**ASP.NET Page Life-Cycle**

When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. It is important for you to understand the page life cycle so that you can write code at the appropriate life-cycle stage for the effect you intend.

In general terms, the page goes through the stages outlined in the following table :

|  |  |
| --- | --- |
| Stage | Description |
| Page request | The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page. |
| Start | In the start stage, page properties such as [Request](http://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) and [Response](http://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) are set. At this stage, the page also determines whether the request is a postback or a new request and sets the [IsPostBack](http://msdn.microsoft.com/en-us/library/system.web.ui.page.ispostback.aspx) property. The page also sets the [UICulture](http://msdn.microsoft.com/en-us/library/system.web.ui.page.uiculture.aspx) property. |
| Initialization | During page initialization, controls on the page are available and each control's [UniqueID](http://msdn.microsoft.com/en-us/library/system.web.ui.control.uniqueid.aspx) property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state. |
| Load | During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state. |
| Postback event handling | If the request is a postback, control event handlers are called. After that, the [Validate](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.basevalidator.validate.aspx) method of all validator controls is called, which sets the [IsValid](http://msdn.microsoft.com/en-us/library/system.web.ui.ivalidator.isvalid.aspx) property of individual validator controls and of the page. (There is an exception to this sequence: the handler for the event that caused validation is called after validation.) |
| Rendering | Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the [Render](http://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method for each control, providing a text writer that writes its output to the [OutputStream](http://msdn.microsoft.com/en-us/library/system.web.httpresponse.outputstream.aspx) object of the page's [Response](http://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) property. |
| Unload | The [Unload](http://msdn.microsoft.com/en-us/library/system.web.ui.control.unload.aspx) event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as [Response](http://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) and [Request](http://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) are unloaded and cleanup is performed. |

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 directive is set to true, page events are automatically bound to methods that use the naming convention of Page\_event, such as Page\_Load and Page\_Init

**Working with Events**

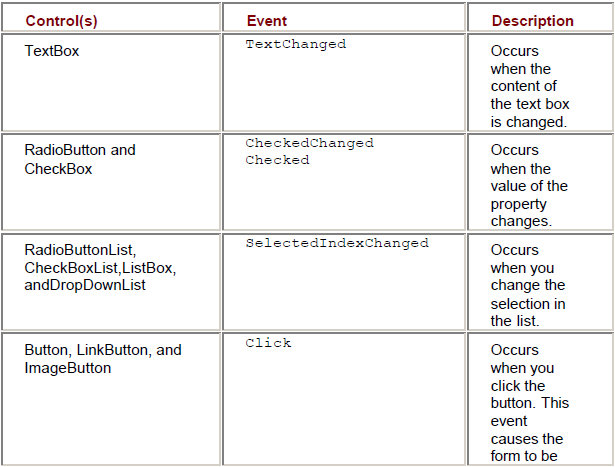
A Web Forms application provides fast, dynamic, and user-interactive Web pages. When users interact with different Web controls on a page, events are raised. In the traditional client forms or client-based Web applications, the events are raised and handled on the client side. However, in Web Forms applications, the events are raised either on the client or on the server, but are always handled on the server. ASP.NET server controls support only server-side events, while HTML server controls support both server-side and client-side events.

**Round trips**

Most Web pages require processing on the server. For example, consider an Orders Web page used to receive orders on the Web. When a user enters a value for the quantity of a product to be bought, the page must check on the server to see whether or not the quantity requested is available. This type of dynamic functionality is accomplished by handling server control events. Whenever a user interaction requires some kind of processing on the server, the Web Forms page is submitted to the server, processed, and then returned to the browser (client). This sequence is called a round trip.



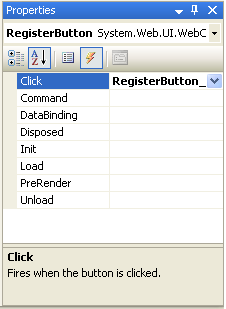
Most of the user interactions with the server controls result in round trips. Because a round trip involves sending the Web Forms page to the server and then displaying the processed form in the browser, the server control events affect the response time in the form. Therefore the number of events available in Web Forms server controls is limited, usually to Click events. The events that occur quite often, such as the OnMouseOver event, are not supported by server controls. However, some server controls support events that occur when the control's value changes,The most popular Events are :



By default, only the Click event of the Button, LinkButton, and ImageButton server controls causes the form to be submitted to the server for processing — the form is said to be posted back to the server. The Change events associated with other controls are captured and cached and do not cause the form to be submitted immediately. When the form is posted back (as a result of a button click), all the pending events are raised and processed. No particular sequence exists for processing these Change events, such as TextChanged and CheckChanged on the server. The Click event is processed only after all the Change events are processed.You can set the change events of server controls to result in the form post back to the server. To do so, modify the AutoPostBack property to True.

**Event handlers**

When the events are raised, you need to handle them for processing. The procedures that are executed when an event occurs are called event handlers. You can create any event to any control the supports postback or form submission by choosing event name in the HTML Editor or by browsing the Events list that comes up when pointing the control in design mode as show below , The event handler is a procedure that will be placed automatically in the C# CodeBehind File , you can put your won logic to handle the business assigned to that control



*handler to perform the intended task.*

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

{

}

In this code:

§ **The procedure RegisterButton\_Click** is the event handler for the Click event of the button with ID RegisterButton.

§ **The procedure takes two arguments**. The first argument contains the event sender. An event sender is an object, such as a form or a control, that can generate events. The second argument contains the event data.

**Implementing the events and event handlers**

***Homework*** : Impalement Event Model to Transfer the User from 1.aspx to 2.aspx with displaying querystring value , Use Textbox Control in 1.aspx and use Label in 2.aspx