**ASP.NET Controls**

ASP.NET Web server controls are objects on ASP.NET Web pages that run when the page is requested and that render markup to a browser. Many Web server controls resemble familiar HTML elements, such as buttons and text boxes. Other controls encompass complex behavior, such as a calendar controls, and controls that manage data connections.

The topics in this section describe what ASP.NET Web controls are and how to work with them.

 In This Section

[ASP.NET User Controls](http://msdn.microsoft.com/en-us/library/y6wb1a0e.aspx)

[ASP.NET Server Controls](http://msdn.microsoft.com/en-us/library/bb386416.aspx)

[ASP.NET Web Parts Controls](http://msdn.microsoft.com/en-us/library/e0s9t4ck.aspx)

**ASP.NET User Controls**

In addition to using Web server controls in your ASP.NET Web pages, you can create your own custom, reusable controls using the same techniques you use for creating ASP.NET Web pages. These controls are called user controls.

A user control is a kind of composite control that works much like an ASP.NET Web page—you can add existing Web server controls and markup to a user control, and define properties and methods for the control. You can then embed them in ASP.NET Web pages, where they act as a unit.

The topics in this section provide information on how user controls work, how to create them, and how to add them to ASP.NET Web pages.

**ASP.NET User Controls Overview**

At times, you might need functionality in a control that is not provided by the built-in ASP.NET Web server controls. In those cases, you can create your own controls. You have two options. You can create:

* User controls. User controls are containers into which you can put markup and Web server controls. You can then treat the user control as a unit and define properties and methods for it.
* Custom controls. A custom control is a class that you write that derives from [Control](http://msdn.microsoft.com/en-us/library/system.web.ui.control.aspx) or [WebControl](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.webcontrol.aspx).

User controls are substantially easier to create than custom controls, because you can reuse existing controls. They make it particularly easy to create controls with complex user interface elements.

This topic provides an overview of working with ASP.NET user controls.

 User Control Structure

An ASP.NET Web user control is similar to a complete ASP.NET Web page (.aspx file), with both a user interface page and code. You create the user control in much the same way you create an ASP.NET page and then add the markup and child controls that you need. A user control can include code to manipulate its contents like a page can, including performing tasks such as data binding.

A user controls differs from an ASP.NET Web page in these ways:

* The file name extension for the user control is .ascx.
* Instead of an [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive, the user control contains an [@ Control](http://msdn.microsoft.com/en-us/library/d19c0t4b.aspx) directive that defines configuration and other properties.
* User controls cannot run as stand-alone files. Instead, you must add them to ASP.NET pages, as you would any control.
* The user control does not have html, body, or form elements in it. These elements must be in the hosting page.

You can use the same HTML elements (except the html, body, or form elements) and Web controls on a user control that you do on an ASP.NET Web page. For example, if you are creating a user control to use as a toolbar, you can put a series of [Button](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.aspx) Web server controls onto the control and create event handlers for the buttons.

The following example shows a user control that implements a spinner control where users can click up and down buttons to rotate through a series of choices in a text box.

|  |
| --- |
| **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. For more information, see [Script Exploits Overview](http://msdn.microsoft.com/en-us/library/w1sw53ds.aspx). |

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl23VisualBasic');)

<%@ Control Language="VB" ClassName="UserControl1" %>

<script runat="server">

Protected colors As String() = {"Red", "Green", "Blue", "Yellow"}

Protected currentColorIndex As Integer = 0

Protected Sub Page\_Load(ByVal sender As Object, \_

ByVal e As System.EventArgs)

If IsPostBack Then

currentColorIndex = CInt(ViewState("currentColorIndex"))

Else

currentColorIndex = 0

DisplayColor()

End If

End Sub

Protected Sub DisplayColor()

textColor.Text = colors(currentColorIndex)

ViewState("currentColorIndex") = currentColorIndex.ToString()

End Sub

Protected Sub buttonUp\_Click(ByVal sender As Object, \_

ByVal e As System.EventArgs)

If currentColorIndex = 0 Then

currentColorIndex = colors.Length - 1

Else

currentColorIndex -= 1

End If

DisplayColor()

End Sub

Protected Sub buttonDown\_Click(ByVal sender As Object, \_

ByVal e As System.EventArgs)

If currentColorIndex = colors.Length - 1 Then

currentColorIndex = 0

Else

currentColorIndex += 1

End If

DisplayColor()

End Sub

</script>

<asp:TextBox ID="textColor" runat="server"

ReadOnly="True" />

<asp:Button Font-Bold="True" ID="buttonUp" runat="server"

Text="^" OnClick="buttonUp\_Click" />

<asp:Button Font-Bold="True" ID="buttonDown" runat="server"

Text="v" OnClick="buttonDown\_Click" />

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl24CSharp');)

<% @ Control Language="C#" ClassName="UserControl1" %>

<script runat="server">

protected int currentColorIndex;

protected String[] colors = {"Red", "Blue", "Green", "Yellow"};

protected void Page\_Load(object sender, EventArgs e)

{

if (IsPostBack)

{

currentColorIndex =

Int16.Parse(ViewState["currentColorIndex"].ToString());

}

else

{

currentColorIndex = 0;

DisplayColor();

}

}

protected void DisplayColor()

{

textColor.Text = colors[currentColorIndex];

ViewState["currentColorIndex"] = currentColorIndex.ToString();

}

protected void buttonUp\_Click(object sender, EventArgs e)

{

if(currentColorIndex == 0)

{

currentColorIndex = colors.Length - 1;

}

else

{

currentColorIndex -= 1;

}

DisplayColor();

}

protected void buttonDown\_Click(object sender, EventArgs e)

{

if(currentColorIndex == (colors.Length - 1))

{

currentColorIndex = 0;

}

else

{

currentColorIndex += 1;

}

DisplayColor();

}

</script>

<asp:TextBox ID="textColor" runat="server"

ReadOnly="True" />

<asp:Button Font-Bold="True" ID="buttonUp" runat="server"

Text="^" OnClick="buttonUp\_Click" />

<asp:Button Font-Bold="True" ID="buttonDown" runat="server"

Text="v" OnClick="buttonDown\_Click" />

Notice that the user control looks much like an ASP.NET page — it contains several controls (a [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) control and two [Button](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.aspx) controls) and code that handles the buttons' [Click](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.click.aspx) events and the page’s [Load](http://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) event. However, the control contains no markup except for the controls, and instead of an [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive it contains an [@ Control](http://msdn.microsoft.com/en-us/library/d19c0t4b.aspx) directive.

 Adding a User Control to a Page

You add a user control to a page by registering it on the host page. When you register it, you specify the .ascx file that contains the user control, a tag prefix, and a tag name that you will use to declare the user control on the page. For details, see [How to: Include a User Control in an ASP.NET Web Page](http://msdn.microsoft.com/en-us/library/sbz9etab.aspx).

 Defining Properties and Methods for a User Control

You can define properties and methods for a user control the same way you do for a page. By defining a property for a user control, you make it possible to set its properties declaratively and in code.

 Events in User Controls

When a user control contains Web server controls, you can write code in the user control to handle the events raised by the child controls. For example, if your user control contains a [Button](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.aspx) control, you can create a handler in the user control for the button's [Click](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.click.aspx) event.

By default, events raised by child controls in a user control are not available to the host page. However, you can define events for your user control and raise them so that the host page is notified of the event. You do this in the same way that you define events for any class. For more information, see [Raising an Event](http://msdn.microsoft.com/en-us/library/wkzf914z.aspx).

 Referencing External Resources

When a user control runs, references to external resources such as images or anchors to other pages are resolved using the URL of the user control as the base URL. For example, if the user control contains an [Image](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.image.aspx) control whose [ImageUrl](http://msdn.microsoft.com/en-us/library/system.web.ui.mobilecontrols.image.imageurl.aspx) property is set to Images/Button1.gif, the URL of the image is added to the URL of the user control to resolve the complete path to the image. If the user control references a resource that is not in a subfolder of the user control itself, you must provide a path that resolves to the correct folder at run time. For more information on specifying paths for ASP.NET server controls, see [ASP.NET Web Site Paths](http://msdn.microsoft.com/en-us/library/ms178116.aspx).

 Caching and User Controls

User controls can support caching directives that are separate from the host page. You can therefore add user controls to pages and to cache portions of a page. For details, see [Caching Portions of an ASP.NET Page](http://msdn.microsoft.com/en-us/library/h30h475z.aspx).

**ASP.NET Server Controls**

ASP.NET Web server controls are objects on ASP.NET Web pages that run when the page is requested and that render markup to the browser. Many Web server controls are similar to familiar HTML elements, such as buttons and text boxes. Other controls encompass complex behavior, such as a calendar controls, and controls that you can use to connect to data sources and display data.

ASP.NET also provides AJAX-enabled server controls. These consist of server and client code that integrate to produce rich client behavior. When you add an AJAX control to an ASP.NET Web page, the page automatically sends supporting client script to the browser for AJAX functionality. You can also provide additional client code to customize the functionality of a control, but this is not required.

**ASP.NET Web Server Controls Overview**

When you create ASP.NET Web pages, you can use these types of controls:

* HTML server controls   HTML elements exposed to the server so you can program them. HTML server controls expose an object model that maps very closely to the HTML elements that they render.
* Web server controls   Controls with more built-in features than HTML server controls. Web server controls include not only form controls such as buttons and text boxes, but also special-purpose controls such as a calendar, menus, and a tree view control. Web server controls are more abstract than HTML server controls in that their object model does not necessarily reflect HTML syntax.
* Validation controls   Controls that incorporate logic to enable you to what users enter for input controls such as the [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) control. Validation controls enable you to check for a required field, to test against a specific value or pattern of characters, to verify that a value lies within a range, and so on. For more information, see [Validation ASP.NET Controls](http://msdn.microsoft.com/en-us/library/debza5t0.aspx).
* User controls   Controls that you create as ASP.NET Web pages. You can embed ASP.NET user controls in other ASP.NET Web pages, which is an easy way to create toolbars and other reusable elements. For more information, see [ASP.NET User Controls](http://msdn.microsoft.com/en-us/library/y6wb1a0e.aspx).

|  |
| --- |
| **Note:** |
| You can also create output for mobile devices. To do so, you use the same ASP.NET page framework, but you create Mobile ASP.NET Web pages instead of ASP.NET Web pages and use controls specifically designed for mobile devices. For details, see [Creating ASP.NET Mobile Web Pages](http://msdn.microsoft.com/en-us/library/8htez1ds.aspx). |

You can use all types of controls on the same page. The following sections provide more detail about ASP.NET server controls.

|  |
| --- |
| **Note:** |
| In some situations, server controls require client script in order to function properly. If a user has disabled scripting in the browser, the controls might not function as you intend. For details, see [ASP.NET Web Server Controls and Browser Capabilities](http://msdn.microsoft.com/en-us/library/x3k2ssx2.aspx). |

 HTML Server Controls

HTML server controls are HTML elements (or elements in other supported markup, such as XHTML) containing attributes that make them programmable in server code. By default, HTML elements on an ASP.NET Web page are not available to the server. Instead, they are treated as opaque text and passed through to the browser. However, by converting HTML elements to HTML server controls, you expose them as elements you can program on the server.

The object model for HTML server controls maps closely to that of the corresponding elements. For example, HTML attributes are exposed in HTML server controls as properties.

Any HTML element on a page can be converted to an HTML server control by adding the attribute runat="server". During parsing, the ASP.NET page framework creates instances of all elements containing the runat="server" attribute. If you want to reference the control as a member within your code, you should also assign an id attribute to the control.

The page framework provides predefined HTML server controls for the HTML elements most commonly used dynamically on a page: the form element, the input elements (text box, check box, Submit button), the select element, and so on. These predefined HTML server controls share the basic properties of the generic control, and in addition, each control typically provides its own set of properties and its own event.

HTML server controls offer the following features:

* An object model that you can program against on the server using familiar object-oriented techniques. Each server control exposes properties that enable you to manipulate the control's markup attributes programmatically in server code.
* A set of events for which you can write event handlers in much the same way you would in a client-based form, except that the event is handled in server code.
* The ability to handle events in client script.
* Automatic maintenance of the control's state. When the page makes a round trip to the server, the values that the user entered into HTML server controls are automatically maintained and sent back to the browser.
* Interaction with ASP.NET validation controls so you can verify that a user has entered appropriate information into a control.
* Data binding to one or more properties of the control.
* Support for styles if the ASP.NET Web page is displayed in a browser that supports cascading style sheets.
* Pass-through of custom attributes. You can add any attributes you need to an HTML server control and the page framework will render them without any change in functionality. This enables you to add browser-specific attributes to your controls.

For details about how to convert an HTML element to an HTML server control, see [How to: Add HTML Server Controls to a Web Page Using ASP.NET Syntax](http://msdn.microsoft.com/en-us/library/s37470tf.aspx).

 Web Server Controls

Web server controls are a second set of controls designed with a different emphasis. They do not necessarily map one-to-one to HTML server controls. Instead, they are defined as abstract controls in which the actual markup rendered by the control can be quite different from the model that you program against. For example, a [RadioButtonList](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.radiobuttonlist.aspx) Web server control might be rendered in a table or as inline text with other markup.

Web server controls include traditional form controls such as buttons and text boxes as well as complex controls such as tables. They also include controls that provide commonly used form functionality such as displaying data in a grid, choosing dates, displaying menus, and so on.

Web server controls offer all of the features described above for HTML server controls (except one-to-one mapping to elements) and these additional features:

* A rich object model that provides type-safe programming capabilities.
* Automatic browser detection. The controls can detect browser capabilities and render appropriate markup.
* For some controls, the ability to define your own layout for the control using [Templates](http://msdn.microsoft.com/en-us/library/system.web.ui.design.templategroup.templates.aspx).
* For some controls, the ability to specify whether a control's event causes immediate posting to the server or is instead cached and raised when the page is submitted.
* Support for themes, which enable you to define a consistent look for controls throughout your site. For details, see [ASP.NET Themes and Skins](http://msdn.microsoft.com/en-us/library/wcyt4fxb.aspx).
* Ability to pass events from a nested control (such as a button in a table) to the container control.

The controls use syntax such as the following:

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl23other');)

<asp:button attributes runat="server" id="Button1" />

The attributes in this case are not those of HTML elements. Instead, they are properties of the Web control.

When the ASP.NET Web page runs, the Web server control is rendered on the page using appropriate markup, which often depends not only on the browser type but also on settings that you have made for the control. For example, a [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) control might render as an input tag or a textarea tag, depending on its properties.

Detailed how-to and reference documentation is available for each control separately. For more information, see [Individual ASP.NET Web Server Controls](http://msdn.microsoft.com/en-us/library/fxh7k08z.aspx).

**ASP.NET Web Parts Controls**

ASP.NET Web Parts controls are an integrated set of controls for creating Web sites that enable end users to modify the content, appearance, and behavior of Web pages directly in a browser. The topics in this section provide information on what Web Parts are, how they work, and how to use them to create user-customizable ASP.NET Web pages.

**ASP.NET Web Parts Overview**

ASP.NET Web Parts is an integrated set of controls for creating Web sites that enable end users to modify the content, appearance, and behavior of Web pages directly from a browser. The modifications can be applied to all users on the site or to individual users. When users modify pages and controls, the settings can be saved to retain a user's personal preferences across future browser sessions, a feature called personalization. These Web Parts capabilities mean that developers can empower end users to personalize a Web application dynamically, without developer or administrator intervention.

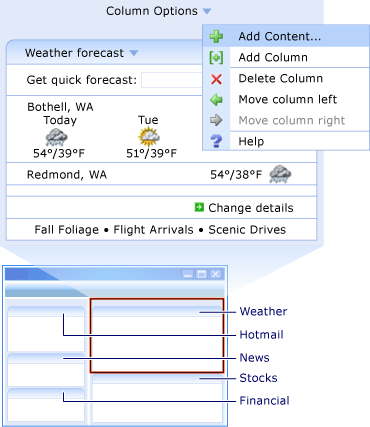
Using the Web Parts control set, you as a developer can enable end users to:

* Personalize page content. Users can add new Web Parts controls to a page, remove them, hide them, or minimize them like ordinary windows.
* Personalize page layout. Users can drag a Web Parts control to a different zone on a page, or change its appearance, properties, and behavior.
* Export and import controls. Users can import or export Web Parts control settings for use in other pages or sites, retaining the properties, appearance, and even the data in the controls. This reduces data entry and configuration demands on end users.
* Create connections. Users can establish connections between controls so that, for example, a chart control could display a graph for the data in a stock ticker control. Users could personalize not only the connection itself, but the appearance and details of how the chart control displays the data.
* Manage and personalize site-level settings. Authorized users can configure site-level settings, determine who can access a site or page, set role-based access to controls, and so on. For example, a user in an administrative role could set a Web Parts control to be shared by all users, and prevent users who are not administrators from personalizing the shared control.

 Web Parts Essentials

The Web Parts control set consists of three main building blocks: personalization, user interface (UI) structural components, and actual Web Parts UI controls. For more details, see [Web Parts Control Set Overview](http://msdn.microsoft.com/en-us/library/k3w2y2tf.aspx). Much of your development effort will focus on Web Parts controls, which are simply ASP.NET controls that can use the features of the Web Parts control set.

As an example of how Web Parts controls can be used to build personalizable Web pages, examine the following screen shot.

Typical Web Parts page  


This page contains several basic elements of a Web Parts application:

* Use of zones for page layout. There are two columns that can contain controls: one has the Weather and Stock Quotes controls, the other has Hotmail and News controls. These columns in Web Parts terminology are called zones--regions on a page that contain Web Parts controls. Zones exist to lay out Web Parts controls on a page, and to provide a common UI for the controls. There can be one or many zones on a page, each zone can contain one or many Web Parts controls, and each zone can have a vertical or horizontal orientation for page layout.
* Web Parts controls within the zones. Each control has UI verbs (actions that a user can perform) that can appear as links, buttons, or clickable images on the control. In the preceding screen shot, notice that each control has a button in its title bar that exposes a drop-down menu. In the menus for each control are options to change details particular to that control, and other options to carry out common actions such as moving or deleting a control, and getting help. Some controls, such as the Weather control, allow users to personalize them so the controls display only information relevant to the user.
* Links to enable extensive personalization. These allow users to change the content, color, and layout of the page. For instance, if users click the Add Column link, a Web Parts application could enable them to add another column to a page. Or users could click the Add Content link, which displays a catalog of controls that that they can optionally add to the page. One of those could be a stock charting control. A user could add that control to one of the zones on the page, and could then connect it to the existing Stock Quotes control to chart the stock data it contains.

Developer Scenarios for Using Web Parts

You will typically work with Web Parts in one of three ways: creating pages that use Web Parts controls, creating individual Web Parts controls, or creating complete, personalizable Web applications, such as a portal.

### Page Development

Page developers can use visual design tools such as Microsoft Visual Studio 2005 to create pages that use Web Parts. One advantage in using a tool such as Visual Studio is that the Web Parts control set provides features for drag-and-drop creation and configuration of Web Parts controls in a visual designer. For example, you can use the designer to drag a Web Parts zone, or a Web Parts editor control, onto the design surface, and then configure the control right in the designer using the UI provided by the Web Parts control set. This can speed development of Web Parts applications and reduce the amount of code you have to write.

### Control Development

You can use any existing ASP.NET control as a Web Parts control, including standard Web server controls, custom server controls, and user controls. For maximum programmatic control of your environment, you can also create custom Web Parts controls that derive from the [WebPart](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.webparts.webpart.aspx) class. For individual Web Parts control development, you will typically either create a user control and use it as a Web Parts control, or develop a custom Web Parts control.

As an example of developing a custom Web Parts control, you could create a control to provide any of the features provided by other ASP.NET server controls that might be useful to package as a personalizable Web Parts control: calendars, lists, financial information, news, calculators, rich text controls for updating content, editable grids that connect to databases, charts that dynamically update their displays, or weather and travel information. If you provide a visual designer with your control, then any page developer using Visual Studio can simply drag your control into a Web Parts zone and configure it at design time without having to write additional code.

### Web Application Development

Developing fully integrated and personalizable Web applications--such as a portal-- involves the most comprehensive use of Web Parts. You can develop a Web site that allows extensive user personalization of the UI and content--with features similar to [MSN](http://www.msn.com/). Or you can even develop a packaged application that can be shipped and used by companies or fee-based ISPs that provide portal hosting services.

In a Web application scenario, you could offer a complete solution for end users to manage and personalize the application. This could include a set of Web Parts controls that provide the desired features for the site, a consistent set of themes and styles that allow end users to personalize the UI in a consistent way, catalogs of Web Parts controls from which users can select the ones they want to appear on a page, authentication services, and role-based management (for example, allowing administrative users to personalize Web Parts controls and site settings for all users).

For each part of your application, you can extend the Web Parts control set as needed to provide greater control over the environment. For example, besides authoring custom Web Parts controls for the primary UI of your pages, you might also want to develop a custom Web Parts catalog that is consistent with the look and feel of your application, and gives users more flexibility to choose how controls are added to a page. Or you could extend a zone control to provide additional UI options for the Web Parts controls it contains. You could also write a custom personalization provider to give more flexibility and control over how the personalization data is stored and managed.