**Introduction to ASP.NET Events**





[**http://msdn.microsoft.com/rampup**](http://msdn.microsoft.com/rampup)

Contents

[Introduction 3](#_Toc223528907)

[Application & Session Events 3](#_Toc223528908)

[Page Events 4](#_Toc223528909)

[Control Events 7](#_Toc223528910)

[Client Events 10](#_Toc223528911)

## Introduction

An important feature of ASP.NET is that it allows you to program Web pages using an event-based model that is similar to that in client applications. As a simple example, you can add a button to an ASP.NET Web page and then write an event handler for the button's click event. Although this is common in Web pages that work exclusively with client script (by handling the button's **onclick** event in dynamic HTML), ASP.NET brings this model to server-based processing.

Events raised by ASP.NET server controls work somewhat differently than events in traditional HTML pages or in client-based Web applications. The difference arises primarily because of the separation of the event itself from where the event is handled. In client-based applications, events are raised and handled on the client. In ASP.NET Web pages, however, events associated with server controls originate on the client (browser) but are handled on the Web server by the ASP.NET page.

For events raised on the client, the ASP.NET Web control event model requires that the event information be captured on the client and an event message transmitted to the server, through an HTTP post. The page must interpret the post to determine what event occurred and then call the appropriate method in your code on the server to handle the event.

ASP.NET handles the task of capturing, transmitting, and interpreting the event. When you create event handlers in an ASP.NET Web page, you can typically do so without thinking about how the event information is captured and made available to your code. Instead, you can create event handlers in much the same way you would in a traditional client form. However, there are some aspects of event handling in ASP.NET Web pages that you should be aware of.

### Application & Session Events

In addition to page and control events, ASP.NET provides ways for you to work with life-cycle events that are raised when your application starts or stops or when an individual user's session starts or stops, including the following:

* Application events are raised for all requests to an application. For example, the [BeginRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.beginrequest.aspx) event of the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) object (Application\_BeginRequest) is raised when any ASP.NET Web page or XML Web service in your application is requested. This event allows you to initialize resources that will be used for each request to the application. A corresponding event, the [EndRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.endrequest.aspx) event of the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) object (Application\_EndRequest), provides you with an opportunity to close or otherwise dispose of resources used for the request.
* Session events are similar to application events (there is a [Start](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.start.aspx) and an [End](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.end.aspx) event), but are raised with each unique session within the application. A session begins when a user requests a page for the first time from your application and ends either when your application explicitly closes the session or when the session times out.

You can create handlers for these types of events in the Global.asax file.

### Page Events

Within each stage of the life cycle of a page, the page raises events that you can handle to run your own code. For control events, you bind the event handler to the event, either declaratively using attributes such as onclick, or in code.

Pages also support automatic event wire-up, meaning that ASP.NET looks for methods with particular names and automatically runs those methods when certain events are raised. If the AutoEventWireup attribute of the [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive is set to true (or if it is not defined, since by default it is true), page events are automatically bound to methods that use the naming convention of Page\_event, such as Page\_Load and Page\_Init. For more information on automatic event wire-up, see [ASP.NET Web Server Control Event Model](http://msdn.microsoft.com/en-us/library/y3bwdsh3.aspx).

The following table lists the page life-cycle events that you will use most frequently. There are more events than those listed; however, they are not used for most page processing scenarios. Instead, they are primarily used by server controls on the ASP.NET Web page to initialize and render themselves. If you want to write your own ASP.NET server controls, you need to understand more about these stages. For information about creating custom controls, see [Developing Custom ASP.NET Server Controls](http://msdn.microsoft.com/en-us/library/zt27tfhy.aspx).

|  |  |
| --- | --- |
| Page Event | Typical Use |
| [PreInit](http://msdn.microsoft.com/en-us/library/system.web.ui.page.preinit.aspx)  | Use this event for the following:* Check the [IsPostBack](http://msdn.microsoft.com/en-us/library/system.web.ui.page.ispostback.aspx) property to determine whether this is the first time the page is being processed.
* Create or re-create dynamic controls.
* Set a master page dynamically.
* Set the [Theme](http://msdn.microsoft.com/en-us/library/system.web.ui.page.theme.aspx) property dynamically.
* Read or set profile property values.

ms178472.alert_note(en-us,VS.90).gif**Note:**If the request is a postback, the values of the controls have not yet been restored from view state. If you set a control property at this stage, its value might be overwritten in the next event.  |
| [Init](http://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx)  | Raised after all controls have been initialized and any skin settings have been applied. Use this event to read or initialize control properties. |
| [InitComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.initcomplete.aspx)  | Raised by the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object. Use this event for processing tasks that require all initialization be complete. |
| [PreLoad](http://msdn.microsoft.com/en-us/library/system.web.ui.page.preload.aspx)  | Use this event if you need to perform processing on your page or control before the [Load](http://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) event. Before the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) instance raises this event, it loads view state for itself and all controls, and then processes any postback data included with the [Request](http://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) instance. |
| [Load](http://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx)  | The [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) calls the [OnLoad](http://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) event method on the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx), then recursively does the same for each child control, which does the same for each of its child controls until the page and all controls are loaded.Use the [OnLoad](http://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) event method to set properties in controls and establish database connections. |
| Control events | Use these events to handle specific control events, such as a [Button](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.aspx) control's [Click](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.click.aspx) event or a [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) control's [TextChanged](http://msdn.microsoft.com/en-us/library/system.web.ui.mobilecontrols.textbox.textchanged.aspx) event.ms178472.alert_note(en-us,VS.90).gif**Note:**In a postback request, if the page contains validator controls, check the [IsValid](http://msdn.microsoft.com/en-us/library/system.web.ui.page.isvalid.aspx) property of the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) and of individual validation controls before performing any processing. |
| [LoadComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.loadcomplete.aspx)  | Use this event for tasks that require that all other controls on the page be loaded. |
| [PreRender](http://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx)  | Before this event occurs:* The [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object calls [EnsureChildControls](http://msdn.microsoft.com/en-us/library/system.web.ui.control.ensurechildcontrols.aspx) for each control and for the page.
* Each data bound control whose [DataSourceID](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.databoundcontrol.datasourceid.aspx) property is set calls its [DataBind](http://msdn.microsoft.com/en-us/library/system.web.ui.control.databind.aspx) method. For more information, see [Data Binding Events for Data-Bound Controls](http://msdn.microsoft.com/en-us/library/ms178472.aspx#databindingevents) later in this topic.

The [PreRender](http://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event occurs for each control on the page. Use the event to make final changes to the contents of the page or its controls. |
| [SaveStateComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.savestatecomplete.aspx)  | Before this event occurs, [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) has been saved for the page and for all controls. Any changes to the page or controls at this point will be ignored.Use this event perform tasks that require view state to be saved, but that do not make any changes to controls. |
| [Render](http://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx)  | This is not an event; instead, at this stage of processing, the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object calls this method on each control. All ASP.NET Web server controls have a [Render](http://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method that writes out the control's markup that is sent to the browser.If you create a custom control, you typically override this method to output the control's markup. However, if your custom control incorporates only standard ASP.NET Web server controls and no custom markup, you do not need to override the [Render](http://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method. For more information, see [Developing Custom ASP.NET Server Controls](http://msdn.microsoft.com/en-us/library/zt27tfhy.aspx).A user control (an .ascx file) automatically incorporates rendering, so you do not need to explicitly render the control in code. |
| [Unload](http://msdn.microsoft.com/en-us/library/system.web.ui.control.unload.aspx)  | This event occurs for each control and then for the page. In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections.For the page itself, use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks.ms178472.alert_note(en-us,VS.90).gif**Note:**During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream. If you attempt to call a method such as the Response.Write method, the page will throw an exception. |

ASP.NET pages raise life-cycle events such as Init, Load, PreRender, and others. By default, you can bind page events to methods using a naming convention of Page\_eventname. For example, to create a handler for the page's Load event, you can create a method named Page\_Load. At run time, ASP.NET will find methods based on this naming convention and automatically perform the binding between the event and the method. You can use the convention of Page\_eventname for any event exposed by the Page class.

If you prefer, you can bind handlers to events explicitly. The automatic binding of page events based on the method naming convention is controlled by a page property named AutoEventWireup. By default, for C#, this property is set to true, and ASP.NET performs the automatic lookup and binding described earlier. Alternatively, you can set this property to false by adding the attribute AutoEventWireup=false in the [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive. You can then create methods with any name and bind them to page events explicitly.

By default, for Visual Basic, this property is set to false. In Visual Basic, handlers are bound to events by using the [Handles](http://msdn.microsoft.com/en-us/library/6k46st1y.aspx) keyword. This keyword is inserted automatically by Visual Studio as part of the method that you create when you select a page event from the drop-down box. The following example illustrates use of the [Handles](http://msdn.microsoft.com/en-us/library/6k46st1y.aspx) keyword:

Visual Basic

Sub MyPageLoad(sender As Object, e As EventArgs) **Handles MyBase.Load**

One disadvantage of the AutoEventWireup attribute is that it requires that the page event handlers have specific, predictable names. This limits your flexibility in how you name event handlers. Another disadvantage is that performance is adversely affected, because ASP.NET searches for methods at run-time. For a Web site with high traffic volumes, the impact on performance could be significant.

### Control Events

Unlike events in desktop applications, ASP.NET server-control events are raised as well as handled on the server. When a Web request communicates a client-side action to the server, a control can raise events on the server in response to the client action. The event is handled by the page or by its child controls, and ASP.NET sends a response back to the client. This results in a user experience similar to that of a desktop application. However, control developers must understand that only one client-side event is posted to the server — the postback event. Common user-interface events such as mouse clicks or key presses are not communicated to the server and thus cannot be processed on the server.

The base class **System.Web.UI.Control** provides the events that govern a control's execution lifecycle, such as initialization, loading, and unloading. The server loads an ASP.NET page every time it is requested and then unloads it after the request is completed. The page and the server controls it contains are responsible for executing the request and rendering HTML back to the client. Although the communication between the client and the server is stateless and disconnected, the client experience must appear to be that of a continuously executing process.

This illusion of continuity is created by the ASP.NET page framework and by the page and its controls. On postback, a control must behave as if it were starting where it left off at the end of the previous Web request. The ASP.NET page framework makes it relatively easy to perform state management, but control developers must be aware of the control execution sequence to achieve the effect of continuity. Control developers need to understand which information is available to a control at each phase in its lifecycle, which data is persisted, and what the control's state is when it is rendered. For example, a control is unable to invoke its parent until the tree of controls on a page has been populated.

The following table provides a high-level overview of the phases in the lifecycle of a control. For details, follow the links in the table.

|  |  |  |
| --- | --- | --- |
| **Phase** | **What a control needs to do** | **Method or event to override** |
| Initialize | Initialize settings needed during the lifetime of the incoming Web request.. | **Init** event (**OnInit** method) |
| Load view state | At the end of this phase, the **ViewState** property of a control is automatically populated. A control can override the default implementation of the **LoadViewState** method to customize state restoration. | **LoadViewState** method |
| Process postback data | Process incoming form data and update properties accordingly. **Note**   Only controls that process postback data participate in this phase. | **LoadPostData** method (if **IPostBackDataHandler** is implemented) |
| Load | Perform actions common to all requests, such as setting up a database query. At this point, server controls in the tree are created and initialized, the state is restored, and form controls reflect client-side data.  | **Load** event (**OnLoad** method) |
| Send postback change notifications | Raise change events in response to state changes between the current and previous postbacks. **Note**   Only controls that raise postback change events participate in this phase. | **RaisePostDataChangedEvent** method (if **IPostBackDataHandler** is implemented) |
| Handle postback events | Handle the client-side event that caused the postback and raise appropriate events on the server. **Note**   Only controls that process postback events participate in this phase. | **RaisePostBackEvent** method (if **IPostBackEventHandler** is implemented) |
| Prerender | Perform any updates before the output is rendered. Any changes made to the state of the control in the prerender phase can be saved, while changes made in the rendering phase are lost. | **PreRender** event (**OnPreRender** method) |
| Save state | The **ViewState** property of a control is automatically persisted to a string object after this stage. This string object is sent to the client and back as a hidden variable. For improving efficiency, a control can override the **SaveViewState** method to modify the **ViewState** property. | **SaveViewState** method |
| Render | Generate output to be rendered to the client. | **Render** method |
| Dispose | Perform any final cleanup before the control is torn down. References to expensive resources such as database connections must be released in this phase. | **Dispose** method |
| Unload | Perform any final cleanup before the control is torn down. Control authors generally perform cleanup in **Dispose** and do not handle this event.  | **UnLoad** event (**On UnLoad** method) |

**Note** To override an *EventName* event, override the **On***EventName* method (and call base. **On***EventName*).

The methods and events in the third column are inherited from **System.Web.UI.Control**, with the following exceptions: **LoadPostData** and **RaisePostDataChangedEvent** are methods of the **IPostBackDataHandler** interface, and **RaisePostBackEvent** belongs to the **IPostBackEventHandler** interface. If your control participates in postback data processing you must implement **IPostBackDataHandler**. If your control receives postback events you must implement **IPostBackEventHandler**.

The **CreateChildControls** method is not listed in the table because it is called whenever the ASP.NET page framework needs to create the controls tree and this method call is not limited to a specific phase in a control's lifecycle. For example, **CreateChildControls** can be invoked when loading a page, during data binding, or during rendering.

### Client Events

ASP.NET Web applications are not limited to server-based tools and languages. You can include ECMAScript (JavaScript or JScript) in your ASP.NET Web pages to create rich browser-based functionality. A wide range of features for client script support is available in ASP.NET.

One option is to create and add individual snippets of client script to ASP.NET Web pages to support browser behavior that you design for your application. This option is practical if you want to include only a few small pieces of JavaScript code or if you are working with JavaScript code that you already have. This option also helps keep the size of your ASP.NET Web pages to a minimum.

Alternatively, you can take advantage of the powerful AJAX features of ASP.NET. These AJAX features include a complete framework that supports the following:

* Object-oriented programming in JavaScript.
* Asynchronous postbacks.
* Application Web services for authentication and profiles.
* Server controls that manage client script with extended debugging and trace support.

Using ASP.NET AJAX features lets you take advantage of JavaScript functionality with less manual coding. It provides extended JavaScript functionality, and it provides a framework for creating client functionality that is integrated into ASP.NET.

Because ASP.NET Web pages just render HTML markup, you can add your own client script to ASP.NET pages. Client script is supported in ASP.NET pages to the extent that the browser requesting the page supports client script. If the page is running in a browser on a mobile phone or other device, the degree of support for client script varies, depending on the browser.

There are several options for including client script in ASP.NET pages:

* You can include client script statically in a script block that includes code or that uses an include attribute to reference a JavaScript (.js) file. Use this option to insert script blocks or JavaScript files contain client script that you do not have to create dynamically, and that do not require additional AJAX functionality provided by the Microsoft AJAX Library.
* You can dynamically create and add client script to ASP.NET Web page by using the [ClientScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.clientscriptmanager.aspx) class. Use this option when you want to create scripts that depend on information that is available only at run time.
* If you plan to take advantage of the AJAX features of ASP.NET, you can manage client-script files by using the [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) server control.
* The [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) server control also ensures that the Microsoft AJAX Library is loaded on the browser before your scripts run.

##### Including Static Client Script Blocks

You can add script blocks to an ASP.NET page just as you would for any HTML page. You can use client script to write event handlers for client events such as the page's onload event. When an ASP.NET page is running in the browser, the markup elements on the page are addressable in client script. They raise all the client events that they do in an HTML page.

An ASP.NET Web page can also access a script file by referring to it in the src attribute of a <script> tag, as in the following example:

<script type="text/javascript" src="MyScript.js"></script>

Keeping client script in external .js files rather than in the pages themselves can help organize your client scripts. It can also make them easier to manage for version control and easier to share between pages.

External .js files are cached by the browser, similar to the way Web pages and images are cached. After the script has been loaded by the browser as an external file, it is available in the cache to any other Web page that requires it. This can help increase the performance of the Web application.

##### Creating Client Script Dynamically

In many cases, you can create the client script for your page declaratively, usually as a script block. However, you can also create client script dynamically. This is useful if the script depends on information that is available only at run time. For example, you might insert client script into a page that addresses a server control whose name (ID) is not known until the application runs, or you might create script that depends on values that you get from a user.

You can create and insert client script dynamically into a rendered page by calling methods of the [ClientScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.clientscriptmanager.aspx) class, such as the following:

* [RegisterClientScriptBlock](http://msdn.microsoft.com/en-us/library/system.web.ui.clientscriptmanager.registerclientscriptblock.aspx), which inserts a script block at the top of the rendered page.
* [RegisterStartupScript](http://msdn.microsoft.com/en-us/library/system.web.ui.clientscriptmanager.registerstartupscript.aspx), which inserts a script block at the end of the rendered page.

The following example shows how to add dynamically generated client script to the page. The code checks whether a check box named checkDisplayCount is selected. If so, the code performs the following tasks:

* It creates a client script function that uses a span element to display the character count in a [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) control named TextBox1.
* It adds a client event to the [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) control.
* It generates the span element.

The code assumes that the page contains a check box named checkDisplayCount whose [AutoPostBack](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.checkbox.autopostback.aspx) property is set to true and a [PlaceHolder](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.placeholder.aspx) control named PlaceHolder1.

End Sub

C#

void Page\_Load(object sender, EventArgs e)

{

 if(checkDisplayCount.Checked)

 {

 String scriptText = "";

 scriptText += "function DisplayCharCount(){";

 scriptText += " spanCounter.innerText = " +

 " document.forms[0].TextBox1.value.length";

 scriptText += "}";

 ClientScriptManager.RegisterClientScriptBlock(this.GetType(),

 "CounterScript", scriptText, true);

 TextBox1.Attributes.Add("onkeyup", "DisplayCharCount()");

 LiteralControl spanLiteral = new

 LiteralControl("<span id=\"spanCounter\"></span>");

 PlaceHolder1.Controls.Add(spanLiteral);

 }

}