**AJAX Client Life-Cycle Events**

This page is specific to **Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following: [Microsoft Visual Studio 2010/.NET Framework 4.0](http://msdn.microsoft.com/en-us/library/bb386417(VS.100).aspx)

An AJAX-enabled ASP.NET page raises the same server life-cycle events as an ASP.NET 2.0 Web page and in addition raises client life-cycle events. The client events enable you to customize the UI for both postbacks and for asynchronous postbacks (partial-page updates). The client events also help you manage custom script components during the lifetime of the page in the browser.

The client events are raised by classes in the Microsoft AJAX Library. These classes are automatically instantiated when a page contains ASP.NET AJAX server controls. The client classes provide APIs that enable you to bind to events and to provide handlers for those events. Because the Microsoft AJAX Library is browser independent, the code you write for your handlers works the same in all supported browsers.

The key event for initial requests (GET requests) and synchronous postbacks is the [load](http://msdn.microsoft.com/en-us/library/bb383829.aspx) event of the [Application](http://msdn.microsoft.com/en-us/library/bb310856.aspx) instance. When script in a **load** event handler runs, all scripts and components have been loaded and are available. When partial-page rendering with [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) controls is enabled, the key client events are the events of the [PageRequestManager](http://msdn.microsoft.com/en-us/library/bb311028.aspx) class. These events enable you to handle many common scenarios. These include the ability to cancel postbacks, to give precedence to one postback over another, and to animate [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) controls when their content is refreshed.

Client events are useful whether you are creating pages or writing components. If you are a page developer, you can provide custom script that is called when the page loads and unloads in the browser.

For more information about the server life-cycle events, see [ASP.NET Page Life Cycle Overview](http://msdn.microsoft.com/en-us/library/ms178472.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Client Classes

The two main Microsoft AJAX Library classes that raise events during the client life cycle of an ASP.NET AJAX Web page are the [Application](http://msdn.microsoft.com/en-us/library/bb310856.aspx) and [PageRequestManager](http://msdn.microsoft.com/en-us/library/bb311028.aspx) classes.

The **Application** class is instantiated in the browser when the page contains a [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control. The **Application** class resembles the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) server control, which derives from the [Control](http://msdn.microsoft.com/en-us/library/system.web.ui.control.aspx) class, but provides additional functionality for raising server events. Similarly, the **Application** class derives from the [Sys.Component](http://msdn.microsoft.com/en-us/library/bb397516.aspx) class, but raises client life-cycle events that you can handle.

If a page contains a [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control and one or more [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) controls, the page can perform partial-page updates (if partial-page rendering is enabled and supported in the browser). In that case, an instance of the **PageRequestManager** class is automatically available in the browser. The **PageRequestManager** class raises client events that are specific to asynchronous postbacks. For details about partial-page rendering, see [Partial-Page Rendering Overview](http://msdn.microsoft.com/en-us/library/bb386573.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Adding Handlers for Client Events

To add or remove handlers for events raised by the **Application** and **PageRequestManager** classes, use the **add\_***eventname* and **remove\_***eventname* methods of those classes. The following example shows how to add a handler named MyLoad to the [init](http://msdn.microsoft.com/en-us/library/bb397532.aspx) event of the **Application** object.

JavaScript

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl90_ctl00_ctl01');" \o "Copy Code)

Sys.Application.add\_init(MyInit);

function MyInit(sender) {

}

Sys.Appplication.remove\_init(MyInit);

|  |
| --- |
| **NoteNote:** |
| This example shows just the syntax of the **add\_***eventname* and **remove\_***eventname* methods. Details about what you can do with specific events are provided later in this topic. |

**Handling the Application Load and Unload Events**

To handle the **load** and [unload](http://msdn.microsoft.com/en-us/library/bb310821.aspx) events of the **Application** object, you do not have to explicitly bind a handler to the event. Instead, you can create functions that use the reserved names pageLoad and pageUnload. The following example shows how to add a handler for the **load** event of the **Application** object by using this approach.

JavaScript

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl90_ctl00_ctl04');" \o "Copy Code)

function pageLoad(sender, args) {

}

**Events for Other Client Classes**

This topic describes only the events that are raised by the **Application** and **PageRequestManager** classes. The Microsoft AJAX Library also contains classes for adding, clearing, and removing handlers for DOM element events. These classes include the following:

* The [Sys.UI.DomEvent.addHandler](http://msdn.microsoft.com/en-us/library/bb310798.aspx) method or the shortcut [$addHandler](http://msdn.microsoft.com/en-us/library/bb311019.aspx).
* The [Sys.UI.DomEvent.clearHandlers](http://msdn.microsoft.com/en-us/library/bb310816.aspx) method or the shortcut [$clearHandlers](http://msdn.microsoft.com/en-us/library/bb310959.aspx).
* The [Sys.UI.DomEvent.removeHandler](http://msdn.microsoft.com/en-us/library/bb310935.aspx) method or the shortcut [$removeHandler](http://msdn.microsoft.com/en-us/library/bb397510.aspx).

Events raised by DOM elements are not discussed in this topic.

http://i.msdn.microsoft.com/Global/Images/clear.gif Client Events of the Application and PageRequestManager Classes

The following table lists client events of the **Application** and **PageRequestManager** classes that you can handle in AJAX ASP.NET-enabled pages. The order in which the events are raised is described later in this topic.

|  |  |
| --- | --- |
| **Event** | **Description** |
| [Sys.Application.init Event](http://msdn.microsoft.com/en-us/library/bb397532.aspx) | Raised after all scripts have been loaded but before any objects are created. If you are writing a component, the **init** event gives you a point in the life cycle to add your component to the page. The component can then be used by other components or by script later in the page life cycle. If you are a page developer, you should use the **load** event instead of the **init** event for most scenarios.  The **init** event is raised only one time when the page is first rendered. Subsequent partial-page updates do not raise the **init** event. |
| [Sys.Application.load Event](http://msdn.microsoft.com/en-us/library/bb383829.aspx) | Raised after all scripts have been loaded and all objects in the application that are created by using [$create](http://msdn.microsoft.com/en-us/library/bb397487.aspx) are initialized. The **load** event is raised for all postbacks to the server, which includes asynchronous postbacks.  If you are a page developer, you can create a function that has the name pageLoad, which automatically provides a handler for the **load** event. The pageLoad handler is called after any handlers that have been added to the **load** event by the **add\_load** method.  The **load** event takes an *eventargs* parameter, which is an [Sys.ApplicationLoadEventArgs](http://msdn.microsoft.com/en-us/library/bb383787.aspx) object. You can use the event arguments to determine whether the page is being refreshed as a result of a partial-page update and what components were created since the previous **load** event was raised. |
| [Sys.Application.unload Event](http://msdn.microsoft.com/en-us/library/bb310821.aspx) | Raised before all objects are disposed and before the browser window's **window.unload** event occurs.  If you are a page developer, you can create a function that has the name pageUnload, which automatically provides a handler for the **unload** event. The pageUnload event is called just before the page is unloaded from the browser. During this event, you should free any resources that your code is holding. |
| [Sys.Component.propertyChanged Event](http://msdn.microsoft.com/en-us/library/bb311023.aspx) | Potentially raised when a property of a component changes. This event is raised only if a component developer has called the [Sys.Component.raisePropertyChange](http://msdn.microsoft.com/en-us/library/bb397463.aspx) method in a property set accessor. For more information, see [Defining Custom Component Properties and Raising PropertyChanged Events](http://msdn.microsoft.com/en-us/library/bb398901.aspx).  The **propertyChanged** event takes an *eventargs* parameter, which is a [Sys.applicationLoadEventArgs](http://msdn.microsoft.com/en-us/library/bb383787.aspx) object. |
| [Sys.Component.disposing Event](http://msdn.microsoft.com/en-us/library/bb397541.aspx) | Raised when the **Application** instance is disposed. |
| [Sys.WebForms.PageRequestManager initializeRequest Event](http://msdn.microsoft.com/en-us/library/bb397460.aspx) | Raised before an asynchronous request starts. You can use this event to cancel a postback, such as to give precedence to another asynchronous postback.  The **initializeRequest** event takes an *eventargs* parameter, which is a [Sys.WebForms.InitializeRequestEventArgs](http://msdn.microsoft.com/en-us/library/bb311030.aspx) object. This object makes available the element that caused the postback and the underlying request object. **InitializeRequestEventArgs** also exposes a [cancel](http://msdn.microsoft.com/en-us/library/bb310859.aspx) property. If you set **cancel** to **true**, the new postback is canceled. |
| [Sys.WebForms.PageRequestManager beginRequest Event](http://msdn.microsoft.com/en-us/library/bb397432.aspx) | Raised before an asynchronous postback starts and the postback is sent to the server. If there is a postback already processing, it is stopped (by using the [abortPostBack](http://msdn.microsoft.com/en-us/library/bb397526.aspx) method). You can use this event to set request headers or to begin an animation on the page to indicate that the request is in process.  The **beginRequest** event takes an *eventargs* parameter, which is a [Sys.WebForms.BeginRequestEventArgs](http://msdn.microsoft.com/en-us/library/bb384003.aspx) object. This object makes available the element that caused the postback and the underlying request object. |
| [Sys.WebForms.PageRequestManager pageLoading Event](http://msdn.microsoft.com/en-us/library/bb383832.aspx) | Raised after the response from the server to an asynchronous postback is received, but before any content on the page is updated. You can use this event to provide a custom transition effect for updated content.  The **pageLoading** event takes an *eventargs* parameter, which is an [Sys.WebForms.PageLoadingEventArgs](http://msdn.microsoft.com/en-us/library/bb310960.aspx) object. This object makes available information about what panels will be deleted and updated as a result of the most recent asynchronous postback. |
| [Sys.WebForms.PageRequestManager pageLoaded Event](http://msdn.microsoft.com/en-us/library/bb397523.aspx) | Raised after all content on the page is refreshed, as a result of either a synchronous or an asynchronous postback. For synchronous postbacks, panels can only be created, but for asynchronous postbacks, panels can be both created and updated. You can use this event to manage a custom transition effect for updated content.  The **pageLoaded** event takes an *eventargs* parameter, which is an [Sys.WebForms.PageLoadedEventArgs](http://msdn.microsoft.com/en-us/library/bb397476.aspx) object. This object makes available information about which panels were updated and created in the most recent postback. |
| [Sys.WebForms.PageRequestManager endRequest Event](http://msdn.microsoft.com/en-us/library/bb383810.aspx) | Raised after the response for an asynchronous postback is processed and the page is updated, or during the processing of the response if there is an error. If an error occurs, the page is not updated. Use this event to provide customized error notification to users or to log errors.  The **endRequest** event takes an *eventargs* parameter, which is a [Sys.WebForms.EndRequestEventArgs](http://msdn.microsoft.com/en-us/library/bb397499.aspx) object. This object makes available information about errors that have occurred and whether the error was handled. It also makes available the response object. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Event Order Example

The following example shows the client events that are raised on a page that includes two [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) controls, one nested inside the other. You will notice a difference between clicking the button in the parent panel and clicking the button in the nested panel. The button in the parent panel causes the parent panel to be updated and the nested panel to be deleted and recreated. The button in the nested panel causes just the nested panel to be updated.

JavaScript

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl92_ctl00_ctl01');" \o "Copy Code)

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

</script>

<html >

<head runat="server">

<title>Client Event Example</title>

<style type="text/css">

#OuterPanel { width: 600px; height: 200px; border: 2px solid blue; }

#NestedPanel { width: 596px; height: 60px; border: 2px solid green;

margin-left:5 px; margin-right:5px; margin-bottom:5px;}

</style>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:ScriptManager ID="ScriptManager1" runat="server">

<Scripts>

<asp:ScriptReference Path="ClientEventTest.js" />

</Scripts>

</asp:ScriptManager>

<asp:UpdatePanel ID="OuterPanel" UpdateMode="Conditional" runat="server">

<ContentTemplate>

Postbacks from inside the outer panel and inner panel are

asynchronous postbacks. PRM = Sys.WebForms.PageRequestManager. APP = Sys.Application.

<br /><br />

<asp:Button ID="OPButton1" Text="Outer Panel Button" runat="server" />

Last updated on

<%= DateTime.Now.ToString() %>

<br /><br />

<asp:UpdatePanel ID="NestedPanel" UpdateMode="Conditional" runat="server">

<ContentTemplate>

<asp:Button ID="NPButton1" Text="Nested Panel 1 Button" runat="server" />

Last updated on

<%= DateTime.Now.ToString() %>

<br />

</ContentTemplate>

</asp:UpdatePanel>

</ContentTemplate>

</asp:UpdatePanel>

<input type="button" onclick="Clear();" value="Clear" />

<asp:Button ID="FullPostBack" runat="server" Text="Full Postback" />

<a href="http://www.microsoft.com">Test Window Unload</a>

<br />

<span id="ClientEvents"></span>

</div>

</form>

</body>

</html>

JavaScript

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl92_ctl00_ctl02');" \o "Copy Code)

// Hook up Application event handlers.

var app = Sys.Application;

app.add\_load(ApplicationLoad);

app.add\_init(ApplicationInit);

app.add\_disposing(ApplicationDisposing);

app.add\_unload(ApplicationUnload);

// Application event handlers for component developers.

function ApplicationInit(sender) {

var prm = Sys.WebForms.PageRequestManager.getInstance();

if (!prm.get\_isInAsyncPostBack())

{

prm.add\_initializeRequest(InitializeRequest);

prm.add\_beginRequest(BeginRequest);

prm.add\_pageLoading(PageLoading);

prm.add\_pageLoaded(PageLoaded);

prm.add\_endRequest(EndRequest);

}

$get('ClientEvents').innerHTML += "APP:: Application init. <br/>";

}

function ApplicationLoad(sender, args) {

$get('ClientEvents').innerHTML += "APP:: Application load. ";

$get('ClientEvents').innerHTML += "(isPartialLoad = " + args.get\_isPartialLoad() + ")<br/>";

}

function ApplicationUnload(sender) {

alert('APP:: Application unload.');

}

function ApplicationDisposing(sender) {

$get('ClientEvents').innerHTML += "APP:: Application disposing. <br/>";

}

// Application event handlers for page developers.

function pageLoad() {

$get('ClientEvents').innerHTML += "PAGE:: Load.<br/>";

}

function pageUnload() {

alert('Page:: Page unload.');

}

// PageRequestManager event handlers.

function InitializeRequest(sender, args) {

$get('ClientEvents').innerHTML += "<hr/>";

$get('ClientEvents').innerHTML += "PRM:: Initializing async request.<br/>";

}

function BeginRequest(sender, args) {

$get('ClientEvents').innerHTML += "PRM:: Begin processing async request.<br/>";

}

function PageLoading(sender, args) {

$get('ClientEvents').innerHTML += "PRM:: Loading results of async request.<br/>";

var updatedPanels = printArray("PanelsUpdating", args.get\_panelsUpdating());

var deletedPanels = printArray("PanelsDeleting", args.get\_panelsDeleting());

var message = "-->" + updatedPanels + "<br/>-->" + deletedPanels + "<br/>";

document.getElementById("ClientEvents").innerHTML += message;

}

function PageLoaded(sender, args) {

$get('ClientEvents').innerHTML += "PRM:: Finished loading results of async request.<br/>";

var updatedPanels = printArray("PanelsUpdated", args.get\_panelsUpdated());

var createdPanels = printArray("PaneslCreated", args.get\_panelsCreated());

var message = "-->" + updatedPanels + "<br/>-->" + createdPanels + "<br/>";

document.getElementById("ClientEvents").innerHTML += message;

}

function EndRequest(sender, args) {

$get('ClientEvents').innerHTML += "PRM:: End of async request.<br/>";

}

// Helper functions.

function Clear()

{

$get('ClientEvents').innerHTML = "";

}

function printArray(name, arr)

{

var panels = name + '=' + arr.length;

if(arr.length > 0)

{

panels += "(";

for(var i = 0; i < arr.length; i++)

{

panels += arr[i].id + ',';

}

panels = panels.substring(0, panels.length - 1);

panels += ")";

}

return panels;

}

http://i.msdn.microsoft.com/Global/Images/clear.gif Event Order for Common Scenarios

The order of events depends on what controls are used on the page and what type of request is made (initial request, postback, or asynchronous postback). This section describes the order of events for several common scenarios.

**Initial Request**

During the initial request for the page, a limited number of client events are raised. Assume the following scenario for the initial request:

* The page contains a [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control, and the control's [SupportsPartialRendering](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.supportspartialrendering.aspx) and [EnablePartialRendering](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.enablepartialrendering.aspx) property are both **true**.
* The request is a GET request.
* A response from the server is returned successfully.

The following client events occur, in this order:

1. The initial request is sent to the server.
2. The response is received by the client.
3. The **Application** instance raises the **init** event.
4. The **Application** instance raises the **load** event.

The **init** event of the **Application** instance is raised only one time for the life of the page in the browser. It is not raised for subsequent asynchronous postbacks. During an initial request, no **PageRequestManager** events are raised.

**Asynchronous Postback**

An asynchronous postback sends some page data to the server, receives a response, and updates a part of the page. Assume the following scenario for an asynchronous postback:

* The page contains a [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control, and the control's [SupportsPartialRendering](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.supportspartialrendering.aspx) and [EnablePartialRendering](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.enablepartialrendering.aspx) properties are both **true**.
* The page contains an [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) control, and the control's [ChildrenAsTriggers](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.childrenastriggers.aspx) property is **true**.
* A button inside the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) control initiates an asynchronous postback.
* A response from the server is returned successfully.

The following client events occur, in this order:

1. The button inside the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) is clicked, which initiates an asynchronous postback.
2. The **PageRequestManager** instance raises the **initializeRequest** event.
3. The **PageRequestManager** instance raises the **beginRequest** event.
4. The request is sent to the server.
5. The response is received by the client.
6. The **PageRequestManager** instance raises the **pageLoading** event.
7. The **PageRequestManager** instance raises the **pageLoaded** event.
8. The **Application** instance raises the **load** event.
9. The **PageRequestManager** instance raises the **endRequest** event.

For more information, see [Working with PageRequestManager Events](http://msdn.microsoft.com/en-us/library/bb398976.aspx).

Note that the **load** event of the **Application** instance is raised after the **PageRequestManager** **pageLoaded** event and before its **endRequest** event.

**Multiple Asynchronous Postbacks**

Multiple asynchronous postbacks can occur when users initiate a new request before a previously initiated request has finished processing on the server or in the browser. Assume the following scenario for multiple asynchronous postbacks:

* The page contains a [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control and the control's [SupportsPartialRendering](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.supportspartialrendering.aspx) and [EnablePartialRendering](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.enablepartialrendering.aspx) property are both **true**.
* The page contains an [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) control.
* A button inside the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) control that initiates an asynchronous postback is clicked two times. The second click occurs while the server is processing the request initiated by the first click.
* A response to the first request is returned successfully from the server.

The following client events occur, in this order:

1. A button inside the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) is clicked, which initiates an asynchronous postback.
2. The **PageRequestManager** instance raises the **initializeRequest** event.
3. The **PageRequestManager** instance raises the **beginRequest** event.
4. The request is sent to the server.
5. The button is clicked again, which initiates a second asynchronous postback.
6. The **PageRequestManager** instance raises the **initializeRequest** event for the second button click.
7. The **PageRequestManager** instance raises the **endRequest** event for the first button click.
8. The **PageRequestManager** instance raises the **beginRequest** event for the second button click.
9. The request initiated by the second click is sent to the server.
10. The response is received for the second click.
11. The **PageRequestManager** instance raises the **pageLoading** event.
12. The **PageRequestManager** instance raises the **pageLoaded** event.
13. The **Application** instance raises the **load** event.
14. The **PageRequestManager** instance raises the **endRequest** event.

The default behavior of asynchronous postbacks is that the most recent asynchronous postback takes precedence. If two asynchronous postbacks occur in sequence, and if the first postback is still being processed in the browser, the first postback is canceled. If the first postback has been sent to the server, the server processes the second request when it arrives and does not return the first request. For information about how to give precedence to a specific asynchronous postback, see [Giving Precedence to a Specific Asynchronous Postback](http://msdn.microsoft.com/en-us/library/bb386456.aspx).

**Browsing Away from a Page**

When the user browses away from a page, the current page is unloaded from the browser and you can handle the **unload** event to free resources. Assume the following scenario for browsing away from a page:

* The page contains a [ScriptManager](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx) control and the control's [SupportsPartialRendering](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.supportspartialrendering.aspx) and [EnablePartialRendering](http://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.enablepartialrendering.aspx) property are both **true**.
* The target page exists.

The following client events are raised, in this order:

1. The request for new page is initiated.
2. The response for the new page is received by the browser.
3. The **Application** instance raises the **unload** event.
4. The new page is displayed.

If there is an error in the request for the new page, the **unload** event is still raised, but the new page is not displayed.

http://i.msdn.microsoft.com/Global/Images/clear.gif