

# Web Form Fundamentals

## Today You Will Learn

- The Page Class
- Application Events
- ASP.NET Configuration

# The Page Class

- Every web page is a custom class that inherits from `System.Web.UI.Page`.
- The properties of Page Class are:

Property	Description
IsPostBack	This Boolean property indicates whether this is the first time the page is being run (False) or whether the page is being resubmitted in response to a control event, typically with stored view state information (True). You'll usually check this property in the Page.Load event handler to ensure that your initial web page initialization is only performed once.
EnableViewState	When set to False, this overrides the EnableViewState property of the contained controls, thereby ensuring that no controls will maintain state information.
Application	This collection holds information that's shared between all users in your website. For example, you can use the Application collection to count the number of times a page has been visited. You'll learn more in Chapter 8.

# The Page Class

Property	Description
Request	This refers to an <code>HttpRequest</code> object that contains information about the current web request. You can use the <code>HttpRequest</code> object to get information about the user's browser, although you'll probably prefer to leave these details to ASP.NET. You'll use the <code>HttpRequest</code> object to transmit information from one page to another with the query string in Chapter 8.
Response	This refers to an <code>HttpResponse</code> object that represents the response ASP.NET will send to the user's browser. You'll use the <code>HttpResponse</code> object to create cookies in Chapter 8, and you'll see how it allows you to redirect the user to a different web page later in this chapter.
Session	This collection holds information for a single user, so it can be used in different pages. For example, you can use the <code>Session</code> collection to store the items in the current user's shopping basket on an e-commerce website. You'll learn more in Chapter 8.
Cache	This collection allows you to store objects that are time-consuming to create so they can be reused in other pages or for other clients. This technique, when implemented properly, can improve performance of your web pages. Chapter 23 discusses caching in detail.
User	If the user has been authenticated, this property will be initialized with user information. Chapter 19 describes this property in more detail.

# The Page Class

## Sending the User to a New Page

- There are several ways to transfer a user from one page to another.
  - Use an ordinary `<a>` anchor element:  
Click `<a href="newpage.aspx">here</a>` to go to newpage.aspx.
  - Use code:
    - First we need a control that causes the page to be posted back. Use event handler that reacts to the `OnClick` event of a control such as `Button`.
    - When the page is posted back and your event handler runs, you can use the `Redirect` method to send the user to the new page.

`HttpResponse.Redirect()` method

# The Page Class

When you use the `Redirect()` method, ASP.NET immediately stops processing the page and sends a redirect message back to the browser.

You can also send the user to another website using an absolute URL (a URL that starts with `http://`).

```
Response.Redirect("http://www.prosetech.com");
```

- Use the `HttpServerUtility.Transfer()` method:

```
Server.Transfer("newpage.aspx");
```

The advantage of using the `Transfer()` method is the fact that it doesn't involve the browser.

Instead of sending a redirect message back to the browser, ASP.NET simply starts processing the new page as though the user had originally requested that page.

# Application Events

- Application events are used to perform additional processing tasks.
- Basic ASP.NET features like session state and authentication use application events to plug into the ASP.NET processing pipeline.
- To handle application events you need the help of another ingredient: the **global.asax** file.

## The global.asax File

- The global.asax file allows you to write code that responds to global application events.
- These events fire at various points during the lifetime of a web application, including when the application domain is first created.
- To add a global.asax file to an application in Visual Studio, choose **Website ➤ Add New Item**, and select the **Global Application Class** file type. Then, click **OK**.

# Application Events

- It contains event handlers that respond to application events.

```
<%@ Application Language = "C#" %>
<script language = "c#" runat = "server">
protected void Application_EndRequest(object sender, EventArgs e)
{
    Response.Write(" < hr />This page was served at " +
        DateTime.Now.ToString());
}
</script>
```

## Additional Application Events

- Application.EndRequest is only one of more than a dozen events you can respond to in your code.
- To create a different event handler, you simply need to create a subroutine with the defined name.

# Application Events

Event Handling Method	Description
<code>Application_Start()</code>	Occurs when the application starts, which is the first time it receives a request from any user. It doesn't occur on subsequent requests. This event is commonly used to create or cache some initial information that will be reused later.
<code>Application_End()</code>	Occurs when the application is shutting down, generally because the web server is being restarted. You can insert cleanup code here.
<code>Application_BeginRequest()</code>	Occurs with each request the application receives, just before the page code is executed.
<code>Application_EndRequest()</code>	Occurs with each request the application receives, just after the page code is executed.
<code>Session_Start()</code>	Occurs whenever a new user request is received and a session is started. Sessions are discussed in detail in Chapter 8.
<code>Session_End()</code>	Occurs when a session times out or is programmatically ended. This event is only raised if you are using in-process session state storage (the InProc mode, not the StateServer or SQLServer modes).
<code>Application_Error()</code>	Occurs in response to an unhandled error. You can find more information about error handling in Chapter 7.



# ASP.NET Configuration

- Every web application includes a web.config file.

**The ASP.NET configuration files have several key advantages:**

- **They are never locked:** You can update web.config settings at any point, even while your application is running.
- **They are easily accessed and replicated:** Provided you have the appropriate network rights, you can change a web.config file from a remote computer. You can also copy the web.config file and use it to apply identical settings to another application or another web server that runs the same application in a web farm scenario
- **The settings are easy to edit and understand:** The settings in the web.config file are human-readable, which means they can be edited and understood without needing a special configuration tool.

# ASP.NET Configuration

## The web.config File

- The web.config file uses a predefined XML format.
- The entire content of the file is nested in a root `<configuration>` element.

```
<?xml version="1.0" ?>
```

```
<configuration>
```

```
    <appSettings>...</appSettings>
```

```
    <connectionStrings>...</connectionStrings>
```

```
    <system.web>...</system.web>
```

```
</configuration>
```

- Note that the web.config file is case-sensitive, like all XML documents, and starts every setting with a lowercase letter. This means you cannot write `<AppSettings>` instead of `<appSettings>`.

# ASP.NET Configuration

- There are three sections in the web.config file.
  - The `<appSettings>` section allows you to add your own miscellaneous pieces of information.
  - The `<connectionStrings>` section allows you to define the connection information for accessing a database.
  - The `<system.web>` section holds every ASP.NET setting you'll need to configure.

## `<appSettings>`

- When you create a new website, there's just one element in the `<system.web>` section, named `<compilation>`:

# ASP.NET Configuration

```
<?xml version="1.0" ?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0">
    </compilation>
  </system.web>
</configuration>
```

The **<compilation>** element specifies two settings using two attributes:

- **debug:** This attribute tells ASP.NET whether to compile the application in debug mode so you can use Visual Studio's debugging tools.
- **targetFramework:** This attribute tells Visual Studio what version of .NET you're planning to use on your web server.

# ASP.NET Configuration

## Storing Custom Settings in the web.config File

- ASP.NET also allows you to store your own settings in the web.config file, in an element called <appSettings>.

```
<?xml version="1.0" ?>  
<configuration>  
    <appSettings>  
        <!-- Custom application settings go here. -->  
    </appSettings>  
    <system.web>  
        <!-- ASP.NET Configuration sections go here. -->  
    </system.web>  
</configuration>
```

- You can enter custom settings using an <add> element that identifies a unique variable name (key) and the variable contents (value).

# ASP.NET Configuration

```
<appSettings>
```

```
  <add key="DataFilePath" value="e:\NetworkShare\Documents\WebApp\Shared" />
```


```
</appSettings>
```

## Reasons to do special settings in web.config


- To centralize an important setting that needs to be used in many different pages.
- To make it easy to quickly switch between different modes of operation.
- To set some initial values.

ASP.NET is configured, by default, to deny any requests for .config files. That means remote users will not be able to access the file.

# Life Cycle of Page(Events)


Page Event	Typical Use
<a href="#">PreInit</a>	<p>Raised after the start stage is complete and before the initialization stage begins. Use this event for the following:</p> <ul style="list-style-type: none"><li>• Check the <a href="#">IsPostBack</a> property to determine whether this is the first time the page is being processed. The <a href="#">IsCallback</a> and <a href="#">IsCrossPagePostBack</a> properties have also been set at this time.</li><li>• Create or re-create dynamic controls.</li><li>• Set a master page dynamically.</li><li>• Set the <a href="#">Theme</a> property dynamically.</li><li>• Read or set profile property values.</li></ul> <p> <b>Note</b></p> <p>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.</p>
<a href="#">Init</a>	<p>Raised after all controls have been initialized and any skin settings have been applied. The <a href="#">Init</a> event of individual controls occurs before the <a href="#">Init</a> event of the page. Use this event to read or initialize control properties.</p>
<a href="#">InitComplete</a>	<p>Raised at the end of the page's initialization stage. Only one operation takes place between the <a href="#">Init</a> and <a href="#">InitComplete</a> events: tracking of view state changes is turned on. View state tracking enables controls to persist any values that are programmatically added to the <a href="#">ViewState</a> collection. Until view state tracking is turned on, any values added to view state are lost across postbacks. Controls typically turn on view state tracking immediately after they raise their <a href="#">Init</a> event. Use this event to make changes to view state that you want to make sure are persisted after the next postback.</p>
<a href="#">PreLoad</a>	<p>Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the <a href="#">Request</a> instance.</p>

# Life Cycle of Page(Events)

Load	<p>The <a href="#">Page</a> object calls the <a href="#">OnLoad</a> method on the <a href="#">Page</a> object, and then recursively does the same for each child control until the page and all controls are loaded. The <a href="#">Load</a> event of individual controls occurs after the <a href="#">Load</a> event of the page.</p> <p>Use the <a href="#">OnLoad</a> event method to set properties in controls and to establish database connections.</p>
Control events	<p>Use these events to handle specific control events, such as a <a href="#">Button</a> control's <a href="#">Click</a> event or a <a href="#">TextBox</a> control's <a href="#">TextChanged</a> event.</p> <p> <b>Note</b></p> <p>In a postback request, if the page contains validator controls, check the <a href="#">IsValid</a> property of the <a href="#">Page</a> and of individual validation controls before performing any processing.</p>
LoadComplete	<p>Raised at the end of the event-handling stage.</p> <p>Use this event for tasks that require that all other controls on the page be loaded.</p>
PreRender	<p>Raised after the <a href="#">Page</a> object has created all controls that are required in order to render the page, including child controls of composite controls. (To do this, the <a href="#">Page</a> object calls <a href="#">EnsureChildControls</a> for each control and for the page.)</p> <p>The <a href="#">Page</a> object raises the <a href="#">PreRender</a> event on the <a href="#">Page</a> object, and then recursively does the same for each child control. The <a href="#">PreRender</a> event of individual controls occurs after the <a href="#">PreRender</a> event of the page.</p> <p>Use the event to make final changes to the contents of the page or its controls before the rendering stage begins.</p>
PreRenderComplete	<p>Raised after each data bound control whose <a href="#">DataSourceID</a> property is set calls its <a href="#">DataBind</a> method. For more information, see <a href="#">Data Binding Events for Data-Bound Controls</a> later in this topic.</p>



# Life Cycle of Page(Events)

SaveStateComplete	Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback.
Render	<p>This is not an event; instead, at this stage of processing, the <a href="#">Page</a> object calls this method on each control. All ASP.NET Web server controls have a <a href="#">Render</a> method that writes out the control's markup to send to the browser.</p> <p>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 <a href="#">Render</a> method. For more information, see <a href="#">Developing Custom ASP.NET Server Controls</a>.</p> <p>A user control (an .ascx file) automatically incorporates rendering, so you do not need to explicitly render the control in code.</p>
Unload	<p>Raised for each control and then for the page.</p> <p>In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections.</p> <p>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.</p> <p> <b>Note</b></p> <p>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 <b>Response.Write</b> method, the page will throw an exception.</p>

**END OF LECTURE**