**The ASP.NET life cycle could be divided into two groups:**

* Application Life Cycle
* Page Life Cycle

**ASP.NET Application Life Cycle**

The application life cycle has the following stages:

* User makes a request for accessing application resource, a page. Browser sends this request to the web server.
* A unified pipeline receives the first request and the following events take place:
  + An object of the class Application Manager is created.
  + An object of the class Hosting-Environment is created to provide information regarding the resources.
  + Top level items in the application are compiled.
  + Response objects are created. The application objects such as HttpContext, HttpRequest and HttpResponse are created and initialized.
  + An instance of the HttpApplication object is created and assigned to the request.

**ASP.NET Page Life Cycle**

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

The Page class creates a hierarchical tree of all the controls on the page. All the components on the page, except the directives, are part of this control tree. You can see the control tree by adding trace= "true" to the page directive. We will cover page directives and tracing under 'directives' and 'event handling'.

**The page life cycle phases are:**

* Initialization
* Instantiation of the controls on the page
* Restoration and maintenance of the state
* Execution of the event handler codes
* Page rendering

Understanding the page cycle helps in writing codes for making some specific thing happen at any stage of the page life cycle. It also helps in writing custom controls and initializing them at right time, populate their properties with view-state data and run control behavior code.

Following are the different stages of an ASP.NET page:

* **Page request** - When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.
* **Starting of page life cycle** - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.
* **Page initialization** - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.
* **Page load** - At this stage, control properties are set using the view state and control state values.
  + **Validation** - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.
* **Postback event handling** - If the request is a postback (old request), the related event handler is invoked.
* **Page rendering** - At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.
* **Unload** - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

## The Role of View State

View state's purpose in life is simple: it's there to persist state across postbacks. (For an ASP.NET Web page, its state is the property values of the controls that make up its control hierarchy.)

<asp: Label runat="server" ID="lblMessage"

Font-Name="Verdana" Text="Hello, World!"></asp: Label>

<br />

<asp: Button runat="server"

Text="Change Message" ID="btnSubmit"></asp: Button>

<br />

<asp: Button runat="server" Text="Empty Postback"></asp: Button>

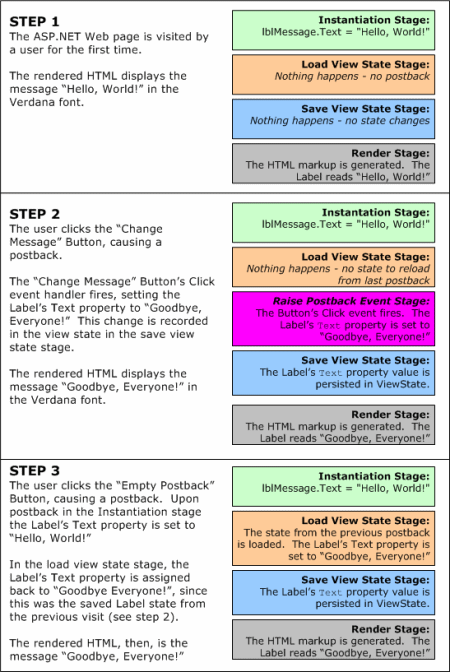
‘And the code-behind class contains the following event handler for the Button's Click event:

private void btnSubmit\_Click(object sender, EventArgs e)

{

lblMessage.Text = "Goodbye, Everyone!";

}



**The Structure of an ASP.NET Page**

The following is a list of the important elements of an ASP.NET page:

* Directives
* Code declaration blocks
* ASP.NET controls
* Code render blocks
* Server-side comments
* Server-side include directives
* Literal text and HTML tags

### Directives

A *directive* controls how an ASP.NET page is compiled. The beginning of a directive is marked with the characters <%@ and the end of a directive is marked with the characters %>. A directive can appear anywhere within a page. By convention, however, a directive typically appears at the top of an ASP.NET page.

Eg:

<%@ Page Language="C#" %>

### Code Declaration Blocks

A *code declaration block* contains all the application logic for your ASP.NET page and all the global variable declarations, subroutines, and functions. It must appear within a <Script Runat="Server">tag.

Eg:

<Script runat="Server">

Sub mySub

...subroutine code

End Sub

</Script>

### ASP.NET Controls

*ASP.NET controls* can be freely interspersed with the text and HTML content of a page. The only requirement is that the controls should appear within a <form Runat="Server"> tag. And, for certain tags such as <span Runat="Server"> and <ASP:Label Runat="Server"/>, this requirement can be ignored without any dire consequences.

One significant limitation of ASP.NET pages is that they can contain only one <form Runat="Server"> tag. This means that you cannot group ASP.NET into multiple forms on a page. If you try, you get an error.

### Code Render Blocks

If you need to execute code within the HTML or text content of your ASP.NET page, you can do so within *code render blocks*. The two types of code render blocks are *inline code* and *inline expressions*. Inline code executes a statement or series of statements. This type of code begins with the characters<% and ends with the characters %>.

Inline expressions, on the other hand, display the value of a variable or method (this type of code is shorthand for Response.Write). Inline expressions begin with the characters <%= and end with the characters %>.

### Server-side Comments

You can add comments to your ASP.NET pages by using *server-side comment* blocks. The beginning of a server-side comment is marked with the characters <%-- and the end of the comment is marked with the characters --%>.

### Server-side Include Directives

You can include a file in an ASP.NET page by using one of the two forms of the *server-side include directive*. If you want to include a file that is located in the same directory or in a subdirectory of the page including the file, you would use the following directive:

<!-- #INCLUDE file="includefile.aspx" -->

### Literal Text and HTML Tags

The final type of element that you can include in an ASP.NET page is HTML content. The static portion of your page is built with plain old HTML tags and text.

**First asp.net sample program**

<% @Page Language="C#" %><!-- code section -->

<script runat="server">

private void convertoupper(object sender, EventArgs e)

{

string str = mytext.Value;

changed\_text.InnerHtml = str.ToUpper();

}

</script>

<!-- Layout -->

<html>

<head>

<title> Change to Upper Case </title>

</head>

<body>

<h3> Conversion to Upper Case </h3>

<form runat="server">

<input runat="server" id="mytext" type="text" />

<input runat="server" id="button1" type="submit" value="Enter..." OnServerClick="convertoupper"/>

<hr />

<h3> Results: </h3>

<span runat="server" id="changed\_text" />

</form>

</body>

</html>

# ASP.NET Validation Controls

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.ASP.NET provides the following validation controls:

* RequiredFieldValidator
* RangeValidator
* CompareValidator

## RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

<asp:RequiredFieldValidator ID="rfvcandidate"

runat="server" ControlToValidate ="ddlcandidate"

ErrorMessage="Please choose a candidate"

InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>

## RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"

ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"

MinimumValue="6" Type="Integer">

</asp:RangeValidator>

## CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

<asp:CompareValidator ID="CompareValidator1" runat="server"

ErrorMessage="CompareValidator">

</asp:CompareValidator>