ASP.NET Concepts (Version 1.1)

Also see

- **ADO.NET Concepts**
- **How to Create Resources?**
- **Web Services**

Machine.config – A configuration file that acts as the base configuration for all the web sites and applications that are on the server

Assembly Cache – Stores assemblies that are to be used by several applications (web and windows).

COM – Component Object Model

<% code render block %> - Classic code render blocks are supported by ASP.NET. The code between <% and %> is actually compiled and not interrupted.

Uplevel & downlevel processing – For uplevel processing the ASP.NET server sends DHTML code with HTML version greater than HTML 3.2. This does not require postback to the server on every user events. However, for downlevel processing HTML 3.2 code is produced and requires round trips back to the server for dynamic actions.

ASP.NET directives:

Directive	Description
<%@ Page %>	Used to define page properties such as language, code behind classes.
<%@ Import %>	Used to import namespaces.
<%@ Register %>	Used to register an assembly or control in the Global Cache.
<%@ OutputCache %>	Used to activate output caching on response.

Lifetime of an ASP.NET application:

1. ASP.NET application is created when the first request is send to the server. Before that no ASP.NET code executes. Now **Application** object can be used to store objects with application scope.

2. When the first request is made, a pool of **HttpApplication** instances is created and **Application_Start** event is raised. The **HttpApplication** instances process this request and any subsequent requests until the last instance exits and **Application_End** event is raised.

<u>Notes</u>:

1. ASP.NET processes request *concurrently* and thus allows **Application** object to be accessed by multiple **threads**. When storing objects in application scope, you should either serialize the object using Lock() and Unlock() methods or make the object thread safe.

2. The **init()** and **Dispose()** of **HttpApplication** are called per instance and thus can be used several times between **Application_Start** and **Application_End**. Only these 2 events are shared among all the instances of **HttpApplication** in one ASP.NET application.

Global.asax

- Is located at the root of an ASP.NET application's virtual directory tree.
- May contain implementations of **Application_Start, Application_End, Session_Start,** and **Session_End** to handle application level logic.
- ASP.NET automatically parses and compiles this file into a .NET framework class -- that extends **HttpApplicaton** class-- the first time a request is made to the server for the application.
- Is configured to automatically reject any direct URL request so that external users cannot view or download the code within the application directory.
- List of a few event handlers that can be defined in a Global.asax file:
  - **Application_Start & Application_End:** Only called when the page is opened first time.
  - **Session_Start & Session_End:** Also only called when the first request is made.
  - **Application_BeginRequest & Application_EndRequest:** These methods are called upon each request made to the application server for a particular application.
  - **Application_Error**

- **Static objects, .NET Framework classes,** and COM components can all be defined in Global.asax file using the **<object>** tag. The scope for these objects can be one of the following:
  - **appinstance:** denotes that object in specific to one instance of **HttpApplicaton** class and <u>is not</u> shared.
  - **session:** object is available only during a session.
  - **application:** object is available throughout the lifetime of the application.

Managing Application State:

- Store objects in **application** scope that are modified infrequently and their access should be read-only after their initializations since these objects stored in application scope can accessed by multiple threads concurrently.   For read-only data, this will result in the initialization of the source only once in **Application_Start** and used from that point on.

- Store objects in **session** scope in **Session_Start** event handler (method) using **Session** object when object is specific to individual user. **Session** state features can also be configured in **web.config** file using **<sessionState>** tag. By default ASP.NET stores the **session state** in the **same process** that processes the requests just as ASP code does. But this mechanism can be changed and **session data** can be stored in an **external process** and even on a different machine. To do this:
    1. Start the asp.net state service (**aspnet_state.exe**) [**default port: 42424**]
    2. Set the **mode** attribute of **<sessionState>** in **web.config** to "StateServer".
    3. Set the **stateConnectionString** attribute of **<sessionState>** to the value of machine running **aspnet_state.exe** service.

- Use **client-side cookies** to store session information for each request by instantiating a **HttpCookie** object and adding key/value string using **Add()** method of this instance. Browsers put a 4KB limit cookie files.
- Use **ViewState** to manage state information for each individual controls on a page. The information is stored in **StateBag** class instance in key/value pair. The data should be **request-specific**.

HTTP Handlers & Factories:

- Each incoming request is processed by a class that implements **System.Web.IHttpHandler**.   You can develop your own classes to process/filter requests by creating a class that implement **System.Web.IHttpHandler** and its **ProcessRequest()** method and **IsReusable** property. Factories assign each request to one handler, which processes the request.   **IHTTPHandlerFactory** provides the infrastructure to resolve URL requests to specific **IHttpHandler** instances.
- Custom HTTP handlers and factories cab be configured in **web.config** file using **<httahandlers>** tag.

Caching: Used to retain pages and data between HTTP requests and reuse them without recreating them.   The lifetime of cache is equal to the lifetime of an ASP.NET application, when the application is restarted the cache is recreated. Three kinds of caching that ASP.NET supports are:

- **output:**
    - Caches a dynamic response generated by a request.
    - Good for caching entire pages.

    - Enabled by default, but responses are not cached until explicitly stated to do so. To make responses eligible for caching use either **low-level OutputCache API** or **high-level @OutputCache directive**. There is an **expiration/validation policy** and **public cache visibility**.
    - Pages are not placed in the output cache unless they have a valid expiration or validation policy and public cache visibility.

- **fragment:** Caches portions of response generated by a request. Good for parts of pages or data that is expensive to reconstruct or recreate.
- **data:** Caches arbitrary objects programmatically to be used across requests using a full-featured **caching engine**. **Cache** class is used to add key/value pairs to store objects and their value. For application that need more sophisticated functionality, ASP.NET cache support:
    - **Scavenging –** Cached objects that are not frequently used can be removed from the cache when memory becomes scarce.
    - **Expiration –** Allows the programmer to set a lifetime on the object which can be explicit such as 6:00 PM or relative such 20 minutes from the time of caching. After the expiration time the object is removed and null is returned unless the object is reloaded into the cache.
    - **File and Key dependencies –** Used to place a constraint on an external source or other items in the cache so the cached data contains the updated information related to the external source or other item in the cache.

Configuration:

- A **machine.config configuration file specifies** the configurations for the entire machine. The **web.config** configuration file, found in any directory of an application, can overrides the configuration settings found in **machine.config** file. The changes made to **web.config** file will apply to any child directories found underneath that virtual directory.
- The settings in **web.config** files are parsed and stored in a collection at runtime for each URL. This collection is cached from that point on and ASP.NET watches the changes to file and upon any changes invalidates the cache.
- The presence of a **web.config** file is completely optional. The following order is taken the ASP.NET when looking for application settings.
    - Apply settings from **machine.config**
    - Override the settings for the site with **C:\Inetpub\wwwroot\web.config** file if one is present.
    - Override the setting for the application with **D:\ApplicationDir\web.config** if one is present.

– Override the settings for the dir with **D:\ApplicationDir\ApplicationSubDir\web.config** file if one is present.

- **Sections of web.config:** An xml file that contains sections and handlers for each section.
  – **<configuration />**
    - Root element of each configuration file.
    - Contains the following **sub-sections:**
      1. **Configuration Section Handlers:** .Net frame work classes that implement **IConfigurationSectionHandler** interface.
      2. **Configuration Section Groups:** Optional logical grouping scheme for similar sections.
      3. **Configuration Sections:** Sections in the file that specific section handlers must process.

  – Example:

```
// This is my own section handler
class MyConfigurationSectionHandler : IConfigurationSectionHandler {

      //… define code here.
}
----------------------------------------------------------------------
<configuration>
  <configSections>
    <sectionGroup name=”my.web”> <!--This is optional -->
      <section name=”mymodule”
               type=”MyConfigurationSectionHandler, System.Web” />
    </sectionGroup>
  </configSections>

<my.web>
  <mymodule>
   <add name="CookielessSession"
     type="System.Web.SessionState.CookielessSessionModule,System.Web"
   />
   <add name="OutputCache"
       type="System.Web.Caching.OutputCacheModule,System.Web"
   />
   <add name="Session"
       type="System.Web.SessionState.SessionStateModule,System.Web"
   />
   <add name="WindowsAuthentication"
      type="System.Web.Security.WindowsAuthenticationModule,System.Web"
   />
    …

   <add name="FileAuthorization"
      type="System.Web.Security.FileAuthorizationModule,System.Web" />

 </mymodule>
```

```
</my.web>
</configuration>
```

- By default configuration settings are applied to the current directory and all childs beneath it. You can use **<location path=MyDir/SubDir />** tag to configuration settings to specific child paths. This is important when you want to apply globalization settings to a certain directory pages as in the following code snippet:

```
<configuration>

  <location path="EnglishPages">
    <system.web>
      <globalization
        requestEncoding="iso-8859-1"
        responseEncoding="iso-8859-1"
      />
    </system.web>
  </location>

  <location path="EnglishPages/OneJapanesePage.aspx">
    <system.web>
      <globalization
        requestEncoding="Shift-JIS"
        responseEncoding="Shift-JIS"
      />
    </system.web>
  </location>

</configuration>
```

- **Locking Down Configuration Settings:** You can lock down a group of settings by using the **allowOverride** attribute of **<location>** tag. This will prevent any **inheriting children** from overriding the parent configuration settings.
- **Standard ASP.NET Configuraiton Sections:** The table below lists the default section handlers that are supported by ASP.NET

No.	Section Name	Description
Section Group	**<system.web>**	
1	**<httpModules>**	- Responsible for configuring HTTP modules within an application. - Participate in processing of every request into an application. - Common uses include security and logging.
2	**<httpHandlers>**	- Responsible for mapping incoming URLs to **IHTTPHandler** classes. - Subdirectories do not inherit these settings
3	**<sessionState>**	- Responsible for configuring the session state HTTP module.
4	**<globalization>**	- Responsible for configuring the globalization

		settings of an application.
5	**<compilation>**	- Responsible for all compilation settings used by ASP.NET.
6	**<trace>**	- Responsible for configuring the ASP.NET **trace service**.
7	**<processModel>**	- Responsible for configuring the ASP.NET process model settings on IIS Web Server systems.
8	**<browserCaps>**	- Resposible for controlling the settings of the browser capabilities component.
9	**<clientTarget>**	-
10	**<pages>**	-
11	**<customErrors>**	-
12	**<httpRuntime>**	-
13	**<identity>**	-
14	**<authorization>**	-
15	**<authentication>**	-
16	**<machineKey>**	-
17	**<turst>**	-
18	**<securityPolicy>**	-
19	**<webControls>**	-
20	**<webServices>**	-
21	**<deviceFilters>**	-
22	**<mobileControls>**	-
23	**<appSettings>**	- May contain application specific information such as DB Connection strings, file paths, and remote XML web service URLs.
24	**<runtime>**	
25	**<mscorlib>**	
26	**<startup>**	
27	**<system.runtime.remoting>**	
28	**<system.diagnostics>**	
Section Group	**<system.net>**	
29	**<authenticationModules>**	
30	**<defaultProxy>**	
31	**<connectionManagement>**	
32	**<webRequestModules>**	
33	**<settings>**	
34	**<system.windows.forms>**	

- **Retrieving Configuration Settings:** Use **System.Configuration.ConfigurationSettings** class to retrieve the configurations settings stored in the config file.

Application Deployment:

- **Assembly (Portable Executable DLL):**
  - Classes that are contained in the DLL files.
  - Can span over multiple DLL files.
  - Can use an **assembly** on a computer by deploying it into an **assembly cache**.
  - Assembly cache can be local to an application or global to a computer. Only code intended to be shared by multiple applications should be deployed in **global system assembly cache.** Code used by only individual applications should be deployed into **local assembly cache.**
  - An assembly can be deployed into an application's **local assembly cache** by simply copying over the files using **xcopy** or **ftp** commands to a folder that is marked as '**assembly cache location**'.
  - To deploy a COM component you must run **Regsvr32.exe** from the local machine.
  - The default location for a **local assembly cache** is **\bin** under the root. This directory is also configured denied access to the remote clients to avoid steeling the cache.
  - Example of a simple ASP.NET application folder layout:

```
C:\inetpub\wwwroot
    Web.cfg
    Default.aspx

    \bin       <= Application assembly cache directory
         MyPages.dll
          MyBizLogic.dll

     \order
          SubmitOrder.aspx
          OrderFailed.aspx

          \img
          HappyFace.gif
```

  - Eache application is launched from a new CLR construct called **AppDomain** that enables process host to provide security and configuration isolation at runtime.
  - Once a DLL is loaded, it will stay loaded until the application referencing it is either tored down or recycled.
  - **Process Model Configuration:** You can override the process model configuration settings for an application by changing the settings to **<processModel>** section of **Machine.config** in your application's **web.config** file
  - ASP.NET process can be recycled **2 ways, reative and proactive. Reactive recycling** takes place when a **process misbeaves** or is unable to serve request. **Proactive recycling** takes place periodically even if the **process is healthy**.
  - **Process Model Events** can be **logged** using **logLevel** attribute of **<processModel>.** The levels are **All, None, Errors.**

- The technique where multiple processes are created for multiple CPUs is called **web gardening.**
- **Error Handling:**
  - **Configuration Errors**
  - **Parser Errors**
  - **Compilation Errors**
  - **Run-time Errors**

  Errors can also be handled in Page_Errors method for individual pages and in Application_Error of Global.asax file for application level errors.
- **Security**
  - **Impersanation** means that **COM** objects can execute code under a different user name.
  - **<authentication>** tab
  - **Windows-based Authentication:**
    1. **System.Security.Principal.WindowsPrincipal**
    2. **WindowsAuthentication_OnAuthenticate** event of **Global.asax**
    3. **System.Security.Principal.IPrincipal**
  - **Form-based Authentication:**
    1. **<authentication><form/></authentication>**
    2. **System.Web.Security.FormsAuthenticationTicket**
    3. **FormsAuthentication.Authenticate**
  - **Authorizing Users and Roles**
    1. **POST and GET** can be restricred
    2. * - all users, ?- anonymous users.
  - **User Account Impersonate**
    1. ASP.NET doesn't doe impersonation per request however, it can be configured to do so by setting the following: **<identity impersonate="true"/>**
    2. Since ASP.NET does dynamic compilation, enabling impersonation requires that read/write access is allowed on **application's CodeGen directory** and on **%Windir%\assembly, global cache.**
  - **Security & Web Services:**
- **Inernationalization**
  - Default encoding is **Unicode**
  - **ResponseEncoding** attribute to set other encodings
  - **CultureInfo, RegionInfo**
  - **Localization:**
    1. **Copy** and **translate** the page to the target language. Use the **Culture** property

        2. Use controls to find the the culture settings on the main page.
    - **Resource Files**
        1. **ResourceManager**
        2. Files can be “**loose**” or part of an **assembly**.
        3. **ResourceWriter, Resgen.exe (INPUT: key==value** or and **XML file in .resx** format**, OUTPUT: .resources** files.**)**.
        4. To include a **.resources** file into an assembly, use the related **compiler switch** or **AL.exe** tool. Assemblies containing only **.resrouces** files are called **satellite assemblies.**

- **Tracing**
    - **Page-level Tracing: TraceContext, Trace.Warn(), Trace.Write()**
    - **Application-level Tracing: <trace>** in **web.config**
    - Trace Statistics are stroed in **trace.axd** file.

- **Debugging**
    - symbol files **(.pdb files) –** Map the binary code to source code lines
    - **<compilation debug=”true”/>** in web.config
    - When debugging is enabled and a request is sent to the page, an **Aspnet_wp.exe** process is created and application is loaded into the memory.
    - .NET Framework debugger, **DbgClr.exe**
- **Performance Tips**
    - Disable Session State when not in use
    - Choose your Session state provider carefully: in-process, out-process, out-process in SQLServer
    - Avoide extensive round trips to the server
    - Use server controls sparingly and appropriately
    - Avoid Excessive server control in view state
    - Use HttpResponse.Write instead of Response.Write unless where not much contcatenation is required
    - Do not rely heavily on exceptions
    - Port call-intesive COM components to manage code
    - Use SQL Stored Procedures for data access
    - Use SQLDataReader for fast-forward read-only data cursor.
    - Cache data and output wherever possible.
    - Enable web gardening for multiprocessor computers
    - Disable Debug mode.
    - Use the Peformance tool to monitor ASP.net application.