Agenda

1. *Introduction to Configuration files*
2. *Page settings in web.config*
3. *Custom Errors*
4. *URL Re-Writing*
5. *Tracing*
6. *Using ConfigSource Attribute*
7. *Using Location Section*
8. *HttpApplication class - Global.asax*

Configuration File (web.config)

Web.config is an xml based configuration file containing all configuration settings for the WebForms present in same directory and sub directory.

It can be present in any directory of a web application it overrides and extends the configuration setting of the parent directory web.config:

1. <FrameworkFolder>\Config\Machine.Config
2. <FrameworkFolder>\Config\Web.Config
3. /Web.Config
4. /AppFolder/Web.Config
5. /AppFolder/SubFolder/Web.Config

- Because of the above behavior the directory structure of web app can be planned based on the configuration settings for the various web forms.
- A web.config contains sections marked as **allowDefinition = "MachineToApplication**", These configuration settings must be present only in either machine.config or in web applications root directory

Very Important: Any changes made to "web.config" file of the web application would lead to creation of **New AppDomain** and thus all the Session, Application and Global (Static) data are lost.

Web.config

```
<configuration>
 <appSettings>
  <add key="k1" value="v1" />
  <add key="k2" value="v2" />
  <add key="k3" value= "v3" />
 </appSettings>
</configuration>
```

Default.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
   lblvalue1.Text = ConfigurationManager.AppSettings["k1"];
   lblvalue2.Text = ConfigurationManager.AppSettings["k2"];
   lblvalue3.Text = ConfigurationManager.AppSettings["k3"];
}
```

Note: We have to add reference to System.Configuration (Right click on project -> Add Reference)

Page Settings in web.config and in Page directive

Most of the attributes which we can set for <%@Page can also be set at application level in web.config file.

<pages **validateRequest**="true" **maintainScrollPositionOnPostBack**="true" >

ValidateRequest set to true will not allow tags to be posted and accepted by the page

CSS Improvements in 4.0

The other major improvements in ASP.Net 4.0 are on the way the HTML rendered to the client browser. The rendered HTML will be complaint with latest HTML standards.

Few things will be,

- A new property called controlRenderingCompatibilityVersion is added to the Pages element of configuration files.

  <pages **controlRenderingCompatibilityVersion**="3.5|4.0"/>

- The value "3.5" indicates the controls are rendered in legacy ways. The value "4.0" indicates the control will be rendered with the latest improvements of 4.0 framework.
- Till 3.5 framework, setting "**Enabled=false**" for any control will render disabled attributes in the HTML for the control. According to the latest HTML standard the disabled attribute should be set only for INPUT tags. ASP.Net 4.0 addresses this issue by rendering disabled attribute to only INPUT tags and a CSS class called "aspNetDisabled" to other controls to disable it when controlRenderingCompatibilityVersion property is set to "4.0".

  Below you can find a HTML rendered for a Label and TextBox control when controlRenderingCompatibilityVersion property is set to "4.0".

```
<span id="Label1" class="aspNetDisabled">Label</span>
<input name="TextBox1" type="text" id="TextBox1" disabled="disabled" />
```

- From ASP.Net 2.0, the hidden fields generated for viewstate are packed inside a div tag for XHTML standard. This leads a small white space on the page. ASP.Net 4.0 now includes this div with a css class "aspNetHidden". We can now define the class to prevent the white space by setting border to 0.

  ...

MetaKeywords and MetaDescription in ASPX Page.

Page.**MetaKeywords** = "asp.net,C#";

Page.**MetaDescription** = "This is an asp.net site that hosts asp.net tutorials.";

or

<%@ Page Language="C#" AutoEventWireup="true" **MetaKeywords**="asp.net,C#" **MetaDescription**="This is an asp.net site that hosts asp.net tutorials" CodeFile="Default.aspx.cs" Inherits="_Default" %>

Custom Errors

Must be placed only in Root directory web.config

```
<customErrors  mode="RemoteOnly" defaultRedirect="Errors.aspx">
    <error statusCode="404" redirect="~/"/>
    <error statusCode="500" redirect="~/ServerError.aspx"/>
</customErrors>
```

- **Mode="RemoteOnly"** means redirect the request to error page only if the request comes from remote client.
- **Mode="Off"** for both local and remote users it renders error description and does not redirect.
- **Mode="On"** for both local and remote users it redirects to the error page.

When it redirect to Custom error page, the querystring parameter "**aspxerrorpath**" is appended to the url and its value is the url of the page which actually has an error.

If an unhandled exception is thrown, ASP.NET framework does the following:

1. Raises the **Error** event of the Page i.e. Page_Error event handler method is executed. If the exception is handled here and **Context.ClearError()** is executed, ASP.NET stops further processing of the error.
2. If not, **Application_Error** in Global.asax is executed. If the exception is handled here and Context.ClearError() is executed, ASP.NET stops further processing of error.
3. If not, ASP.NET then uses the <customErrors> configuration section of web.config.

Global.asax

```
protected void Page_Error(object sender, EventArgs e)
{
   Exception ex = Context.Error;
   if (ex is ApplicationException)
   {
      Response.Write("Page_Error: " + ex.Message);
      Context.ClearError(); //Further processing of error is stopped.
   }
}
protected void Application_Error(object sender, EventArgs e)
{
   Exception ex = Context.Error.InnerException;
   if (ex is ApplicationException)
   {
      Response.Write("Application_Error: " + ex.Message);
      Context.ClearError(); //Further processing of error is stopped.
   }
}
```

URL Re-Writing

URL Re-Writing is a process where the path of the URL is rewritten. This is done with the help of **Context.RewritePath(….).** An alternate way is to add **UrlMapping in web.config**

In web.config

```
<urlMappings>
 <add url="~/Monitor.aspx" mappedUrl="~/Computer.aspx? Id=2"/>
 <add url="~/Mouse.aspx" mappedUrl="~/Computer.aspx? Id=3"/>
 <add url="~/Keyboard.aspx" mappedUrl="~/Computer.aspx? Id=1"/>
</urlMappings>
```

Application_BeginRequest

```
protected void Application_BeginRequest(object sender, EventArgs e)
{
   /* Response.Write("URL: " + Request.Url + "<br>");
    Response.Write("Path: " + Request.Path + "<br>");
    Response.Write("FilePath: " + Request.FilePath + "<br>");
    Response.Write("RawUrl: " + Request.RawUrl + "<br>");*/
   string id= Request.QueryString["Id"];
   Context.RewritePath("~/Computer.aspx", "", "id=" + Id);
}
```

Tracing

Application Level Tracing:

```
<trace enabled="true" pageOutput="true"  requestLimit="10"  mostRecent="true" localOnly="true"
traceMode="SortByTime" writeToDiagnosticsTrace="true" />
```

If writeToDiagnosticsTrace = “true”, the output generated by Trace.Write and Trace.Warn will be written to Output Window in VS.NET (only if the application is executed in debug mode using F5)

- To view trace summary from browser we have to use**: http://localhost/AppName/Trace.axd**
- **Page Level Tracing - In ASPX Page: <%@ Page Trace**="true" **TraceMode**="SortByTime" … %>
- **Trace**.**Write**(" … ") & **Trace**.**Warn**(" … "); - Are used for writing to Trace output so that we can find the amount of time a block of code has taken.

List of Page Events Raised during the Page life time: PreInit, Init, InitComplete, PreLoad, Load, LoadComplete, ….. PreRender, PreRenderComplete, SaveState, SaveStateComplete,

Using ConfigSource attribute

It is used for linking the web.config to another file.

1. To the project add a file "AppSettings.xml"

AppSettings.xml

```
<appSettings>
 <add key="k1" value="v11"/>
 <add key="k2" value="v2"/>
</appSettings>
```

2. **In Web.Config:**

web.config

```
<appSettings configSource="AppSettings.xml"/>
```

Advantage: Even if AppSettings.xml is modified the existing AppDomain is not abandoned

Using Location Section

1. In a given directory web.cofig, if we want to provide different configuration settings for a given WebForm, the settings must be enclosed in <location> with path specifying the WebForm

Eg:

```
<configuration>
 <location path="Unsecured.aspx">
        <system.web>
            <authorization>
                <allow users="*"/>
            </authorization>
        </system.web>
 </location>
</configuration>
```

2. If any Configuration Setting is enclosed within <location **allowOverride**="false">, it is locked and cannot be overriden in the subdirectory configuration file.

Note: Location must be nested immediately under <Configuration> tag. (after configSections…)

HttpApplication Class / Global.asax

The root directory of a web application has a special significance and certain content can be present on in that folder. It can have a special file called as **"Global.asax"**. ASP.Net framework uses the content in the **global.asax** and creates a class at runtime which is inherited from **HttpApplication**.

During the lifetime of an application, ASP.NET maintains a pool of Global.asax defined HttpApplication instances. When an application receives an http request, the ASP.Net page framework assigns one of these instances to process that request. That instance is responsible for managing the entire lifetime of the request it is assigned to and the instance can only be reused after the request has been completed when it is returned to the pool.

The instance members in **Global.asax** cannot be used for sharing data across requests but static member can be.

Global.asax can contain the event handlers of HttpApplication object and some other important methods which would execute at various points in a web application.

Some important Events / Methods of HttpApplication Objects:

- **Application_Start:** When the application receives the first request and it is loaded to create the App-Domain.
- **Application_End:** When the web server is shut down and the web applications are unloaded.
- **Session_Start:** When the first request comes from a given client and a session is created.
- **Session_End:** When the session times out or is abandon. This event fires only if the **mode = "InProc"**
- **Application_BeginRequest:** Is executed just before processing of the webform for every request.
- **Application_EndRequest:** After processing the webform, before rendering the output.
- **Application_Error:** Is executed for handling any type of unhandled exception raised in any of the WebForm's of the application and thus can be used for **centralized exception handling**

Example : To get the total number of Active Sessions on the Server at any given instance of the time

/App_Code/Settings.cs

```
public class Settings
{
   public static int SessionCounter;
}
```

Global.asax

```
void Session_Start(object sender, EventArgs e)
{
   Settings.SessionCounter++;
}
void Session_End(object sender, EventArgs e)
{
   Settings.SessionCounter--;
}
```

GetActiveSessionCount.aspx

```
<asp:Label ID="lblActiveSessions" runat="server"></asp:Label>
```

/GetActiveSessionCount.aspx.cs

```
lblActiveSessions.Text = Settings.SessionCounter.ToString();
```