
    message2.Text = "FORM RESULTS:";
    message2.Font.Bold = BooleanOption.True;
    // Display a list of selected items with values
    for (int i = 0; i < SelectionList1.Items.Count; i++)
    {
        // Create a string and a TextView control
        TextView txtView = new TextView();
        string txt = "";
        string spec = "{0} is {1}<br />";
        // Display a list of selected items with values
        // Get the list item
        MobileListItem itm = SelectionList1.Items[i];
        // List the selected items and values
        if (itm.Selected)
        {
            txt += String.Format(spec, itm.Text, itm.Value);
        }
        // Put the text into the TextView
        txtView.Text = txt;
        // Add txtView to the form
        Form1.Controls.Add(txtView);
    }
    // Hide unnecessary controls
    SelectionList1.Visible = false;
    link1.Visible = false;
    Command1.Visible = false;
}

// Property to persist the count between postbacks
private int count
{
    get
    {
        object o = ViewState["FormCount"];
        return o == null ? 0 : (int)o;
    }
    set
    {
        ViewState["FormCount"] = value;
    }
}

// A custom class for the task array
private class Task
{
    private String _TaskName;
    private String _Status;
    public Task(String TaskName, String Status)
    {
        _TaskName = TaskName;
        _Status = Status;
    }
    public String TaskName
    {
        get
        {
            return _TaskName;
        }
    }
    public String Status
    {

```

```
        get
        {
            return _Status;
        }
    }
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml"><body>
<!-- The first form: Form1 -->
<mobile:Form ID="Form1" Runat="server" OnDeactivate="Form1_Deactivate" OnActivate="Form1_Activate">
    <mobile:Label ID="message1" Runat="server"> Welcome to ASP.NET </mobile:Label>
    <mobile:Label ID="message2" Runat="server" />
    <mobile:SelectionList Runat="server" ID="SelectionList1" ForeColor="red" SelectType="CheckBox" />
    <mobile:Link ID="link1" Runat="server" NavigateUrl="#Form2" Text="Next Form" /><br />
    <mobile:Command ID="Command1" Runat="server" Text="Submit" OnClick="Command1_OnSubmit" />
</mobile:Form>

<!-- The second form: Form2 -->
<mobile:Form ID="Form2" Runat="server" OnActivate="Form2_Activate">
    <mobile:Label ID="message4" Runat="server"> Welcome to ASP.NET </mobile:Label>
    <mobile:Link ID="Link2" Runat="server" NavigateUrl="#Form1" Text="Back" />
</mobile:Form>
</body></html>
```

Image Class

Displays an image on a mobile Web page.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class Image **Inherits** MobileControl **Implements** IPostBackEventHandler

C#

public class Image : MobileControl, IPostBackEventHandler

The **Image** class provides a way to choose an image file based on device characteristics. For certain devices, you can specify built-in images by using a **symbol:** prefix as part of the URL in the **ImageUrl** property. For more information, see the Device-Specific section of the **Image**.

Note:

If your application relies on cookieless sessions, or might receive requests from browsers that require cookieless sessions, using a tilde ("~") in a path can inadvertently result in creating a new session and potentially losing session data. To set a property with a path such as ("~/path"), resolve the path using **ResolveUrl** ("~/path") before assigning it to the property.

Example

The following code sample code shows how to use a **<DeviceSpecific>** control within an Image control to specify different images for different devices. If you have available a file named Sunshine.gif, it will appear in Internet Explorer. If you view the page on a WML device such as an OpenWave phone or a CHTML device, it will appear as a sun icon. The page also displays the name of the browser and the resolved name of the image.

Note:

The following code sample uses the single-file code model and may not work correctly if copied directly into a code-behind file. This code sample must be copied into an empty text file that has an .aspx extension.

```
C#
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<script runat="server">
protected void Page_Load(object sender, EventArgs e)
{
    string spec1 = "Device: {0}";
    string spec2 = "Image source: {0}";
    if (!IsPostBack)
    {
        Label1.Text = string.Format(spec1, Device.Browser);
        Label2.Text = string.Format(spec2, Image1.ImageUrl);
    }
}
</script>

<html><body>
<mobile:form id="form1" runat="server">
    <mobile:Image ID="Image1" Runat="server" AlternateText="Sunshine">
        <DeviceSpecific ID="imgDevSec" Runat="server">
            <Choice Filter="isWML11" ImageUrl="symbol:44" />
            <Choice Filter="isCHTML10" ImageUrl="symbol:63726" />
            <Choice ImageUrl="sunshine.gif" />
        </DeviceSpecific>
    </mobile:Image>
    <mobile:Label ID="Label1" Runat="server" />
    <mobile:Label ID="Label2" Runat="server" />
</mobile:form>
</body></html>
```

The following is the <deviceFilters> section of the Web.config file:

```
<deviceFilters>
    <filter name="isWML11" compare="PreferredRenderingType" argument="wml11" />
    <filter name="isCHTML10" compare="PreferredRenderingType" argument="html10" />
</deviceFilters>
```

Link Class

Represents a hyperlink to another **MobileControl.Form** on the same mobile page or to an arbitrary URI.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

```
Visual Basic
Public Class Link Inherits TextControl Implements IPostBackEventHandler
C#
public class Link : TextControl, IPostBackEventHandler
```

On all devices, the link is rendered in such a way that the **Text** property displays on the device to the user. When the user invokes the link, the browser navigates to the specified **NavigateUrl** property. The application must contain code that verifies that a link is browsable on a particular device. For example, the link <http://www.microsoft.com> is not browsable on a WML-capable device.

If the **Text** property is empty, the value in the **NavigateUrl** property is also used for the **Text** property. Specifically, when the text writer renders a link, it first checks the **Text** property for the text to display; if that property is empty, it displays the value of the **NavigateUrl** property.

 **Note:**

If your application relies on cookieless sessions, or might receive requests from devices that require cookieless sessions, using a tilde ("~") in a path can inadvertently result in creating a new session and potentially losing session data. To set a property with a path such as ("~/path"), resolve the path using **ResolveUrl**("~/path") before assigning it to the property.

Label Class

Provides control capability to represent a label control for displaying text on a mobile device.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class Label **Inherits** TextControl

C#

public class Label : TextControl

The **Text** property, which is inherited from the **TextControl** class, provides the string that is rendered as the label.

List Class

Renders a list of items as either a static display or an interactive list.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class List **Inherits** PagedControl **Implements** INamingContainer, ITemplateable, IPostBackEventHandler

C#

public class List : PagedControl, INamingContainer, ITemplateable, IPostBackEventHandler

This control supports templated rendering by using device template sets and internal pagination.

Example

The following code example demonstrates how an array binds and fills a **List**. Notice that you can programmatically set the **DataTextField** and **DataValueField** properties of the **List** object.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
' Persist across multiple postbacks.
Private Shared doneCount, schedCount, pendCount As Integer

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        ' Set the DataMembers of the List
        List1.DataValueField = "Status"
        List1.DataTextField = "TaskName"

        ' Create an ArrayList of task data
        Dim arr As ArrayList = New ArrayList()
        arr.Add(New Task("Define transactions", "scheduled"))
        arr.Add(New Task("Verify transactions", "scheduled"))
        arr.Add(New Task("Check balance sheet", "scheduled"))
        arr.Add(New Task("Compile balance sheet", "scheduled"))
    End If
End Sub
```

```
arr.Add(New Task("Prepare report", "scheduled"))
arr.Add(New Task("Send report", "scheduled"))

' Bind the array to the list
List1.DataSource = arr List1.DataBind()

Const spec As String = "Start: {0} tasks are done, {1} " & "tasks are scheduled, and {2} tasks are
pending."
Label2.Text = String.Format(spec, doneCount, schedCount, pendCount)
List1.Decoration = ListDecoration.Bulleted

End If
End Sub

Private Sub Status_ItemCommand(ByVal sender As Object, ByVal e As ListCommandEventArgs)
Const spec As String = "You now have {0} tasks done, {1} " & "tasks scheduled, and {2} tasks pending."
' Move selection to next status toward 'done'
Select Case e.ListItem.Value
Case "scheduled"
    schedCount -= 1
    pendCount += 1
    e.ListItem.Value = "pending"
Case "pending"
    pendCount -= 1
    doneCount += 1
    e.ListItem.Value = "done"

End Select

' Show the status of the current task
Label1.Text = e.ListItem.Text & " is " & e.ListItem.Value
' Show current selection counts
Label2.Text = String.Format(spec, doneCount, schedCount, pendCount)
End Sub

Private Sub Status_DataBinding(ByVal sender As Object, ByVal e As ListDataBindEventArgs)
' Increment initial counts
Select Case e.ListItem.Value
Case "done" doneCount += 1
Case "scheduled" schedCount += 1
Case "pending" pendCount += 1

End Select
End Sub

' Custom class for the ArrayList items
Private Class Task
Private _TaskName, _Status As String
Public Sub New(ByVal TaskName As String, ByVal Status As String)
_TaskName = TaskName
_Status = Status

End Sub
Public ReadOnly Property TaskName() As String
Get
Return _TaskName

End Get
End Property
Public ReadOnly Property Status() As String
Get
Return _Status

End Get
End Property
End Class
</script>

C#

<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
// Persist across multiple postbacks.
private static int doneCount, schedCount, pendCount;
```

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        // Set the DataMembers of the List
        List1.DataValueField = "Status";
        List1.DataTextField = "TaskName";

        // Create an ArrayList of task data
        ArrayList arr = new ArrayList();
        arr.Add(new Task("Define transactions", "scheduled"));
        arr.Add(new Task("Verify transactions", "scheduled"));
        arr.Add(new Task("Check balance sheet", "scheduled"));
        arr.Add(new Task("Compile balance sheet", "scheduled"));
        arr.Add(new Task("Prepare report", "scheduled"));
        arr.Add(new Task("Send report", "scheduled"));

        // Bind the array to the list
        List1.DataSource = arr; List1.DataBind();

        const string spec = "Start: {0} tasks are done, {1} " + "tasks are scheduled, and {2} tasks are pending.";
        Label2.Text = String.Format(spec, doneCount, + schedCount, pendCount);
        List1.Decoration = ListDecoration.Bulleted;
    }
}

private void Status_ItemCommand(object sender, ListCommandEventArgs e)
{
    const string spec = "You now have {0} " + "tasks done, {1} tasks scheduled, and " + "{2} tasks pending.";

    // Move selection to next status toward 'done'
    switch (e.ListItem.Value)
    {
        case "scheduled":
            schedCount -= 1;
            pendCount += 1;
            e.ListItem.Value = "pending";
            break;

        case "pending":
            pendCount -= 1;
            doneCount += 1;
            e.ListItem.Value = "done";
            break;
    }

    // Show the status of the current task
    Label1.Text = e.ListItem.Text + " is " + e.ListItem.Value;
    // Show current selection counts
    Label2.Text = String.Format(spec, doneCount, schedCount, pendCount);
}

private void Status_DataBinding(object sender, ListDataBindEventArgs e)
{
    // Increment initial counts
    switch (e.ListItem.Value)
    {
        case "done":
            doneCount += 1;
            break;

        case "scheduled":
            schedCount += 1;
            break;

        case "pending":
            pendCount += 1;
            break;
    }
}

// Custom class for the ArrayList items
private class Task
{
    private string _TaskName;
    private string _Status;
```

```
public Task(string taskName, string status)
{
    _TaskName = taskName;
    _Status = status;
}

public string TaskName
{
    get { return _TaskName; }
}

public string Status
{
    get { return _Status; }
}
}
</script>

<html><body>
<mobile:form id="form1" runat="server">
    <mobile:Label ID="Label3" Runat="server"> Click a task to change its status from scheduled to pending or
    from pending to done: </mobile:Label>
    <mobile:List runat="server" id="List1" OnItemCommand="Status_ItemCommand"
    OnItemDataBind="Status_DataBinding" />
    <mobile:Label runat="server" id="Label1" ForeColor="green" Font-Italic="true" />
    <mobile:Label id="Label2" runat="server" />
</mobile:form>
</body></html>
```

MobilePage Class

Serves as the base class for all ASP.NET mobile Web Forms pages.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class MobilePage Inherits Page

C#

public class MobilePage : Page

The **MobilePage** class inherits from the ASP.NET **Page** class. To specify a mobile page and use ASP.NET mobile controls, an ASP.NET mobile Web Forms page must contain the following page directive.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="c#" %>
```

The **Inherits** attribute is required. The **Language** attribute is set to the language used on the page, if needed.

Note:

ASP.NET mobile pages allow multiple mobile forms on each page, whereas ASP.NET Web pages allow only one form per page.

ObjectList Class

Enables you to specify multiple fields for display, per item in an object list.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class ObjectList **Inherits** PagedControl **Implements** INamingContainer, ITemplateable, IPostBackEventHandler

C#

public class ObjectList : PagedControl, INamingContainer, ITemplateable, IPostBackEventHandler

Much of the behavior of the **ObjectList**, including support for templated rendering through device template sets and internal pagination, is similar to the behavior of the **List**.

Example

The following code example demonstrates how to create an array of a user-defined class and then bind it to an **ObjectList** object when the page loads. It also shows how the list and details views display the commands. For this example, there is also a button that displays a form with a list of all the fields using the **AllFields** property.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
    Dim bakeryCount, dairyCount, produceCount As Integer

    Private Sub Page_Load(ByVal o As Object, ByVal e As EventArgs)
        If Not IsPostBack Then

            ' Create an array and bind it to the list
            Dim arr As New ArrayList()
            arr.Add(New GroceryItem ("Bakery", "Rolls", "On Sale"))
            arr.Add(New GroceryItem ("Dairy", "Eggnog", "Half price"))
            arr.Add(New GroceryItem ("Produce", "Apples", "A dollar a bushel"))
            arr.Add(New GroceryItem ("Bakery", "Bread", "On Sale"))
            List1.DataSource = arr
            List1.DataBind()

            ' To show only one field on opening page, comment the next line
            List1.TableFields = "Item;Department"
            List1.LabelField = "Department"

            ' Display a report after items are databound
            Const txt As String = "Number of items by Department<br>Produce: " + "{0}<br />Dairy: {1}<br />Bakery: {2}"
            TextView2.Text = String.Format(txt, produceCount, dairyCount, bakeryCount)

        End If
    End Sub

    ' Command event for buttons
    Private Sub List1_Click(ByVal sender As Object, ByVal e As ObjectListCommandEventArgs)
        If e.CommandName = "Reserve" Then
            ActiveForm = Form2
        ElseIf e.CommandName = "Buy" Then
            ActiveForm = Form3
        Else
            ActiveForm = Form4
        End If
    End Sub
End Sub
```

```
' Count items in each department
Private Sub List1_ItemDataBind(ByVal sender As Object, ByVal e As ObjectListDataBindEventArgs)
    Select Case CType(e.DataItem, GroceryItem).Department
        Case "Bakery" bakeryCount += 1
        Case "Dairy" dairyCount += 1
        Case "Produce" produceCount += 1
    End Select
End Sub

Private Sub AllFields_Click(ByVal sender As Object, ByVal e As EventArgs)
    ActiveForm = Form5
    Dim spec As String = "{0}: {1}<br/>"
    Dim flds As IObjectListFieldCollection = List1.AllFields
    Dim i As Integer

    For i = 0 To flds.Count - 1
        TextView1.Text += String.Format(spec, (i + 1), flds(i).Title)
    Next
End Sub

' Structure for ArrayList records
Private Class GroceryItem
    ' A private class for the Grocery List
    Private _department, _item, _status As String

    Public Sub New(ByVal department As String, ByVal item As String, ByVal status As String)
        _department = department
        _item = item
        _status = status
    End Sub

    Public ReadOnly Property Department() As String
        Get
            Return _department
        End Get
    End Property

    Public ReadOnly Property Item() As String
        Get
            Return _item
        End Get
    End Property

    Public ReadOnly Property Status() As String
        Get
            Return _status
        End Get
    End Property
End Class
</script>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
    int bakeryCount = 0, dairyCount = 0, produceCount = 0;
    public void Page_Load(Object o, EventArgs e)
    {
        if (!IsPostBack)
        {
            // Create an array and bind it to the list
            ArrayList arr = new ArrayList();
            arr.Add(new GroceryItem("Bakery", "Rolls", "On Sale"));
            arr.Add(new GroceryItem("Dairy", "Eggnog", "Half price"));
            arr.Add(new GroceryItem("Produce", "Apples", "A dollar a bushel"));
            arr.Add(new GroceryItem("Bakery", "Bread", "On Sale"));
            List1.DataSource = arr;
            List1.DataBind();
            // To show only one field on opening page, comment the next line
            List1.TableFields = "Item;Department";
        }
    }
}
```

```
List1.LabelField = "Department";
// Display a report after items are databound
string txt = "Number of items by Department<br>Produce: {0}<br />" + "Dairy: {1}<br />Bakery: {2}";
TextView2.Text = String.Format(txt, produceCount, dairyCount, bakeryCount);
}
}

// Command event for buttons
public void List1_Click(Object sender, ObjectListCommandEventArgs e)
{
    if (e.CommandName == "Reserve")
        ActiveForm = Form2;
    else if (e.CommandName == "Buy")
        ActiveForm = Form3;
    else
        ActiveForm = Form4;
}

// Count items in each department
private void List1_ItemDataBind(object sender, ObjectListDataBindEventArgs e)
{
    switch (((GroceryItem)e.DataItem).Department)
    {
        case "Bakery":      bakeryCount++;
                           break;
        case "Dairy":      dairyCount++;
                           break;
        case "Produce":    produceCount++;
                           break;
    }
}

private void AllFields_Click(object sender, EventArgs e)
{
    ActiveForm = Form5;
    string spec = "{0}: {1}<br/>"; IObjectListFieldCollection
    flds = List1.AllFields;
    for (int i = 0; i < flds.Count; i++)
        TextView1.Text += String.Format(spec, (i + 1), flds[i].Title);
}

// Structure for ArrayList records
private class GroceryItem
{
    // A private class for the Grocery List
    private String _department, _item, _status;
    public GroceryItem(string department, string item, string status)
    {
        _department = department;
        _item = item;
        _status = status;
    }
    public String Department
    {
        get { return _department; }
    }
    public String Item
    {
        get { return _item; }
    }
    public String Status
    {
        get { return _status; }
    }
}
</script>

<html><body>
<mobile:Form id="Form1" runat="server" BackColor="LightBlue">
<mobile:ObjectList id="List1" runat="server" OnItemClick="List1_Click" OnItemDataBind="List1_ItemDataBind">
```

```
<DeviceSpecific ID="DeviceSpecific1" Runat="server">
    <!-- See Web.config for filters -->
    <Choice Filter="isWML11" CommandStyle-Font-Bold="NotSet" />
    <Choice CommandStyle-Font-Bold="true" CommandStyle-Font-Name="Arial" />
</DeviceSpecific>
<Command Name="Reserve" Text="Reserve" />
<Command Name="Buy" Text="Buy" />
</mobile:ObjectList>
<mobile:Command ID="AllFieldsCmd" Runat="server" OnClick="AllFields_Click"> List All Fields</mobile:Command>
<mobile:TextView ID="TextView2" Runat="server" />
</mobile:Form>

<mobile:Form id="Form2" runat="server" BackColor="LightBlue">
    <mobile:Label id="ResLabel" runat="server" text="Sale item reservation system coming soon!" />
    <mobile:Link id="ResLink" NavigateURL="#Form1" runat="server" text="Return" />
</mobile:Form>

<mobile:Form id="Form3" runat="server" BackColor="LightBlue">
    <mobile:Label id="BuyLabel" runat="server" Text="Online purchasing system coming soon!" />
    <mobile:Link ID="BuyLink" NavigateURL="#Form1" Runat="server" text="Return" />
</mobile:Form>

<mobile:Form id="Form4" Runat="server" BackColor="LightBlue">
    <mobile:Label ID="DefLabel" Runat="server" Text="Detailed item descriptions will be here soon!" />
    <mobile:Link ID="DefLink" NavigateURL="#Form1" Runat="server" Text="Return" />
</mobile:Form>

<mobile:Form ID="Form5" Runat="server">
    <mobile:Label ID="Label1" Runat="server"> List of AllFields:</mobile:Label>
    <mobile:TextView ID="TextView1" Runat="server" />
    <mobile:Link ID="Link1" Runat="server" NavigateUrl="#Form1" Text="Return"></mobile:Link>
</mobile:Form>
</body></html>
```

This is a sample Web.config file with several device specific filters.

```
<configuration>
<system.web>
    <compilation debug="true" />
    <authentication mode="Windows" />
    <customErrors mode="Off" />
    <httpRuntime useFullyQualifiedRedirectUrl="true" />
    <mobileControls cookielessDataDictionaryType="System.Web.Mobile.CookielessData" />

    <deviceFilters>
        <!-- Only a few filters are provided here -->
        <filter name="isJPhone" compare="Type" argument="J-Phone" />
        <filter name="isHTML32" compare="PreferredRenderingType" argument="html32" />
        <filter name="isWML11" compare="PreferredRenderingType" argument="wml11" />
    </deviceFilters>
</system.web>
</configuration>
```

Panel Class

Provides a container for organizing controls in a mobile Web forms page.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class Panel **Inherits** MobileControl **Implements** ITemplateable

C#

public class Panel : MobileControl, ITemplateable

Panels are container controls that can be nested. Attributes set on a panel are inherited by the controls that are contained in that panel.

You can use a panel for any of the following functions:

- Grouping controls logically so that they can be shown or hidden.
- Defining a convenient container where controls can be created or removed dynamically.
- Using a single point to apply style attributes to a set of controls, by setting them on the panel.
Because style inheritance applies to panels, attributes that are set on a panel can be inherited by controls that are contained in that panel.
- Providing information to the ASP.NET page framework about which controls to keep together when paginating. By default, the contents of a panel are kept together on a page. You can modify this behavior by setting the **Paginate** property for the panel.

You can include literal text together with its accompanying markup tags in the text contents of the **Panel**.

Example

The following code example demonstrates how to set properties for a panel during page load and how to define functions to manipulate properties for a panel so that they respond to command clicks. At page load, the code also looks for and modifies a label within a device-specific content template in a second panel.

```
C#
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Drawing" %>

<script runat="server">
public void Page_Load(Object sender, EventArgs e)
{
    // Set Panel1 properties
    Panel1.Wrapping = Wrapping.NoWrap;
    Panel1.Alignment = Alignment.Center;
    Panel1.StyleReference = "title";
    // Find Label in Panel2
    Control ctl = Panel2.Content.FindControl("lblStatusToday");
    if (ctl != null)
        ((System.Web.UI.MobileControls.Label)ctl).Text = "I found this label!";
}

public void MakeFontRed(Object sender, EventArgs e)
{
    Panel1.ForeColor = Color.Red;
}

public void MakeFontBlue(Object sender, EventArgs e)
{
    Panel1.ForeColor = Color.Blue;
}
</script>

<html><body>
<mobile:Form runat="server" id="Form1">

<!-- First Panel -->
<mobile:Panel runat="server" id="Panel1">
    <mobile:TextView runat="server" id="TextView1"> A Panel provides a grouping mechanism<br />
    for organizing controls.
    </mobile:TextView>
</mobile:Panel>
```

```
<mobile:Command runat="server" id="Command1" BreakAfter="false" Text="Make Font Red" OnClick="MakeFontRed"/>
<mobile:Command runat="server" id="Command2" BreakAfter="true" Text="Make Font Blue" OnClick="MakeFontBlue"/>

<!-- Second Panel -->
<mobile:Panel ID="Panel2" Runat="server">
    <mobile:DeviceSpecific id="DeviceSpecific1" runat="server">
        <!-- Filter and template for HTML32 devices -->
        <Choice Filter="isHTML32" Xmlns="http://schemas.microsoft.com/mobile/html32template">
            <ContentTemplate>
                <mobile:Label id="Label1" runat="server"> HTML32 Template</mobile:Label>
                <mobile:Label ID="lblStatusToday" Runat="server"/>
            </ContentTemplate>
        </Choice>
        <!-- Default filter and template -->
        <Choice>
            <ContentTemplate>
                <mobile:Label ID="Label1" Runat="server"> Default Template</mobile:Label>
                <mobile:Label ID="lblStatusToday" Runat="server" />
            </ContentTemplate>
        </Choice>
    </mobile:DeviceSpecific>
</mobile:Panel>
</mobile:Form>
</body></html>
```

You will also need to add this section to your Web.config file:

PhoneCall Class

Provides control capability to render a command that the user can select to dial the specified phone number.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class PhoneCall **Inherits** TextControl **Implements** IPostBackEventHandler

C#

public class PhoneCall : TextControl, IPostBackEventHandler

For devices with telephony capability, such as a cell phone, the **PhoneCall** control renders a command that the user can select to dial the specified phone number.

On devices without telephony capability, the **PhoneCall** control renders alternate text, possibly including the phone number or a link; it is not treated as a command that has dialing capability.

RangeValidator Class

Validates that another control's value falls within an allowable range.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class RangeValidator **Inherits** BaseValidator

C#

public class RangeValidator : BaseValidator

The minimum and maximum values of the range are provided either directly or by reference to another control. This class behaves identically to the **System.Web.UI.WebControls.RangeValidator** control.

By default, **Command** controls on a form raise validator controls on the form to perform validation when the form is submitted to the server. To disable automatic validation, set the **CausesValidation** property on the **Command** to **false**.

RegularExpressionValidator Class

Provides control capability to validate that another control's value matches a provided regular expression.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class RegularExpressionValidator **Inherits** BaseValidator

C#

public class RegularExpressionValidator : BaseValidator

This class behaves identically to the ASP.NET System.Web.UI.WebControls.RegularExpressionValidator control.

By default, Command controls on a form raise validator controls on the form to perform validation when the form is submitted to the server. To disable automatic validation, set the CausesValidation property on the Command to false.

Example

The following example shows how you can add regular expression properties, such as the ValidationExpression and Text properties, programmatically during a page load.

Security Note:

This example has a text box that accepts user input, which is a potential security threat. By default, ASP.NET Web pages validate that user input does not include script or HTML elements.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
    Private Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
        Const uNameExpr As String = "^[a-zA-Z]{1,9}$"
        Const phoneExpr As String = "((\d{3}) ?)(\d{3}-)\d{3}-\d{4}"

        ' Define validation expressions.
        RegExprVal1.ValidationExpression = uNameExpr
        RegExprVal2.ValidationExpression = phoneExpr

        ReqFldVal1.Text = "User name is required"
        RegExprVal1.Text = "Must be between 2 to 10 characters"
        RegExprVal2.Text = "Please provide a valid number: (425) 555-0187"

        ' ErrorMessages appear in ValidationSummary.
        RegExprVal1.ErrorMessage = "Incorrect UserName format. Name & " can be 2 to 10 characters long"
```

```
ReqFldVal1.ErrorMessage = "User name required"
RegExprVal2.ErrorMessage = "Please provide a valid number: (000) 000-0000"
End Sub

Private Sub OnCmdClick(ByVal sender As Object, ByVal e As EventArgs)
    If (Page.IsValid) Then
        ActiveForm = Form2
    End If
End Sub
</script>

C#
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
    void Page_Load(Object sender, EventArgs e)
    {
        string uNameExpr = "^([a-zA-Z])(.{1,9})$";
        string phoneExpr = "((\\(\\d{3}\\) ?)|(\\d{3}-))\\d{3}-\\d{4}";

        // Define validation expressions.
        RegExprVal1.ValidationExpression = uNameExpr;
        RegExprVal2.ValidationExpression = phoneExpr;

        ReqFldVal1.Text = "User name is required";
        RegExprVal1.Text = "Must be between 2 to 10 characters";
        RegExprVal2.Text = "Please provide a valid number: (425) 555-0187";

        // ErrorMessages appear in ValidationSummary.
        RegExprVal1.ErrorMessage = "Incorrect UserName format. Name" + " can be 2 to 10 characters long";
        ReqFldVal1.ErrorMessage = "User name required";
        RegExprVal2.ErrorMessage = "Please provide a valid number: (000) 000-0000";
    }

    void OnCmdClick(Object sender, EventArgs e)
    {
        if (Page.IsValid)
        {
            ActiveForm = Form2;
        }
    }
</script>

<html>
<body>

<mobile:Form runat="server" id="Form1" >
    <mobile:Label runat="server" id="Label1" Text="Your information" StyleReference="title" />
    <mobile:Label runat="server" id="Label2" Text="User Name (required)" />
    <mobile:Textbox runat="server" id="TextBox1"/>
    <mobile:RequiredFieldValidator runat="server" id="ReqFldVal1" ControlToValidate="TextBox1" />
    <mobile:RegularExpressionValidator runat="server" id="RegExprVal1" ControlToValidate="TextBox1" />
    <mobile:Label runat="server" id="Label3" Text="Phone" />
    <mobile:Textbox runat="server" id="TextBox2"/>
    <mobile:RegularExpressionValidator runat="server" id="RegExprVal2" ControlToValidate="TextBox2" />
    <mobile:ValidationSummary ID="ValidationSummary1" FormToValidate="Form1" HeaderText="Error Summary:"
        runat="server" />
    <mobile:Command runat="server" id="Command1" Text="Submit" OnClick="OnCmdClick"/>
</mobile:Form>

<mobile:Form id="Form2" runat="server" >
    <mobile:Label ID="Label4" runat="server" Text="Thank You." />
</mobile:Form>

</body>
</html>
```

RequiredFieldValidator Class

Provides control capability to validate whether the value of the associated input control is different from its initial value.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class RequiredFieldValidator **Inherits** BaseValidator

C#

public class RequiredFieldValidator : BaseValidator

This class behaves identically to the ASP.NET **System.Web.UI.WebControls.RequiredFieldValidator** control.

By default, **Command** controls on a form raise validator controls on the form to perform validation when the form is submitted to the server. To disable automatic validation, set the **CausesValidation** property on the **Command** to **false**.

Example

The following code example requires a user to enter a number from 1 to 23. It uses both the **RangeValidator** and the **RequiredFieldValidator** to validate user entries.

Security Note:

This example has a text box that accepts user input, which is a potential security threat. By default, ASP.NET Web pages validate that user input does not include script or HTML elements.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
Private Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Const uNameExpr As String = "^[a-zA-Z]{1,9}$"
    Const phoneExpr As String = "((\d{3}) ?)(\d{3}-)?\d{3}-\d{4}"

    ' Define validation expressions.
    RegExprVal1.ValidationExpression = uNameExpr
    RegExprVal2.ValidationExpression = phoneExpr
    ReqFldVal1.Text = "User name is required"
    RegExprVal1.Text = "Must be between 2 to 10 characters"
    RegExprVal2.Text = "Please provide a valid number: (425) 555-0187"

    ' ErrorMessages appear in ValidationSummary.
    RegExprVal1.ErrorMessage = "Incorrect UserName format. Name" & " can be 2 to 10 characters long"
    ReqFldVal1.ErrorMessage = "User name required"
    RegExprVal2.ErrorMessage = "Please provide a valid number: (000) 000-0000"
End Sub

Private Sub OnCmdClick(ByVal sender As Object, ByVal e As EventArgs)
    If (Page.IsValid) Then
        ActiveForm = Form2
    End If
End Sub
</script>
```

```
C#
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
void Page_Load(Object sender, EventArgs e)
{
    string uNameExpr = "^[a-zA-Z]({1,9})$";
    string phoneExpr = "((\\(\\d{3}\\) ?)(\\d{3}-)?)\\d{3}-\\d{4}";
    // Define validation expressions.
    RegExprVal1.ValidationExpression = uNameExpr;
    RegExprVal2.ValidationExpression = phoneExpr;
    ReqFldVal1.Text = "User name is required";
    RegExprVal1.Text = "Must be between 2 to 10 characters";
    RegExprVal2.Text = "Please provide a valid number: (425) 555-0187";
    // ErrorMessages appear in ValidationSummary.
    RegExprVal1.ErrorMessage = "Incorrect UserName format. Name" + " can be 2 to 10 characters long";
    ReqFldVal1.ErrorMessage = "User name required";
    RegExprVal2.ErrorMessage = "Please provide a valid number: (000) 000-0000";
}

void OnCmdClick(Object sender, EventArgs e)
{
    if (Page.IsValid)
    { ActiveForm = Form2; }
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml"><body>
<mobile:Form runat="server" id="Form1" >
    <mobile:Label runat="server" id="Label1" Text="Your information" StyleReference="title" />
    <mobile:Label runat="server" id="Label2" Text="User Name (required)" />
    <mobile:Textbox runat="server" id="TextBox1" />
    <mobile:RequiredFieldValidator runat="server" id="ReqFldVal1" ControlToValidate="TextBox1" />
    <mobile:RegularExpressionValidator runat="server" id="RegExprVal1" ControlToValidate="TextBox1" />
    <mobile:Label runat="server" id="Label3" Text="Phone" />
    <mobile:Textbox runat="server" id="TextBox2" />
    <mobile:RegularExpressionValidator runat="server" id="RegExprVal2" ControlToValidate="TextBox2" />
    <mobile:ValidationSummary ID="ValidationSummary1" FormToValidate="Form1" HeaderText="Error
    Summary:" runat="server" />
    <mobile:Command runat="server" id="Command1" Text="Submit" OnClick="OnCmdClick"/>
</mobile:Form>
<mobile:Form id="Form2" runat="server" >
    <mobile:Label ID="Label4" runat="server" Text="Thank You." />
</mobile:Form>
</body></html>
```

SelectionList Class

Provides several different visual representations for a list of selectable items.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

```
<ValidationPropertyAttribute("SelectionList")>
Public Class SelectionList Inherits MobileControl Implements IPostBackDataHandler
```

C#

```
[ValidationPropertyAttribute("SelectionList")]
public class SelectionList : MobileControl, IPostBackDataHandler
```

The **SelectionList** class maintains the selection of single or multiple selected items. The **SelectionList** is derived directly from the **MobileControl** class and does not have any of the pagination handling properties, such as the **ItemWeight** property.

Example

In the following code example, the **DataSource** property of the **SelectionList** class is an array of values that is created during the initial page load. You can change the setting of the **SelectType** property to see different versions of a **SelectionList**.

Note:

The following code sample uses the single-file code model and may not work correctly if copied directly into a code-behind file. This code sample must be copied into an empty text file that has an .aspx extension.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
Public Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        Label1.Text = "Select an item"

        ' Create and fill an array list.
        Dim listValues As New ArrayList()
        listValues.Add("One")
        listValues.Add("Two")
        listValues.Add("Three")

        ' Bind the array to the list.
        SelList1.DataSource = listValues
        SelList1.DataBind()

        ' Set the SelectType.
        SelList1.SelectType = ListSelectType.Radio
    Else
        If (SelList1.SelectedIndex > -1) Then
            ' To show the selection, use the Selection property.
            Label1.Text = "Your selection is " & SelList1.Selection.Text

            ' Or, show the selection by using ' the MobileListItemCollection class.
            ' Get the index of the selected item
            Dim idx As Integer = SelList1.SelectedIndex
            Label2.Text = "You have selected " & SelList1.Items(idx).Text

            ' Insert a copy of the selected item
            Dim mi As MobileListItem = SelList1.Selection
            Label3.Text = "The index of your selection is " & mi.Index.ToString()
            SelList1.Items.Insert(idx, New MobileListItem(mi.Text + " Copy"))
        Else
            Label1.Text = "No items selected"
        End If
    End If
End Sub
</script>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
public void Page_Load(Object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        Label1.Text = "Select an item";

        // Create and fill an array list.
        ArrayList listValues = new ArrayList();
```

```
listValues.Add("One");
listValues.Add("Two");
listValues.Add("Three");

// Bind the array to the list.
SelList1.DataSource = listValues;
SelList1.DataBind();

// Set the SelectType.
SelList1.SelectType = System.Web.UI.MobileControls.ListSelectType.Radio;
}
else
{
    if (SelList1.SelectedIndex > -1)
    {
        // To show the selection, use the Selection property.
        Label1.Text = "Your selection is " + SelList1.Selection;

        // Or, show the selection by using the MobileListItemCollection class.
        // Get the index of the selected item
        int idx = SelList1.SelectedIndex;
        Label2.Text = "You have selected " + SelList1.Items[idx].Text;

        // Insert a copy of the selected item
        MobileListItem mi = SelList1.Selection;
        Label3.Text = "The index of your selection is " + mi.Index.ToString();
        SelList1.Items.Insert(idx, new MobileListItem(mi.Text + " Copy"));
    }
    else
    {
        Label1.Text = "No items selected";
    }
}
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<mobile:form id="form1" runat="server">
    <mobile:Label id="Label1" runat="server" Text="Show a list" />
    <mobile:Label id="Label2" runat="server" />
    <mobile:Label id="Label3" runat="server" />
    <mobile:SelectionList runat="server" id="SelList1" />
    <mobile:Command id="Command1" runat="server" Text=" OK " />
</mobile:form>
</body>
</html>
```

StyleSheet Class

Organizes styles that will be applied to other controls.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class StyleSheet **Inherits** MobileControl

C#

public class StyleSheet : MobileControl

A **StyleSheet** control can contain any number of style objects, or more specialized style objects that inherit from the **Style** class. These should have unique name properties. You can then refer to other controls on the same page by their **Name** property. This class has no visual representation.

A page can also use an external style sheet, and several pages can share the same external style sheet.

 Note:

The **StyleSheet** control ignores its own style attributes; setting a style attribute on the **StyleSheet** itself has no effect on the styles contained as children within the **StyleSheet** control.

Example

The following example shows how you can add Style properties to a **StyleSheet** control during the **Page_Load** event.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.UI.MobileControls" %>
<%@ Import Namespace="System.Drawing" %>

<script runat="server">
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        StyleSheet1("Style1").ForeColor = Color.Red
        StyleSheet1("Style1").Font.Size = System.Web.UI.MobileControls.FontSize.Large
        StyleSheet1("Style1").Font.Bold = BooleanOption.True
        StyleSheet1("Style1").Font.Italic = BooleanOption.True
        StyleSheet1("Style2").ForeColor = Color.Blue
        StyleSheet1("Style2").Font.Size = System.Web.UI.MobileControls.FontSize.Normal
        StyleSheet1("Style2").Font.Bold = BooleanOption.False
        StyleSheet1("Style2").Font.Italic = BooleanOption.True
        StyleSheet1("Style3").ForeColor = Color.Green
        StyleSheet1("Style3").Font.Size = System.Web.UI.MobileControls.FontSize.Small
        StyleSheet1("Style3").Font.Bold = BooleanOption.False
        StyleSheet1("Style3").Font.Italic = BooleanOption.False
    End If
End Sub

Private Sub SelectStyle(ByVal sender As Object, ByVal e As EventArgs)
    ' Retrieve the style name as a string.
    Dim myStyle As String = SelectionList1.Selection.ToString()

    ' Match the style name and apply the style to TextView1.
    Select Case myStyle
        Case "hot"           TextView1.StyleReference = "Style1"
        Case "medium"        TextView1.StyleReference = "Style2"
        Case "mild"          TextView1.StyleReference = "Style3"
    End Select
End Sub
</script>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Web.UI.MobileControls" %>
<%@ Import Namespace="System.Drawing" %>

<script runat="server">
protected void Page_Load(Object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        StyleSheet1["Style1"].ForeColor = Color.Red;
        StyleSheet1["Style1"].Font.Size = System.Web.UI.MobileControls.FontSize.Large;
        StyleSheet1["Style1"].Font.Bold = BooleanOption.True;
        StyleSheet1["Style1"].Font.Italic = BooleanOption.True;
        StyleSheet1["Style2"].ForeColor = Color.Blue;
        StyleSheet1["Style2"].Font.Size = System.Web.UI.MobileControls.FontSize.Normal;
        StyleSheet1["Style2"].Font.Bold = BooleanOption.False;
        StyleSheet1["Style2"].Font.Italic = BooleanOption.True;
    }
}
```

```
        StyleSheet1["Style3"].ForeColor = Color.Green;
        StyleSheet1["Style3"].Font.Size = System.Web.UI.MobileControls.FontSize.Small;
        StyleSheet1["Style3"].Font.Bold = BooleanOption.False;
        StyleSheet1["Style3"].Font.Italic = BooleanOption.False;
    }
}

void SelectStyle(object sender, EventArgs e)
{
    // Retrieve the style name as a string.
    String myStyle = SelectionList1.Selection.ToString();

    // Match the style name and apply the style to TextView1.
    switch (myStyle)
    {
        case "hot": TextView1.StyleReference = "Style1";
                    break;
        case "medium": TextView1.StyleReference = "Style2";
                    break;
        case "mild": TextView1.StyleReference = "Style3";
                    break;
    }
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml" ><body>
<mobile:StyleSheet id="StyleSheet1" runat="server">
    <mobile:Style Name="Style1" Font-Name="Arial" BackColor="#E0E0E0" Wrapping="Wrap"> </mobile:Style>
    <mobile:Style Name="Style2" Font-Name="Arial" BackColor="blue" Wrapping="NoWrap"> </mobile:Style>
    <mobile:Style Name="Style3" Font-Name="Arial Narrow" BackColor="Green" Wrapping="NoWrap">
    </mobile:Style>
</mobile:StyleSheet>

<mobile:Form id="Form1" runat="server">
    <mobile:Label id="Label1" runat="server" Text="Today's Special" StyleReference="title" />
    <mobile:TextView id="TextView1" runat="server" StyleReference="Style1">Chili </mobile:TextView>
    <mobile:SelectionList runat="server" id="SelectionList1">
        <item Text="hot" Value="hot"/>
        <item Text="medium" Value="medium"/>
        <item Text="mild" Value="mild"/>
    </mobile:SelectionList>
    <mobile:Command ID="Command1" runat="server" Text="Select Style" OnClick="SelectStyle" />
</mobile:Form>
</body></html>
```

TextBox Class

Provides a text-based control that allows the user to enter text.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

```
<ValidationPropertyAttribute("Text")>
Public Class TextBox Inherits TextControl Implements IPostBackDataHandler
```

C#

```
[ValidationPropertyAttribute("Text")]
public class TextBox : TextControl, IPostBackDataHandler
```

The **TextBox** allows only single-line text input. This control also implements the **IPostBackDataHandler** interface; it can accept input and generate postbacks. With devices using WML, however, entered data might not post back to the server. When the user enters values in a text box, the values are stored in the **Text** property, which is inherited from the **TextControl** base class.

 Note:

A **TextBox** accepts user input, which is a potential security threat. By default, ASP.NET Mobile Web Forms pages validate that user input does not include script or HTML elements.

TextView Class

Provides a programmable control for displaying larger amounts of text, with optional markup tags on a mobile page.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class TextView **Inherits** PagedControl

C#

public class TextView : PagedControl

Unlike literal text in a mobile form, you can set content of a **TextView** from code. Unlike a mobile **Label** control, you can include markup within the value of the **Text** property. Also, the **TextView** control is careful not to break up inserted elements such as links during pagination.

If the **TextView** control is within a **Panel**, set the **Panel.Paginate** property to **true** to allow text to flow across multiple pages on mobile devices. If the **TextView** control is not in a **Panel**, put the name of the **TextView** control into the **Form.ControlToPaginate** property.

Most other properties and methods of the mobile **TextView** control are of more use to developers of new device adapters for rendering the contents on specific devices.

 Caution:

This control can be used to display user input, which might include malicious client script. Check any information that is sent from a client for executable script, SQL statements, or other code before displaying it in your application. You can use validation controls to verify user input before displaying the input text in a control. ASP.NET provides an input request validation feature to block script and HTML in user input.

Example

The following code example creates numbers and identifies each prime number in a sentence, and adds those sentences to the **TextView** control. The example also uses custom pagination to provide only a page full of text at a time.

 Note:

The following code sample uses the single-file code model and may not work correctly if copied directly into a code-behind file. This code sample must be copied into an empty text file that has an .aspx extension.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
Protected Sub Page_Load(ByVal byvalsender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        ' Add items to the list
        SelectList1.Items.Add(New MobileListItem("Verify transactions", "Done"))
    End If

```

```
SelectionList1.Items.Add(New MobileListItem("Check balance sheet", "Scheduled"))
SelectionList1.Items.Add(New MobileListItem("Call customer", "Done"))
SelectionList1.Items.Add(New MobileListItem("Send checks", "Pending"))
SelectionList1.Items.Add(New MobileListItem("Send report", "Pending"))
SelectionList1.Items.Add(New MobileListItem("Attend meeting", "Scheduled"))

' Show all items in list
SelectionList1.Rows = SelectionList1.Items.Count

End If
End Sub

Private Sub TextChanged(ByVal sender As Object, ByVal e As EventArgs)
    ' Called duringPostBack if changed
    Dim task As String = TextBox1.Text
    Dim status As String = TextBox2.Text
    If (task.Length > 0 AndAlso status.Length > 0) Then
        Dim li As New MobileListItem(task, status)

        ' Remove the item if it exists
        If (SelectionList1.Items.Contains(li)) Then
            SelectionList1.Items.Remove(li)
        Else
            ' Add the item if it does not exist
            SelectionList1.Items.Add(li)
        End If

        ' Clear the text boxes
        TextBox1.Text = String.Empty
        TextBox2.Text = String.Empty
    End If

    ' Display all items.
    SelectionList1.Rows = SelectionList1.Items.Count
End Sub
</script>

C#
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
// Returns an array of Boolean values
private bool[] TestPrimes(int from, int howMany)
{
    // Test a range of numbers to determine which are prime.
    bool[] isPrime = new bool[howMany];
    int endAt = from + howMany - 1;
    for (int i = from; i < endAt; i++)
    {
        // Set a default value of true
        isPrime[i - from] = true;
        int sqrt = (int)Math.Sqrt(i);
        for (int factor = 2; factor <= sqrt; factor++)
        {
            if ((i % factor) == 0)
            {
                // Set value as false
                isPrime[i - from] = false;
                break;
            }
        }
    }
    return isPrime;
}

protected void Page_Load(object sender, EventArgs args)
{
    if (!IsPostBack)
    {

```

```
        Primes.ItemCount = 2000;
        Primes.ItemsPerPage = 20;
        form1.ControlToPaginate = Primes;
    }
}

protected void Primes_OnLoadItems(object sender, LoadItemsEventArgs args)
{
    StringBuilder StrBldr = new StringBuilder();
    Primes.Text = "";
    // Start the list at 2.
    int startNumber = args.ItemIndex + 2;
    bool[] isPrime;
    isPrime = TestPrimes(startNumber, args.ItemCount);
    for (int i = 0; i < args.ItemCount; i++)
    {
        string message;
        if (isPrime[i])
            message = String.Format("<b>{0} is prime</b>", i + startNumber);

        else
            message = String.Format("<b>{0}</b> is not prime", i + startNumber);
        StrBldr.Append(message);
        StrBldr.Append("<br />");
    }
    Primes.Text = StrBldr.ToString();
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml"><body>
<mobile:Form id="Form1" runat="server">
    <mobile:Label Id="Label1" runat="server"> Create a new Task with Status</mobile:Label>
    <mobile:SelectionList runat="server" SelectType="ListBox" id="SelectionList1" />
    <mobile:Label Id="Label2" runat="server" Text="Enter the Task name" />
    <mobile:TextBox runat="server" id="TextBox1" OnTextChanged="TextChanged" />
    <mobile:Label Id="Label3" runat="server" Text="Enter the Task status" />
    <mobile:TextBox runat="server" id="TextBox2" />
    <mobile:Command ID="Command1" runat="server" Text="Submit" />
</mobile:Form>
</body></html>
```

ValidationSummary Class

Presents a summary of all the validation errors that have occurred on a form.

Namespace: System.Web.UI.MobileControls

Assembly: System.Web.Mobile (in system.web.mobile.dll)

Visual Basic

Public Class ValidationSummary **Inherits** MobileControl

C#

public class ValidationSummary : MobileControl

The **ValidationSummary** class creates a summary of all validation errors and presents them either inline or on a separate form. The **ValidationSummary** uses the text in the **ErrorMessage** property for the errors that are displayed either inline or on a summary form.

Although the **ValidationSummary** class of the ASP.NET mobile controls mimics the behavior of the Web Forms **ValidationSummary** class in many ways, the mobile controls version of the class does not inherit directly from the Web Forms version of the class. Thus, properties that modify the output of the validation summary, such as the **DisplayMode** property, are not available in mobile controls. The mobile controls version of the summary is derived directly from the **MobileControl** class.

Example

The following code example demonstrates how to create an instance of a **ValidationSummary** class, and add it to a form in an ASP.NET mobile Web application during a page load. The user-defined **Page_Load** event handler determines if there is an error, and then either launches the form containing the **ValidationSummary**, or the congratulatory thank you form.

Security Note:

This example has a text box that accepts user input, which is a potential security threat. By default, ASP.NET Web pages validate that user input does not include script or HTML elements.

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
Private Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)

    ' Define validation expressions
    UserNameExprValidator.ValidationExpression = "[a-zA-Z](.){1,9}$"
    PhoneExprValidator.ValidationExpression = "((\d{3}) ?)(\d{3}-)?\d{3}-\d{4}"
    UserNameReqValidator.Text = "User name required"
    UserNameExprValidator.Text = "Must be 2-10 characters"
    PhoneExprValidator.Text = "Requires a valid number: 425-555-0187"

    ' ErrorMessages appear in ValidationSummary.
    UserNameExprValidator.ErrorMessage = "User name must be 2-10 characters"
    UserNameReqValidator.ErrorMessage = "User name required"
    PhoneExprValidator.ErrorMessage = "Valid number required: 425-555-0187"

End Sub

Private Sub OnCmdClick(ByVal sender As Object, ByVal e As EventArgs)
    If Page.IsValid Then
        ActiveForm = Form2
    Else
        ValSummary.BackLabel = "Return to Form"
        ActiveForm = Form3
    End If
End Sub
</script>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
private void Page_Load(Object sender, EventArgs e)
{
    // Define validation expressions.
    UserNameExprValidator.ValidationExpression = "[a-zA-Z](.){1,9}$";
    PhoneExprValidator.ValidationExpression = "((\d{3}) ?)(\d{3}-)?\d{3}-\d{4}";
    UserNameReqValidator.Text = "User name required";
    UserNameExprValidator.Text = "Must be 2-10 characters";
    PhoneExprValidator.Text = "Requires a valid number: 425-555-0187";

    // ErrorMessages appear in ValidationSummary.
    UserNameExprValidator.ErrorMessage = "User name must be 2-10 characters";
    UserNameReqValidator.ErrorMessage = "User name required";
    PhoneExprValidator.ErrorMessage = "Valid number required: 425-555-0187";
}

private void OnCmdClick(Object sender, EventArgs e)
{
    if (Page.IsValid)
```

```
        ActiveForm = Form2;
    else
    {
        ValSummary.BackLabel = "Return to Form";
        ActiveForm = Form3;
    }
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml"><body>
<mobile:Form runat="server" id="Form1">
    <mobile:Label runat="server" id="HeadingLabel" Text="Provide ur name & number" StyleReference="title"/>
    <mobile:Label runat="server" id="UserNameLabel" Text="User Name (req'd)" />
    <mobile:Textbox runat="server" id="UserNameTextBox"/>
    <mobile:RequiredFieldValidator runat="server" id="UserNameReqValidator"
        ControlToValidate="UserNameTextBox" />
    <mobile:RegularExpressionValidator runat="server" id="UserNameExprValidator"
        ControlToValidate="UserNameTextBox" />
    <mobile:Label runat="server" id="PhoneLabel" Text="Phone" />
    <mobile:Textbox runat="server" id="PhoneTextBox"/>
    <mobile:RegularExpressionValidator runat="server" id="PhoneExprValidator"
        ControlToValidate="PhoneTextBox" />
    <mobile:Command runat="server" id="Cmd1" text="Submit" OnClick="OnCmdClick"/>
</mobile:Form>

<mobile:Form runat="server" id="Form2" >
    <mobile:Label ID="Label1" runat="server" Text="Thank You." />
</mobile:Form>

<mobile:Form ID="Form3" Runat="server">
    <mobile:ValidationSummary ID="ValSummary" FormToValidate="Form1" HeaderText="Error Summary:"
        runat="server" />
</mobile:Form>
</body></html>
```

Understanding State Management

ASP.NET Mobile Web pages provide a number of services for state management. This section describes how ASP.NET mobile controls can use support session or application view state, as well as private view state.

Controlling View State

Microsoft ASP.NET Web pages are capable of maintaining their own state across round trips. When a property is set for a control, ASP.NET saves the property value as part of the control's state. To the application, this makes it appear that the page's lifetime spans multiple client requests. This page-level state is known as the page's view state.

On ordinary ASP.NET Web pages, view state is sent by the server as a hidden field in a form as part of the response to the client, and is returned to the server by the client as part of a postback. However, to reduce bandwidth demand when using mobile controls, ASP.NET does not send a page's view state to the client. Instead, view state is saved as part of a user's session on the server. Where there is a view state, ASP.NET sends a hidden field that identifies this page's view state as part of every response to the client, and the hidden field is returned to the server by the client as part of the next postback.

Managing View State History

Because view state for a given page must be kept on the server, if the user clicks the browser's **Back** button, it is possible for the current state on the server to be out of synchronization with the current page of the browser. For example, suppose the user goes to Page 1, then clicks a button to go to Page 2, then presses **Back** to return to Page 1. The current page on the browser is now Page 1, but the current state on the server is that of Page 2.

To address this issue, ASP.NET mobile Web pages maintain a history of view state information in the user's session. Each identifier sent to the client corresponds to a position in this history. In the previous example, if the user again posts from Page 1, the mobile Web page uses the identifier saved with Page 1 to synchronize view state history.

You can configure the size of this history to fine-tune it for the 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 example:

```
<configuration>
  <system.web> <mobileControls sessionStateHistorySize="10" /> </system.web>
</configuration>
```

Managing Expired Sessions

Because view state is saved in the user's session, it is possible for view state to expire if a page is not posted back within the session expiration time. This expiration is unique to mobile Web pages. When the user posts a page for which there is no view state available, the **OnViewStateExpire** method of the page is called. The default implementation of this method throws an exception indicating that view state has expired. However, if your application is able to restore view state manually after expiration, you can override this method at the page level and choose not to call the base implementation.

Enabling and Disabling View State

The advantage of using the session to manage view state is that it results in a smaller response size. The disadvantage is that inefficient use of session state can lead to poorer performance. If you use controls that contain large amounts of data, you can improve efficiency with techniques such as custom paging or disabling view state. For example, consider a site that displays news stories. Instead of saving article content for each user session, the site can use smarter data access so that only one copy of each article is cached on the server, and session state usage is minimized.

To disable view state for a control and its children, set the **EnableViewState** property of the control to **false**. To disable view state for an entire page, set the **EnableViewState** attribute of the **@ Page** directive to **false**.

Even when view state is disabled, some mobile controls save essential state information across client round trips. An example of such information includes the currently active form on a page. When you turn off view state, the page saves this essential information as a hidden form variable that is sent on a round trip to the client.

Managing Cookies and Client State

By default, the session management features of ASP.NET require the server to write a session cookie to a client. The client subsequently submits the cookie on each request during the session, and the server looks up session state information using the cookie information. However, many mobile devices do not support cookies. For session management (including view state), to work correctly on these devices, you must configure the application to use cookieless session management. With this feature enabled, ASP.NET automatically inserts the session key in application URLs.

Some devices do not support cookies. To persist long-term client state, an application can use client-specific information, such as a customer number entered by the user. Because you cannot rely on a client having cookies, your application must take you to an alternate page that can be bookmarked. The following sample shows an example. Users that browse to this URL view a form where they enter their customer IDs. The application then displays an alternate URL, which users can bookmark.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" EnableViewState="false" %>
<script runat="server" language="c#">

protected void Page_Load(Object sender, EventArgs e)
{
    String customerID = Request.QueryString["cid"];
    if (customerID != null)
    {
        // A customer ID was found. Simulate a lookup by converting the client ID back to a user.
        int underscore = customerID.IndexOf('_');
        if (underscore != -1)
        {
            // If visiting the first time, prompt the user to bookmark.
            if (Session["FirstTime"] != null)
            {
                Session["FirstTime"] = null;
                WelcomeLabel.Text = String.Format("Welcome, {0}", customerID.Substring(0, underscore));
                ActiveForm = WelcomeForm;
            }
            else
            {
                ReturnLabel.Text = String.Format("Welcome back, {0}", customerID.Substring(0, underscore));
                ActiveForm = ReturnForm;
            }
        }
    }
}
```

```
    }  
    }  
}  
}  
  
protected void LoginForm_OnSubmit(Object sender, EventArgs e)  
{  
    // Generate a customer ID. Normally, you would create a new customer profile.  
    String customerID = CustomerName.Text + "_" + System.Guid.NewGuid().ToString();  
    String path = AbsoluteFilePath + "?cid=" + Server.UrlEncode(customerID);  
    Session["FirstTime"] = true;  
    RedirectToMobilePage(path);  
}  
</script>  
  
<mobile:Form runat="server">  
    <mobile:Label runat="server" StyleReference="title"> Welcome to the site. Please register to continue.  
</mobile:Label>  
    <mobile:TextBox runat="server" id="CustomerName" />  
    <mobile:Command runat="server" OnClick="LoginForm_OnSubmit" Text="Register" />  
</mobile:Form>  
  
<mobile:Form id="WelcomeForm" runat="server">  
    <mobile:Label runat="server" id="WelcomeLabel" /> Please bookmark this page for future access.  
</mobile:Form>  
  
<mobile:Form id="ReturnForm" runat="server">  
    <mobile:Label runat="server" id="ReturnLabel" />  
</mobile:Form>
```

Optimizing View State for Mobile Applications

For mobile Web pages, the following considerations are important:

- Saving view state to session state is highly optimized. If there is no view state to save, nothing is stored in the session, and no identifier is sent to the client. However, application developers who want to avoid using session management, or who want pages capable of high throughput, can consider reducing or eliminating the use of view state. In many application cases (such as rendering a page of formatted text), view state is unnecessary and is best turned off.
- In addition to application view state, a mobile Web page must store other types of state information about the page. This information might include the active form or pagination information about the form. Such information is always sent to the client rather than kept on the server, and is usually generated in an optimized way. For example, if the first form is active, or the first page of a form is being shown, this information is not saved, because these are default states. Such state information is called the private view state. All controls can override the **LoadPrivateViewState** and **SavePrivateViewState** methods to read and write private view state.

Note

Good security practice demands that if you include sensitive information in session state, you use a connection with HTTPS and SSL/TLS authentication.

Controlling Session State

Microsoft ASP.NET provides a **HttpSessionState** object that you can use to save information about a user session across multiple requests. The ASP.NET session management feature is scalable and robust, and you can use it across Web farms.

Considerations for Cookieless Sessions

By default, the ASP.NET session object uses a client-side cookie to store an identifier. This identifier is used to locate the session across server round trips. However, ASP.NET also supports a cookieless session mode that initially redirects the client to a new URL containing the session ID, and then automatically parses the session ID out of the URL.

For ASP.NET mobile Web pages that use session state, you must consider the following factors:

- Some mobile devices and gateways do not support cookies. In those cases, the Web server must have session management set to cookieless mode.
- Some mobile devices have problems handling relative URLs after they have been redirected by cookieless session management.

Using Alternatives to Cookies

Some mobile devices do not support cookies, so you must find alternative techniques for scenarios that require persistent state. For example, if a user logs on to a page, the application could assign a logon ID that is used for the remainder of the session. Typically, you use a cookie for this form of authentication, which is called cookie-based authentication. However, cookie-based authentication is not an option for devices that do not support cookies. Instead, you must rely on another state-management mechanism.

Session State

One alternative is to use session state. Because you can configure session state to work without cookies, the user can keep the ID in session state. However, a disadvantage of this technique is that the information expires with the session. Another disadvantage is that this technique always requires session state be used.

Hidden Variables

ASP.NET mobile Web pages do not include a mobile control for writing out hidden variables. Instead, the form provides a collection called **HiddenVariables** within the **MobilePage** class. 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.

This alternative uses hidden variables on a page. Hidden variables are automatically resubmitted as part of a form submission, whether the submission is to the same page or to another page.

Additional State Management Considerations

You cannot store state information in either static variables or member variables between requests. Static variables are shared across all requests to that page, so you never know in which request the value of the variable was set. Member variables lose their value, because the page is discarded after every response and rebuilt for every request.

Designing Secure Mobile Web Applications

Building secure Web sites is always a top priority. There are additional security considerations when building a mobile Web application that might communicate sensitive information over public data networks.

Authentication, authorization, and encryption are the three items you must consider for security in your Web applications. Authentication establishes the identity of a user. Authorization helps to control what the user can or cannot access. Encryption is the mechanism that helps to protect data as it passes between client and server.

ASP.NET mobile controls use the security infrastructure that is in place with Internet Information Services (IIS) and the Microsoft .NET Framework.

Authentication Options for Mobile Devices

This section on designing secure applications describes the additional complexities of authenticating mobile devices.

The following topics are also discussed in this section:

- Windows Authentication
- Passport Authentication
- Forms Authentication
- Authentication on devices that do not support cookies

Windows Authentication

Internet Information Services (IIS), in conjunction with ASP.NET, provides the functionality for negotiating Microsoft Windows-based authentication with the client. In an unsecured application, i.e. one using Anonymous authentication, the specific identity of the requesting user is never considered. Instead, the request is executed by using default accounts that were set up during the IIS installation. Using Internet Services Manager, you can choose from the available Windows authentication models, including Basic, Digest, and Integrated Windows. You can configure these using the Internet Services Manager snap-in for Microsoft Management Console. IIS negotiates the credentials based on which authentication methods the browser supports, and which authentication methods are enabled for the application.

Basic Authentication

Many mobile devices on the market today support only the Basic form of authentication. Basic authentication is the most widely supported credential exchange mechanism but, by itself, is not secure because there is no encryption.

Caution: Using Basic authentication transmits user names and passwords in clear text. If you do not use Secure Sockets Layer protocol (SSL) to protect the application, the data is subject to outside observation. Using HTTPS is strongly recommended for those cases.

Passport Authentication

The ASP.NET does not support using Passport in conjunction with mobile devices.

Forms Authentication

Forms authentication is a part of the .NET Framework architecture that provides for authenticating users without utilizing the IIS authentication primitives. The general sequence of events is:

1. Client requests a page.
2. Browser is redirected to the logon form specified in the configuration if the user is not already authenticated.
3. Client provides credentials in a form posted back to the server.
4. Server validates the credentials, writes the client cookie, and redirects back to the original page.
5. Authentication cookie is checked and the original page is served.

Step 4 creates a problem for some devices that do not support cookies. The

MobileFormsAuthentication.RedirectFromLogin page writes the authentication information into the query string. To keep the user from being redirected to the logon page for every request, the authentication information can be presented in each request as part of the query string. ASP.NET provides a method for carrying data such as this in the query string for relative URLs.

Caution: Form authentication sends the username and password in clear text by default. It is therefore recommended that you use HTTPS for this and other sensitive information, and specify **<form requireSSL=true />** in the Web.config file. If a device does not support SSL directly or through the gateway and it tries to access a page that requires SSL, an error message will be displayed to the user.

Authentication on Devices that do not Support Cookies

Browser authentication models typically use cookies for tracking the authentication of the user. Many mobile devices do not support cookies and are therefore unable to authenticate by this means.

Providing Basic authentication primitives for the widest array of devices possible adds value for the developer targeting mobile devices.

Cookieless authentication requires that an authentication ticket be accessed in another location. In forms authentication, if the cookie is not present, the ASP.NET authentication module checks for it in the query string. You implement this by including the session ID in the query string. This requires rewriting all links on a page to include the authentication ticket conditionally based on these factors:

- Is the application configured to persist cookieless data? This is determined by the setting of the **CookielessDataDictionary** property of the **IPageAdapter** interface.
- Does this device require cookieless authentication?

Note You cannot configure an individual device for cookieless authentication.

The **CookielessDataDictionaryType** attribute in the **<mobileControls>** section must be set in order for authentication to work properly on cookieless devices, as well as devices that have **SupportsCookieWithRedirect** set to **false**. By default, the **CookielessDataDictionaryType** attribute is set to **System.Web.Mobile.CookielessData** in the machine.config file. To override this behavior for a single application, you must set the **CookielessDataDictionaryType** attribute to an empty string ("").

Be aware that some devices and browsers currently require fully qualified URLs in response to an HTTP redirect. Set **useFullyQualifiedRedirectUrl=true** in the System.Web section of either the Machine.config file or Web.config file (at the application level).

Security and WAP Gateways

A Wireless Application Protocol (WAP) gateway serves as an intermediary, decrypting the user's SSL connection and re-encrypting the information to send it to the mobile device.

Note: For maintaining protection for the data transfer channel, WAP relies on a protocol called WTLS (Wireless Transport Layer Security).

In a browser environment, when you connect to a site using SSL/TLS, your browser automatically verifies that the domain part of the URL matches the domain in the X.509 certificate that the HTTPS server presents when you connect to it.

SSL certificates are tamper evident because the cryptographic signature is checked against the root certificates of the major certificate authorities. This check assures that the requesting party is connected to the right host and helps protect you from attack from an intermediary.

Note: Many WAP gateways do not perform this check or, if they do, do not pass information about mismatches back to the user.

Wireless carriers help provide some security between the wireless device and the base station and across the physical network connecting base stations and switching centers. But a carrier's security measures end with the network and do not provide end-to-end, cross-platform security for any wireless device. For example, WAP Internet access introduces a point of potential vulnerability where the Wireless Transport Layer Security (WTLS) (which helps to maintain a restricted connection between the mobile device and the WAP gateway) changes to a SSL connection between the WAP gateway and the Web server. Some corporations are moving to enterprise control of their gateways as a means of assuring that the gateways are trusted.

Port Usage for Mobile Applications

When setting up a firewall, enable the following ports:

Port Number	Keyword
80	http
443	https
42424	ASP.NET State Server (if used)

Accessing Data Using Listing Controls

Mobile Web pages provide three mobile controls, **List**, **SelectionList**, and **ObjectList**, that you can use to display data from a database.

Creating an Object for a List from a Collection Element

The following steps describe how a mobile **List** control constructs a **MobileListItem** instance from an element in the collection:

1. The control checks whether the **DataTextField** or **DataValueField** properties are defined. If they are, the control uses these field names to discover properties from the element and set the **Text** and **Value** properties of the **MobileListItem** instance.
2. If neither the **DataTextField** nor **DataValueField** properties are defined, the control sets the **Text** and **Value** properties of the **MobileListItem** instance to the string representation of the element (by using the **ToString** method).
3. If an **ItemDataBind** event handler is defined, the handler is called. You can use this handler to set properties of the **MobileListItem** instance.

When providing default rendering, the list control represents a **MobileListItem** instance by its **Text** property. In templated rendering, the template can render a desired property of the **MobileListItem** instance or associated data-bound object.

When the **ItemsAsLinks** property is set, the **List** control renders items as hyperlinks. The value of the **Text** property becomes the link text, and the value of the **Value** property becomes the target URL.

Handling the Selection

If your list is complex, such as an array of objects, you cannot directly access the members of the selected item through the selection. However, if you design your application appropriately, you can access the associated object. If you are creating a group of objects to populate your list, you can make the group a global array, and then handle the returned value as an index into the array, as shown in the following code sample.

```
Visual Basic
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<script runat="server">
Private customers(3) As Person
Private Class Person
    Private _Name, _Nickname, _Initials As String
    Public Sub New(ByVal name As String, _ ByVal nickname As String, ByVal initials As String)
        Me._Name = name
        Me._Nickname = nickname
        Me._Initials = initials
    End Sub
    Public ReadOnly Property Name() As String
        Get
            Return _Name
        End Get
    End Property
    Public ReadOnly Property Nickname() As String
        Get
            Return _Nickname
        End Get
    End Property
End Class
End Class
```

```
Public ReadOnly Property Initials() As String
    Get
        Return _Initials
    End Get
End Property
End Class
Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    ReDim customers(2)
    customers(0) = New Person("George Washington", "George", "GW")
    customers(1) = New Person("Abraham Lincoln", "Abe", "AL")
    customers(2) = New Person("Theodore Roosevelt", "Teddy", "TR")
    If (Not IsPostBack) Then ' Bind the array to the list.
        List1.DataSource = customers List1.DataTextField = "Name" List1.DataBind()
    End If
End Sub
Protected Sub List1_ItemCommand(ByVal sender As Object, ByVal e As ListCommandEventArgs)
    Dim selectedPerson As Person = customers(e.ListItem.Index)
    Label1.Text = String.Format("{0} (AKA {1}), initials {2}", selectedPerson.Name, selectedPerson.Nickname, _
        selectedPerson.Initials)
    ActiveForm = Form2
End Sub
Protected Sub Command1_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.ActiveForm = Me.Form1
End Sub
</script>

<html ><body>
<mobile:form id="Form1" runat="server">
    <mobile:List ID="List1" Runat="server" OnItemCommand="List1_ItemCommand"> </mobile:List>
</mobile:form>
<mobile:Form ID="Form2" Runat="server">
    <mobile:Label ID="Label1" runat="server">Label</mobile:Label>
    <mobile:Command ID="Command1" Runat="server" OnClick="Command1_Click">Return to
        Form1</mobile:Command>
</mobile:Form>
</body></html>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
private Person[] customers = new Person[3];
private class Person
{
    private String _Name, _Nickname, _Initials;
    public Person(String name, String nickname, String initials)
    {
        this._Name = name;
        this._Nickname = nickname;
        this._Initials = initials;
    }
    public String Name
    {
        get { return _Name; }
    }
    public String Nickname
    {
        get { return _Nickname; }
    }
    public String Initials
    {
        get { return _Initials; }
    }
}
private void Page_Load(object sender, System.EventArgs e)
{
    customers[0] = new Person("George Washington", "George", "GW");
    customers[1] = new Person("Abraham Lincoln", "Abe", "AL");
```

```
customers[2] = new Person("Theodore Roosevelt", "Teddy", "TR");
if(!IsPostBack)
{
    // Bind the array to the list.
    List1.DataSource = customers;
    List1.DataTextField = "Name";
    List1.DataBind();
}
}
private void List1_ItemCommand(object sender, ListCommandEventArgs e)
{
    Person selectedPerson = customers[e.ListItem.Index];
    Label1.Text = String.Format("{0} (AKA {1}), initials {2}", selectedPerson.Name, selectedPerson.Nickname,
    selectedPerson.Initials);
    ActiveForm = Form2;
}
protected void Command1_Click(object sender, EventArgs e)
{
    this.ActiveForm = this.Form1;
}
</script>

<html ><body>
<mobile:form id="Form1" runat="server">
    <mobile:List ID="List1" Runat="server" OnItemCommand="List1_ItemCommand"> </mobile:List>
</mobile:form>
<mobile:Form ID="Form2" Runat="server">
    <mobile:Label ID="Label1" runat="server">Label</mobile:Label>
    <mobile:Command ID="Command1" Runat="server" OnClick="Command1_Click">Return to
    Form1</mobile:Command>
</mobile:Form>
</body></html>
```

Data Binding a List or SelectionList Control

The **List** and **SelectionList** ASP.NET mobile controls render a basic view of data and enable users to select data items.

You can bind a **List** or **SelectionList** mobile control to a **DataView** or a **DataSet** object, or to any object that implements **IEnumerable** or **IListSource**. To bind a **List** or **SelectionList** mobile control to a **DataView** object, you set the control's **DataSource** property and call its **DataBind** method. The following code example demonstrates how to bind a control to a **DataSet** object that contains a table named **Titles**.

```
C#
myList.DataSource = ds.Tables["Titles"].DefaultView; myList.DataBind();
```

Alternatively, you can bind a **List** or **SelectionList** control to a **DataSet** object. To do so, you set the **DataMember** property to the name of the table. The following example is equivalent to the preceding one.

```
C#
myList.DataSource = ds; myList.DataMember = "Titles"; myList.DataBind();
```

A list item in a **List** or **SelectionList** control can bind to two data values. One data value is bound to the list item's **Text** property, and the other is bound to its **Value** property. You configure binding for list items by setting the **DataTextField** (System.Web.UI.MobileControls.SelectionList.DataTextField) and **DataValueField** (System.Web.UI.MobileControls.SelectionList.DataValueField) properties of the **List** or **SelectionList** control. The **List** control displays each item using its **Text** property. For example, if you want to display each item by its **CustomerName** property, set the **DataTextField** property to **CustomerName**.

You might want to display each item as a summary composed of several data values. To do this, you can handle the **ItemDataBind** event of the **List** control or the **ItemDataBind** event of the **SelectionList** control, and set the **Text** property programmatically. The following code example demonstrates how to render book information as a combination of title and price.

```
C#
private void List_OnItemDataBind(object sender, ListDataBindEventArgs e)
{
    e.ListItem.Text = String.Format ("{0} – {1}", DataBinder.Eval (e.DataItem, "title"), DataBinder.Eval (e.DataItem,
    "price", "{0:C}"));
}
```

On devices that support richer rendering, you can use a template set for your **List** control to show a customized view of a data item. In templated mode, the **List** control functions like the **Repeater** ASP.NET server control. For example, you can use the following item template to show a detailed view of a book.

```
<ItemTemplate>
<tr style="background-color:FFECD8">
    <td><%=# DataBinder.Eval(((MobileListItem)Container).DataItem, _ "title") %></td>
    <td><%=# DataBinder.Eval(((MobileListItem)Container).DataItem, _ "title_id") %></td>
    <td><%=# DataBinder.Eval(((MobileListItem)Container).DataItem, _ "type") %></td>
    <td><%=# DataBinder.Eval(((MobileListItem)Container).DataItem, _ "pub_id") %></td>
    <td><%=# DataBinder.Eval(((MobileListItem)Container).DataItem, _ "price", "{0}", "{0:C}") %></td>
</tr>
</ ItemTemplate>
```

Data Binding and Viewing Data Using an ObjectList Control

You can use the **ObjectList** ASP.NET mobile control to provide a versatile view of data. This control shows two views of your data source, a list view that shows a summary of each item, and a view that shows item details. You can explicitly define the list fields that you want to show for each item, or you can automatically generate them from the data source. By default, the control generates a field for each field in the data source, in the order it appears within the data source. The name of each automatically generated field is used as the field title.

You can bind data in an **ObjectList** control to a **DataView** or a **DataSet** object. To bind data in an **ObjectList** mobile control to a **DataView** object, set the **DataSource** property, and then call the **DataBind** method to perform data binding. For example, if you have a **DataSet** object with a table named **Titles**, you can use the following statements to perform data binding:

```
C#
myObjectList.DataSource = ds.Tables["Titles"].DefaultView;
myObjectList.DataBind();
```

Alternatively, to bind data directly to a **DataSet** object, you must additionally set the **DataMember** property to the name of the table. The following example is equivalent to the previous one:

```
C#
myObjectList.DataSource = ds;
myObjectList.DataMember = "Titles";
myObjectList.DataBind();
```

You can also set an item field to a value composed of several of the data item's properties. To do this, you can override the **ItemDataBind** event of the **ObjectList** control, and set the field programmatically. The following example sets the Summary field to a combination of the title and price of a book:

```
C#
private void ObjectList_OnItemDataBind(Object sender, ObjectListDataBindEventArgs e)
{
    e.ListItem["Summary"] = String.Format( String.Format ("{0} – {1}", DataBinder.Eval(e.DataItem, "title"),
    DataBinder.Eval (e.DataItem, "price")));
}
```

In addition, you can control how each item is rendered in the list view. By default, the list view uses the first field to represent each item. However, you can set the **LabelField** property to any defined or automatically generated field, including one that is not visible in the details view. Using the previous example, you can use the Summary field as the label for an item, while hiding it in the details view.

Performing Data Binding Within an ObjectList

An **ObjectList** control shows content only if it is bound to a data source. The data source can be any object that implements the **IEnumerable** interface or the **ICollection** interface. However, each object in the data source must be of the same class, or must inherit from the same common class. When the **AutoGenerateFields** is set to **true**, the objects in the data source must be of the same class.

For each object in the data source, the **ObjectList** control constructs an **ObjectListItem** instance and stores it in its **Items** collection. This collection can be inspected, but not modified, by your code.

All properties that the object list refers to must be public properties of a class common to all objects in the data source. All properties that fields refer to must also be of a bindable type. Valid bindable types are **String**, **DateTime**, **Decimal**, and the set of primitive types.

For each object in the data source, the control performs the following data binding steps:

1. For each field, the **ObjectList** control uses the field's **DataField** property to determine which property of the data object to look up. Each value is saved in the **ObjectListItem** instance as an indexed field value.
2. After all fields are bound in this way, the control calls any **ItemDataBind** event handler that is defined for the **ObjectList** control. You can use this handler to do more complex data binding and to set values in the **ObjectListItem** instance.

Note

Some changes to an **ObjectList** control require you to rebind the data on that control. These changes include adding or removing fields, changing the **DataField** property of a field, and changing the **DataFormatString** property of a field.

Initiating Automatic Field Generation during Data Binding

At data binding, if the **AutoGenerateFields** property is set to **true**, the **ObjectList** control inspects the data source and then automatically generates fields. If the data source is a list of type **ITypedList**, the **ObjectList** control inspects the information for the type. Otherwise, the **ObjectList** control inspects the type information of the first object in the list.

For each bindable public property of the inspected type, the **ObjectList** control generates a field bound to the property. This occurs during data binding; if you make changes to a field, or add or remove fields, you must bind the properties again.

By default, automatically generated fields are visible, they use default formatting, and they have the same title as the property name. All of these values can be changed programmatically. In addition, you can specify the title of the field by adding an **ObjectListTitleAttribute** attribute to the property. For example, if the object has a property declared as [ObjectListTitle("Address")]myAddress, the generated field will have the title "Address."

If the **ObjectList** control has explicitly defined fields, auto-generated fields are added after those fields. The following example shows how to display a list of custom objects in an **ObjectList** control.

```
Visual Basic
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
Private customers(3) As Person
Private Class Person
    Private _Name, _Nickname, _Initials As String
    Public Sub New(ByVal name As String, ByVal nickname As String, ByVal initials As String)
        Me._Name = name
        Me._Nickname = nickname
        Me._Initials = initials
    End Sub
    Public ReadOnly Property Name() As String
        Get
            Return _Name
        End Get
    End Property
    Public ReadOnly Property Nickname() As String
        Get
            Return _Nickname
        End Get
    End Property
    Public ReadOnly Property Initials() As String
        Get
            Return _Initials
        End Get
    End Property
End Class
Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    ReDim customers(2)
    customers(0) = New Person("George Washington", "George", "GW")
    customers(1) = New Person("Abraham Lincoln", "Abe", "AL")
    customers(2) = New Person("Theodore Roosevelt", "Teddy", "TR")
    If (Not IsPostBack) Then ' Bind the array to the list.
        List1.DataSource = customers List1.DataTextField = "Name"
        List1.DataBind()
    End If
End Sub
Protected Sub List1_ItemCommand(ByVal sender As Object, ByVal e As ListCommandEventArgs)
    'Show the Summary text
    Dim selectedPerson As Person = customers(e.ListItem.Index)
    Label1.Text = _String.Format("{0} (AKA {1}), initials {2}", selectedPerson.Name, selectedPerson.Nickname,
        selectedPerson.Initials)
    ActiveForm = Form2
End Sub
Protected Sub Command1_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.ActiveForm = Me.Form1
End Sub
</script>
```

```
<html><body>
<mobile:form id="Form1" runat="server">
    <mobile:List ID="List1" Runat="server" OnItemCommand="List1_ItemCommand" />
</mobile:form>
<mobile:Form ID="Form2" Runat="server">
    <mobile:Label ID="Label1" runat="server" />
    <mobile:Command ID="Command1" Runat="server" StyleReference="subcommand"
        OnClick="Command1_Click"> Return</mobile:Command>
</mobile:Form>
</body></html>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Collections" %>
<script runat="server">
private ArrayList customers = new ArrayList();
private class Person
{
    private String _Name, _Nickname, _Initials;
    public Person(String name, String nickname, String initials)
    {
        this._Name = name;
        this._Nickname = nickname;
        this._Initials = initials;
    }
    public String Name { get { return _Name; } }
    public String Nickname { get { return _Nickname; } }
    public String Initials { get { return _Initials; } }
}
private void Page_Load(object sender, System.EventArgs e)
{
    customers.Add( new Person("George Washington", "George", "GW"));
    customers.Add( new Person("Abraham Lincoln", "Abe", "AL"));
    customers.Add( new Person("Theodore Roosevelt", "Teddy", "TR"));
    if(!IsPostBack)
    {
        // Bind the array to the list.
        List1.DataSource = customers;
        List1.DataTextField = "Name";
        List1.DataBind();
    }
}
private void List1_ItemCommand(object sender, ListCommandEventArgs e)
{
    Person selectedPerson = (Person)customers[e.ListItem.Index];
    Label1.Text = String.Format("{0} (AKA {1}), initials {2}", selectedPerson.Name, selectedPerson.Nickname,
        selectedPerson.Initials);
    ActiveForm = Form2;
}
protected void Command1_Click(object sender, EventArgs e)
{
    this.ActiveForm = this.Form1;
}
</script>

<html ><body>
<mobile:form id="Form1" runat="server">
    <mobile:List ID="List1" Runat="server" OnItemCommand="List1_ItemCommand"> </mobile:List>
</mobile:form>
<mobile:Form ID="Form2" Runat="server">
    <mobile:Label ID="Label1" runat="server" />
    <mobile:Command ID="Command1" Runat="server" OnClick="Command1_Click">Return</mobile:Command>
</mobile:Form>
</body></html>
```

Associating Commands within an Object List

The **ObjectList** control allows a set of commands to be associated with an item. Each command has a **Name** property used to uniquely identify the command, and a **Text** property used to render the command.

The control provides two ways to define commands:

- Declaratively, using **<Command>** child elements within an **ObjectList** control.
- Programmatically, by constructing **ObjectListCommand** objects and adding them to the **Commands** collection of the control.

By default, all items in the list share the same set of commands. However, before rendering the set of commands for a given item, the control raises the **ShowItemCommands** event. An event handler can use this method to modify the set of visible commands for the item.

When the user selects a command, an **ItemCommand** event is raised, with information about the selected item and the name of the selected command.

Even if you define a default command for an item, you must include a command by the same name in the **Commands** collection. If the control cannot render a UI that includes a shortcut for the default command, it must display the default command as part of the set of commands.

Accessing Field Values of a List Item

When you associate an event handler with an **ObjectList** control, it renders list items as interactive elements. Clicking an item in a list generates an event that retrieves the appropriate action for that item. During data binding, each field is bound to its corresponding property.

To retrieve a field value from an **ObjectListItem** object, use the following syntax, where **lstItem** is an **ObjectListItem** instance:

```
lstItem[fieldName]
```

The following example is similar to the previous example, but uses two **Command** controls for each record, and uses the field syntax to retrieve field values:

Visual Basic

```
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Collections" %>
<script runat="server">

Public Class Person
    'Private Fields
    Private _Name, _Nickname, _Initials, _Wife As String
    'Constructor
    Public Sub New(ByVal name As String, ByVal nickname As String, ByVal initials As String, ByVal wife As String)
        Me._Name = name
        Me._Nickname = nickname
        Me._Initials = initials
        Me._Wife = wife
    End Sub
    'Public Properties
    Public ReadOnly Property Name()
        Get
            Return _Name
        End Get
    End Property
End Class
```

```
Public ReadOnly Property Nickname()
    Get
        Return _Nickname
    End Get
End Property
Public ReadOnly Property Initials()
    Get
        Return _Initials
    End Get
End Property
Public ReadOnly Property Wife()
    Get
        Return _Wife
    End Get
End Property
End Class

Private Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        'An ArrayList for the Person objects
        Dim customers As New ArrayList()
        customers.Add( _ New Person("George Washington", "George", "GW", "Martha"))
        customers.Add( _ New Person("Abraham Lincoln", "Abe", "AL", "Mary"))
        customers.Add( _ New Person("Theodore Roosevelt", "Teddy", "TR", "Alice Lee"))
        'Bind the array to the list.
        ObjectList1.DataSource = customers
        ObjectList1.LabelField = "Name"
        ObjectList1.DataBind()
    End If
End Sub

Protected Sub ObjectList1_ItemCommand( ByVal sender As Object, ByVal e As ObjectListCommandEventArgs)
    If e.CommandName = "ShowSummary" Then
        'Show the Summary text
        Label1.Text = String.Format("{0}, AKA: '{1}', initials: '{2}'", e.ListItem("Name"),
        e.ListItem("Nickname"), e.ListItem("Initials"))
    ElseIf e.CommandName = "MoreInfo" Then
        'Show the More Info text
        Label1.Text = String.Format("{0}'s wife was {1}", e.ListItem("Nickname"), e.ListItem("Wife"))
    End If
    Me.ActiveForm = Form2
End Sub

Protected Sub Command1_Click(ByVal sender As Object, ByVal e As EventArgs)
    'Show the first form
    Me.ActiveForm = Form1
End Sub
</script>

<html><body>
<mobile:form id="Form1" runat="server">
    <mobile:ObjectList ID="ObjectList1" Runat="server" CommandStyle-StyleReference="subcommand"
    LabelStyle-StyleReference="title" OnItemCommand="ObjectList1_ItemCommand">
        <Command Name="ShowSummary" Text="Summary" /> <Command Name="MoreInfo" Text="More Info" />
    </mobile:ObjectList>
</mobile:form>
<mobile:Form ID="Form2" Runat="server">
    <mobile:Label ID="Label1" runat="server" />
    <mobile:Command ID="Command1" Runat="server" OnClick="Command1_Click">Return </mobile:Command>
</mobile:Form>
</body></html>

C#

<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<%@ Import Namespace="System.Collections" %>

<script runat="server">
```

```
private class Person
{
    // Private Fields
    private String _Name, _Nickname, _Initials, _Wife;
    // Constructor
    public Person(string name, string nickname, string initials, string wife)
    {
        this._Name = name;
        this._Nickname = nickname;
        this._Initials = initials;
        this._Wife = wife;
    }
    // Public Properties
    public String Name { get { return _Name; } }
    public String Nickname { get { return _Nickname; } }
    public String Initials { get { return _Initials; } }
    public String Wife { get { return _Wife; } }
}

private void Page_Load(object sender, System.EventArgs e)
{
    if(!IsPostBack)
    {
        // An ArrayList for the Person objects
        ArrayList customers = new ArrayList();
        // Fill the Person object
        customers.Add( new Person("George Washington", "George", "GW", "Martha"));
        customers.Add( new Person("Abraham Lincoln", "Abe", "AL", "Mary"));
        customers.Add( new Person("Theodore Roosevelt", "Teddy", "TR", "Alice Lee"));
        // Bind the array to the list.
        ObjectList1.DataSource = customers;
        ObjectList1.LabelField = "Name";
        ObjectList1.DataBind();
    }
}

protected void ObjectList1_ItemCommand (object sender, ObjectListCommandEventArgs e)
{
    if (e.CommandName == "ShowSummary")
    {
        // Show the Summary text
        Label1.Text = String.Format("{0}, AKA: '{1}', initials: '{2}'", e.ListItem["Name"], e.ListItem["Nickname"], e.ListItem["Initials"]);
    }
    else if (e.CommandName == "MoreInfo")
    {
        // Show the More Info text
        Label1.Text = String.Format("{0}'s wife was {1}", e.ListItem["Nickname"], e.ListItem["Wife"]);
    }
    this.ActiveForm = Form2;
}

protected void Command1_Click(object sender, EventArgs e)
{
    this.ActiveForm = Form1;
}
</script>

<html><body>
<mobile:form id="Form1" runat="server">
    <mobile:ObjectList ID="ObjectList1" Runat="server" CommandStyle-StyleReference="subcommand"
        LabelStyle-StyleReference="title" OnItemCommand="ObjectList1_ItemCommand">
        <Command Name="ShowSummary" Text="Summary" />
        <Command Name="MoreInfo" Text="More Info" />
    </mobile:ObjectList>
</mobile:form>
<mobile:Form ID="Form2" Runat="server">
    <mobile:Label ID="Label1" runat="server" />
    <mobile:Command ID="Command1" Runat="server" OnClick="Command1_Click">Return </mobile:Command>
</mobile:Form>
</body></html>
```

Data Binding in ObjectList Templates

In the **ObjectList** control, you can define templates to customize the user's experience. If you are doing inline data binding in templates, use syntax such as the following:

```
<%#((ObjectListItem)Container)["BookName"]%>
```

You can also use the **System.Web.UI.DataBinder.Eval(System.Object,System.String)** method to bind data in all templates, as in the following example:

```
<%#DataBinder.Eval(((ObjectListItem)Container).DataItem,"fieldname")%>
```

Differences between ObjectList and List Controls

The **ObjectList** ASP.NET mobile control differs from the **List** control in the ways listed in the following table.

Features	ObjectList control	List control
Data-bound	The only way to add items to an object list is to bind the object list to a data source.	Supports adding items statically, programmatically, and through data binding.
Multiple property view	Allows you to view multiple properties, or fields, of each item. Depending on device characteristics, you can render the control as a table that displays more than one property of each object. Alternatively, you can provide UI to allow the user to view additional properties of an object.	Displays one property of each item.
Multiple item commands	Allows you to associate multiple commands with each item. The set of commands for an item can be either shared among all items or unique to the item.	Supports a default command for each item.
Custom pagination and templating	Supported.	Supported.

Note

The table compares the features of the **ObjectList** and **List** controls. However, although both controls support custom pagination and templating, the **SelectionList** control does not support pagination.

Specifying Field Elements within an Object List

Using an **ObjectList** control, you can display multiple fields for each item. Each field is associated with a property name. When a **List** item is bound to a data object, each field is bound to the corresponding property of the object. There are three ways to define a field:

- Declaratively, using the **<Field>** element within an object list.
- Programmatically, by instantiating **ObjectListField** objects and adding them to the **Fields** collection of the control.
- Automatically, by setting the **AutoGenerateFields** property to **true**.

Differences between the SelectionList and List Classes

Although the **SelectionList** and **List** controls are similar, there are fundamental differences in functionality at design time and run time. Both classes maintain a collection of list items. However, whereas the **List**

control is derived from the **PagedControl** and ultimately the **MobileControl** class, the **SelectionList** control is derived directly from the **MobileControl** class and does not have pagination handling properties, such as the **ItemWeight** property.

The major difference between the classes is that the **SelectionList** class supports single or multiple selected items. The **SelectType** property contains the **ListSelectType** enumerated value, which determines whether the **SelectionList** is in single or multiple selection mode.

With the **List** class, you can choose only single items in a list. In contrast, the **SelectionList** class enables you to specify a variety of list types, including **CheckBox**, **DropDown**, **ListBox**, **MultiSelectListBox**, and **Radio**.

SelectionList Functionality

A **SelectionList** control is in single selection mode when you set the **SelectType** property to any of the following:

- DropDown
- ListBox
- Radio

Handling the Selection

To retrieve the current selected item in a **SelectionList** control while in single selection mode, use the **SelectedIndex** and **Selection** properties.

The **CheckBox** and **MultiSelectListBox** enumerated values indicate multiselect mode. To retrieve the selection, query each item's **Selected** property.

The following example shows how to retrieve the selected values from a multiselect list.

```
Visual Basic
<%@ Page Language="VB" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat="server">
Private Class Person
    'Private Fields
    Private _Name, _Nickname As String
    'Constructor
    Public Sub New(ByVal name As String, ByVal nickname As String)
        Me._Name = name
        Me._Nickname = nickname
    End Sub
    'Public Properties
    Public ReadOnly Property Name()
        Get
            Return _Name
        End Get
    End Property
    Public ReadOnly Property Nickname()
        Get
            Return _Nickname
        End Get
    End Property
End Class

'An ArrayList for the Person objects
```

```
Dim presidents = New ArrayList()

Private Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    'Fill the presidents ArrayList
    presidents.Add( New Person("George Washington", "George"))
    presidents.Add( New Person("Abraham Lincoln", "Abe"))
    presidents.Add( New Person("Theodore Roosevelt", "Teddy"))
    If Not IsPostBack Then
        'Bind the array to the list.
        SelectionList1.DataSource = presidents
        'Specify the field to display
        SelectionList1.DataValueField = "Name"
        SelectionList1.DataBind()
    End If
End Sub

Protected Sub Command1_Click( ByVal sender As Object, ByVal e As EventArgs)
    Dim retval As String = String.Empty
    Dim per As Person
    If Not SelectionList1.IsMultiSelect Then
        retval = "Value: "
        'Get the selected item
        per = CType(presidents(SelectionList1.SelectedIndex), Person)
        'Get the text of the item
        If Not IsNothing(per) Then
            retval &= per.Name & "(" & per.Nickname & ")"
        End If
    ElseIf SelectionList1.IsMultiSelect Then
        retval = "Values: "
        'Gather the text from list items
        Dim li As MobileListItem
        Dim i As Integer = 0
        For i = 0 To SelectionList1.Items.Count - 1
            li = SelectionList1.Items(i)
            'Gather text only from selected items
            If li.Selected Then
                per = CType(presidents(li.Index), Person)
                retval &= per.Name & "(" & per.Nickname & "),"
            End If
        Next
    End If
    'Clean ending comma, if any
    If retval.IndexOf(", ") > -1 Then
        retval = retval.Substring(0, retval.Length - 2)
    End If
    'Put return value into the Label
    Label1.Text = retval
    'Activate Form2
    Me.ActiveForm = Form2
End Sub

Protected Sub Command2_Click( ByVal sender As Object, ByVal e As EventArgs)
    'Activate Form1
    Me.ActiveForm = Form1
End Sub
</script>

<html><body>
<mobile:form id="Form1" runat="server">
    Select several items in the list:<br />
    <mobile:SelectionList ID="SelectionList1" Runat="server" SelectType="Checkbox"></mobile:SelectionList>
    <mobile:Command ID="Command1" Runat="server" OnClick="Command1_Click"> Record Choices
    </mobile:Command>
</mobile:form>
<mobile:Form ID="Form2" Runat="server">
    <mobile:Label ID="Label1" runat="server" />
    <mobile:Command ID="Command2" Runat="server" OnClick="Command2_Click">Return </mobile:Command>
</mobile:Form>
</body></html>
```

C#

```
<%@ Page Language="C#" Inherits="System.Web.UI.MobileControls.MobilePage" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<script runat="server">
```

```
private class Person
{
    // Private Fields
    private String _Name, _Nickname;
    // Constructor
    public Person(string name, string nickname)
    {
        this._Name = name;
        this._Nickname = nickname;
    }
    // Public Properties
    public String Name
    {
        get { return _Name; }
    }
    public String Nickname
    {
        get { return _Nickname; }
    }
}
```

// An ArrayList for the Person objects

ArrayList presidents = new ArrayList();

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Fill the Person object
    presidents.Add( new Person("George Washington", "George"));
    presidents.Add( new Person("Abraham Lincoln", "Abe"));
    presidents.Add( new Person("Theodore Roosevelt", "Teddy"));
    if (!IsPostBack)
    {
        // Bind the array to the list.
        SelectionList1.DataSource = presidents;
        // Specify the field to display
        SelectionList1.DataValueField = "Nickname"; SelectionList1.DataBind();
    }
}
```

```
protected void Command1_Click(object sender, EventArgs e)
{
    string retval = String.Empty;
    Person per;
    if (!SelectionList1.IsMultiSelect)
    {
        retval = "Value: ";
        // Get the selected item
        per = (Person)presidents[SelectionList1.SelectedIndex];
        // Get the name and nickname of the person
        if (per != null)
            retval += per.Name + " (" + per.Nickname + ")";
    }
    else if (SelectionList1.IsMultiSelect)
    {
        retval = "Values: ";
        // Gather the text from list items
        foreach (MobileListItem li in SelectionList1.Items)
        {
            // Gather text only from selected items
            if (li.Selected)
            {
                per = (Person)presidents[li.Index];
                retval += per.Name + " (" + per.Nickname + "), ";
            }
        }
    }
}
```

```
        }  
    }  
    // Clean ending comma, if any  
    if (retval.IndexOf(", ") > -1)  
        retval = retval.Substring(0, retval.Length - 2);  
    // Put return value into the Label  
    Label1.Text = retval;  
    // Activate Form2  
    this.ActiveForm = Form2;  
}  
  
protected void Command2_Click(object sender, EventArgs e)  
{  
    // Activate Form1  
    this.ActiveForm = Form1;  
}  
</script>  
  
<html><body>  
<mobile:form id="Form1" runat="server">  
    Select several items in the list:<br />  
    <mobile:SelectionList ID="SelectionList1" Runat="server" SelectType="Checkbox"> </mobile:SelectionList>  
    <mobile:Command ID="Command1" Runat="server" OnClick="Command1_Click"> Record Choices  
    </mobile:Command>  
</mobile:form>  
<mobile:Form ID="Form2" Runat="server">  
    <mobile:Label ID="Label1" runat="server" />  
    <mobile:Command ID="Command2" Runat="server" OnClick="Command2_Click">Return </mobile:Command>  
</mobile:Form>  
</body></html>
```

Adding Items to a List Control

A **List** control contains a collection of items in the **MobileListItem** class. There are a number of ways that you can add items to a **List** control:

- Create **<Item>** elements inside a list. Each **<Item>** element becomes a **MobileListItem** in the list, and its properties are set from attributes of the **<Item>** element.
- Programmatically add items to the list using the **List** control's **Items** collection. You can construct a **MobileListItem** object and add it to the collection before rendering.
- Bind the **List** control to data, specifically to any object that implements the **IEnumerable** interface or the **IListSource** interface, such as **ArrayList** or **DataSet** objects.

SelectionList Controls and Postbacks

Selecting items in a **SelectionList** ASP.NET mobile control does not generate a response from the server. The form on which the **SelectionList** control appears must be posted back to the server. This is usually accomplished with a **Command** control. When the **Command** control posts the form back to the server, the **SelectionList** control raises a **SelectedIndexChanged** event. Your application can provide a method to handle this event.

Another way to respond to a selection is to add support for devices that can handle client JavaScript (for example, HTML browsers). For these devices, use the following procedure:

1. Add a **Panel** control to your **Form** control.
2. Add a **<DeviceSpecific>** element with a **<Choice>** filter set to "supportsJavaScript".
3. Create a content template inside the choice that contains the ASP.NET **DropDownList** control. This is the non-mobile ASP.NET server control.
4. Set the **DropDownList** control's **AutoPostBack** property to **true**.

You must create a **<DeviceSpecific>** filter with a content template for all other devices that do not support JavaScript and that use the **SelectionList** control.

The following code example demonstrates this technique:

```
<mobile:Panel id="Panel1" runat="server">
<mobile:DeviceSpecific id="DeviceSpecific1" runat="server">

<Choice Filter="supportsJavaScript">
    <ContentTemplate>
        <asp:DropDownList id="DropDownList1" runat="server"
            OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged" AutoPostBack="True">
            <asp:ListItem Value="a">1</asp:ListItem>
            <asp:ListItem Value="b">2</asp:ListItem>
            <asp:ListItem Value="c">3</asp:ListItem>
        </asp:DropDownList>
    </ContentTemplate>
</Choice>
<Choice>
    <ContentTemplate>
        <mobile:SelectionList id="SelectionList1" runat="server"
            OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
            <Item Value="a" Text="1"/>
            <Item Value="a" Text="2"/>
            <Item Value="a" Text="3"/>
        </mobile:SelectionList>
        <mobile:Command runat="server" text="Submit"/>
    </ContentTemplate>
</Choice>

</mobile:DeviceSpecific>
</mobile:Panel>
```

This example requires the following settings in the Web.config file:

```
<configuration>
  <system.web>
    <deviceFilters>
      <filter name="supportsJavaScript" compare="javascript" argument="true"/>
    </deviceFilters>
  </system.web>
</configuration>
```

Handling Request Variations When Posting Across Pages

SelectionList controls can require additional handling in the special case of a cross-page post. For example, consider two pages, Source.aspx and Destination.aspx, where Source.aspx contains a **SelectionList** control and posts to Destination.aspx.

The following example shows the markup for the Source.aspx page:

```
C#

<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>

<mobile:Form runat="server" method="post" action="destination.aspx">
  <mobile:SelectionList runat="server" selectType="MultiSelectListBox" id="MultiSelectList1">
    <item text="i" value="1" />
    <item text="ii" value="2" />
    <item text="iii" value="3" />
  </mobile:SelectionList>
  <mobile:command runat="server" text="Post" />
</mobile:Form>
```

The following example shows the markup and code for the Destination.aspx page:

```
C#

<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>

<script runat=server language=cs>
public void Page_Load()
{
    Label1.Text = Request["MultiSelectList1"];
}
</script>

<mobile:form runat=server>
    <mobile:label id="Label1" runat="server" />
</mobile:form>
```

Suppose that the user browses to Source.aspx, selects the first and third items in the list box, and clicks the command button, which causes the page to be posted to Destination.aspx. The text displayed for Label1 in Destination.aspx varies by markup language and device. These differences are due to the HTML and WML specifications and differences in browser implementations. You might see the following results:

Target	Result	Description
HTML browser	"1, 3"	Delimited by commas and spaces with no trailing delimiter.
WML device 1	"1;3"	Delimited by semicolons with no trailing delimiter.
WML device 2	"1;3;"	Delimited by semicolons with a trailing delimiter.

To more easily use the value of the Request["MultiSelectList1"] variable in the destination page, you can preprocess the data posted from the selection list as shown in the following example. This standardizes the various possibilities to agree with the format used by the HTML browser.

```
C#

public void Page_Load()
{
    String selections = Request["MultiSelectList1"];
    if (selections.Length > 0 && selections [selections.Length - 1] == ';')
    {
        selections = selections.Substring(0, selections.Length - 1);
    }
    Label1.Text = selections.Replace(";", ", ");
}
```

Note

This special handling is unnecessary in the common case of posting back to the same page that contains the **SelectionList** control.

Some cHTML devices require unique names for every check box. In this case, the name generated for the check box is of the form *identifier*item number* for each check box. You can use code similar to the following example when coding cross-page scenarios. In the example, MyPage1.aspx contains the following mobile Web Form and a **SelectionList** control.

```
C#

<mobile:form runat=server action=MyPage2.aspx>
    <mobile:selectionList runat="server" id="mySList ...> ...
</mobile:form>
```

MyPage2.aspx has the following code:

```
C#
<script runat="server">
// Create a Form just for the list selections
System.Collections.Specialized.NameValueCollection _myForm = new NameValueCollection();

public void Page_Init()
{
    // Process the Form
    foreach (String key in Request.Form.Keys)
    {
        // Look for an asterisk in the key
        int pos = key.LastIndexOf("*");
        if (pos > -1)
        {
            // Add the modified key to the Form
            _myForm.Add(key.Substring(0, pos), Request.Form[key])
        }
        else
        {
            // Or add the unmodified key to the Form
            _myForm.Add(key, Request.Form[key]);
        }
    }
}

// Use _myForm in place of Request.Form
public void Page_Load()
{
    // Get the processed list of selected items
    String selectedValues = _myForm["mySList"];
    // etc.
}
</script>
```

SelectionLists and Index Values

As much as possible, the **SelectionList** ASP.NET mobile control emits very concise markup language to the client browser. For the most part, ASP.NET does not send the contents of the item's **Value** property to the client. Instead, it sends a zero-based index number for the item.

For example, suppose a **SelectionList** control contains items with text and value settings listed in the following table.

Item Text	Item Value
Rain	"A rainy string"
Snow	"A snowy string"
Sun	"A sunny string"
Wind	"A windy string"

A portion of the markup that the control renders would resemble the following:

```
<Select Name = " WeatherSelectionList">
    <Option Value = "0">Rain</Option>
    <Option Value = "1">Snow</Option>
    <Option Value = "2">Sun</Option>
    <Option Value = "3">Wind</Option>
</Select>
```

When the user chooses an item in the list and the browser posts the form to the server, the client sends the index number of the selected item. If the user selects **Snow**, the client sends the number 1 to the server.

Because the strings in the items' **Value** properties are not being passed between the client and the server, communication is more efficient. This technique is especially helpful for narrow-bandwidth wireless channels.

It is possible that the client does not post the user's input back to the same page. This occurs when the **Action** property of the **Form** control that contains the **SelectionList** control is set to the URL of another page. In that case, the **SelectionList** control does not try to optimize its output. Instead of sending index numbers to the client, it sends the actual strings contained in each item's **Value** property.

Walkthrough: Creating Web Pages for Mobile Devices

Mobile development in ASP.NET does not differ much from traditional ASP.NET browser-based Web development. ASP.NET exposes a **System.Web.Mobile** namespace devoted specifically to Web development. You can create a Web page from the **MobilePage** base class and add controls from the **System.Web.Mobile** namespace.

Mobile development follows the standard .NET Framework event-driven model in which your application responds to user requests, button clicks, and so on.

In this walkthrough, you will create two Web pages that inherit from the **MobilePage** class and that are designed for a mobile device. The first page will have a mortgage calculator that you can use to determine loan information. The second page displays data in a format that is easy to page through on a small device. Tasks illustrated in this walkthrough include:

- Creating an ASP.NET Web page that displays output on a device such as a mobile phone.
- Adding paging so that users with small devices can move effectively through long lists.
- Testing pages with a device emulator.

Prerequisites

To complete this walkthrough, you will need:

- Ability to run the page on a device such as a mobile phone. Alternatively, you can use one of a number of emulators. In this walkthrough, it is assumed that you have an emulator and that it is available on the same computer as your Web server.

Note

You can run this walkthrough even if you have only a desktop browser. However, an emulator will more directly allow you to see the functionality of the controls you will use in this walkthrough.

- Access to Microsoft Internet Information Services (IIS) and permission to create a new application in IIS. It is recommended that you use a local copy of IIS for the pages in the application, which makes it easiest to test the application with your emulator. However, if you cannot create an application using IIS, you might still be able to use an emulator with the Microsoft Visual Web Developer Web server. For details, see the "Next Steps" section at the end of this walkthrough.

Creating the Web Site

If you have already created a Web site in Visual Web Developer, you can use that Web site and skip to the next section, "Creating the Mortgage Calculator." Otherwise, create a new Web site and page by following these steps.

To create a new local IIS Web site under the IIS root

1. Open Visual Web Developer.
2. On the **File** menu, choose **New**, and then choose **Web Site**. The **New Web Site** dialog box appears.
3. Under **Visual Studio installed templates**, select **ASP.NET Web Site**.

4. Click **Browse**. The **Choose Location** dialog box appears.
5. Click the **Local IIS** tab.
6. Select **Default Web Site**.
7. Click the **Create New Web Application** button. A new application is added under **Default Web Site**.
8. In the box for the new Web site, type **DeviceWalkthrough** and then click **Open**. You are returned to the **New Web Site** dialog box with the **Location** box filled in with **http://localhost/DeviceWalkthrough**.
9. In the **Language** list, select the programming language you prefer to work in. The programming language you choose will be the default for your Web site. However, you can use multiple languages in the same Web application by creating pages and components in different programming languages.
10. Click **OK**. Visual Web Developer creates the new Web site and opens a new page named **Default.aspx**.

Creating the Mortgage Calculator

For the walkthrough, you will create a page that inherits from the **MobilePage** class and that contains a simple mortgage calculator. The calculator prompts the user to enter a loan amount, a loan term in years, and the interest rate. The calculator can determine the monthly payment for that loan.

In this walkthrough, you will use controls from the **System.Web.Mobile** namespace that are specifically designed for devices that cannot display as much information as a desktop browser. Instead, the controls present information in separate views that users can switch between.

To begin, you will delete the **Default.aspx** page and create a mobile page in its place.

To add a mobile page

1. Right-click the **Default.aspx** page in Solution Explorer and choose **Delete**.
2. Click **OK** in the dialog box.
3. Right-click the application in Solution Explorer and choose **Add New Item**.
4. Choose **Mobile Web Form** under **Visual Studio installed templates**.
5. Name the mobile Web page **MobileCalculator.aspx** and then click **Add**. A Web page that inherits from the **MobilePage** class is created and added to your Web site.

Now that you have a mobile page, you will add controls that allow users to enter mortgage information.

To add controls for entering mortgage information

1. In the **MobileCalculator.aspx** page you created in the last procedure, make sure you are in Design view. You will see a **Form** control with the default name of **form1**.
2. From the **Mobile Web Forms** folder of the Toolbox, drag controls onto **form1** and set their properties as noted in the following table.

Control	Property settings
Label	ID = HeadingLabel, Text = Loan Calculator
TextBox	ID = PrincipalText, Text = Principal
TextBox	ID = TermText, Text = Term in Years
TextBox	ID = RateText, Text = Annual Rate
Command	ID = Calculate, Text = Calculate

After you have created the **Form** where users enter their loan information, you will create another **Form** that will show the results.

To create a form to display mortgage calculation results

1. From the **Mobile Web Forms** folder of the Toolbox, drag a **Form** control to the design surface. The **Form** control is assigned the default ID of **Form2**.
2. From the **Mobile Web Forms** folder of the Toolbox, drag controls onto **Form2** and set their properties as noted in the following table.

Control	Property settings
Label	ID = LoanDetailsLabel, Text = Loan Details
Label	ID = PaymentLabel, Text = Payment
Command	ID = ReturnToCalculator, Text = Return to Calculator

You can now create the code that will calculate the loan information and display it.

To calculate the mortgage information and display results

1. If you are using C#, add a reference to the **Microsoft.VisualBasic** namespace so you can use the **Pmt** method to calculate the payment information. Follow these steps:
 - a. In Solution Explorer, right-click the Web site name and choose **Property Pages**.
 - b. Click **Add Reference**.
 - c. In the **.NET** tab, select **Microsoft.VisualBasic.dll** and then click **OK**.
 - d. In the **Property Pages** dialog box, click **OK**.
2. In the **form1** control, double-click the **Calculate** button to create a **Click** event handler, and then add the following code.

```

Visual Basic
Protected Sub Calculate_Click(ByVal sender As Object, ByVal e As EventArgs)
    'Get values from the form
    Dim principal As Double = Convert.ToDouble(PrincipalText.Text)
    Dim apr As Double = Convert.ToDouble(RateText.Text)
    Dim monthlyInterest As Double = apr / (12 * 100)
    Dim termInMonths As Double = Convert.ToDouble(TermText.Text) * 12
    Dim monthlyPayment As Double

    'Calculate the monthly payment
    monthlyPayment = Microsoft.VisualBasic.Financial.Pmt(monthlyInterest, termInMonths, -principal, 0, _
        DueDate.BegOfPeriod)

    'Change to the other form
    
```

```
Me.ActiveForm = Me.form2

'Display the resulting details
Dim detailsSpec As String = "{0} @ {1}% for {2} years"
LoanDetailsLabel.Text = String.Format(detailsSpec, principal.ToString("C0"), apr.ToString(), TermText.Text)
PaymentLabel.Text = "Payment: " & monthlyPayment.ToString("C")
End Sub

C#
protected void Calculate_Click(object sender, EventArgs e)
{
    // Get values from the form
    Double principal = Convert.ToDouble(PrincipalText.Text);
    Double apr = Convert.ToDouble(RateText.Text);
    Double monthlyInterest = (Double)(apr / (12 * 100));
    Double termInMonths = Convert.ToDouble(TermText.Text) * 12;
    Double monthlyPayment;

    // Calculate the monthly payment
    monthlyPayment = Microsoft.VisualBasic.Financial.Pmt( monthlyInterest, termInMonths, -principal, 0,
    Microsoft.VisualBasic.DueDate.BegOfPeriod);

    // Change to the other form
    this.ActiveForm = this.form2;
    // Display the resulting details
    string detailsSpec = "{0} @ {1}% for {2} years";
    LoanDetailsLabel.Text = String.Format(detailsSpec, principal.ToString("C0"), apr.ToString(), TermText.Text);
    PaymentLabel.Text = "Payment: " + monthlyPayment.ToString("C");
}
```

The code gathers the values from the text boxes, converts them to appropriate data types, and then uses them as parameters for the Visual Basic **Pmt** function to calculate the monthly cost of the mortgage. (You can use the Visual Basic function in any language as long as you fully qualify the function call with the namespace.) After calculating the monthly amount, the code switches to the second **Form** control and displays the results in the respective **Label** controls.

3. In the **Form2** control, double-click the **Command** control to create a **Click** event handler, and then add the following highlighted code.

```
Visual Basic
Protected Sub ReturnToCalculator_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.ActiveForm = Me.form1
End Sub

C#
protected void ReturnToCalculator_Click(object sender, EventArgs e)
{
    this.ActiveForm = this.form1;
}
```

Testing the Calculator

You are now ready to test the calculator. You can test the calculator in a desktop browser. However, a more interesting test is to use your device emulator.

To test the calculator

1. Press CTRL+F5 to see your page in the default browser, and to get the exact URL. The first form appears on the page.
2. Start your emulator and connect to the URL for your page.

3. When the page appears in the emulator, enter a loan amount of **100000**, the number of years as **30**, and a percentage rate of **5**, and then click **Calculate**. The calculator is replaced by the results view, with the result **534.59**.

Adding Pagination

Many devices have small display areas, making it impractical to display long lists. ASP.NET provides an **ObjectList** control designed for mobile devices that can automatically display an entire screen of information at one time and provide links so that users can move forward and backward in the list.

In this section of the walkthrough, you will create a data listing that displays more information than can be shown on one screen of even a desktop browser. By adding an **ObjectList** control, you will automatically add paging capability to the output, appropriately sized to the browser the user has.

The first thing you need to do is create a mobile Web Forms page and add an **ObjectList** control to it.

To add a mobile Web Forms page and create an ObjectList control on it

1. Right-click the application in Solution Explorer and choose **Add New Item**.
2. Choose **Mobile Web Form** under **Visual Studio installed templates**.
3. Name the page **MobilePaging.aspx** and then click **Add**. A Web page that inherits from the **MobilePage** class is created and added to your project. The page includes a **Form** control named **form1** on it. You can only use controls in the **System.Web.Mobile** namespace on a page that inherits from the **MobilePage** class.
4. From the **Mobile Web Forms** folder of the Toolbox, drag an **ObjectList** control to the design surface and place it on **form1**. An **ObjectList** control is added to your page. It shows a generic set of data that gives you an idea of what the control will look like when it is rendered on the client.

After the **ObjectList** control is created, you need to create data that will populate the control.

To create the data

1. In the **MobilePaging.aspx** page, switch to Design view and double-click the empty design surface to create an empty event handler for the page **Load** event.
2. In the empty handler, add the following code.

```
Visual Basic
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    'Create and fill an array of strings
    Dim listItems(25) As String
    Dim i As Integer
    For i = 0 To 24
        listItems(i) = String.Format("This is item {0}.", i)
    Next

    'Bind the ObjectList to the Items
    Me.ObjectList1.DataSource = listItems
    Me.ObjectList1.DataBind()
End Sub
```

```
C#
protected void Page_Load(object sender, EventArgs e)
{
    // Create and fill an array of strings
    string[] listItems = new string[25];
    for (int i = 0; i < 25; i++)
        listItems[i] = String.Format("This is item {0}.", i);

    // Bind the ObjectList to the Items
    this.ObjectList1.DataSource = listItems;
    this.ObjectList1.DataBind();
}
```

The code creates an array of string objects and populates it with strings. It then binds that array to the **ObjectList** control.

You can now test the page.

To test the page

1. Press CTRL+F5 to run the page. The page is displayed with a long list of numbered items.
2. Start your device emulator and type in the URL of the page:
(http://localhost/DeviceWalkthrough/paging.aspx).
Notice that the data is displayed in a long list.

Adding Paging

Now that you have a page that displays data, you can add paging so that the display is automatically sized to the size of the screen in the device.

To add paging

1. In the **MobilePaging.aspx** page, switch to Design view and then select **form1**.
2. In the Properties window, set the **Paginate** property to **true**.
3. Select the **ObjectList** control and, in the Properties window, set the **ItemsPerPage** property to **5**.

You can now test paging.

To test paging

1. Press CTRL+F5 to run the page in Internet Explorer. The page is displayed with a page of data and a navigation control.
2. Use the **Next** and **Previous** links to move through the data. In your device emulator, enter the URL of the page. The emulator displays a page of data (five items). If necessary, you can scroll the page up or down.
3. Use the links to move to other pages showing more items.

.Developing .ASP NET .Mobile .Web .Applications
Sudhanshu .Singh