**Features of Core web Application**

1. **Web.Config File**  
     
   Configuration file is used to manage various settings that define a website. The settings are stored in XML files that are separate from your application code. In this way you can configure settings independently from your code. Generally a website contains a single Web.config file stored inside the application root directory. However there can be many configuration files that manage settings at various levels within an application.

**Usage of configuration file**

* ASP.NET Configuration system is used to describe the properties and behaviours of various aspects of ASP.NET applications.
* Configuration files help you to manage the many settings related to your website. Each file is an XML file (with the extension .config) that contains a set of configuration elements. Configuration information is stored in XML-based text files.

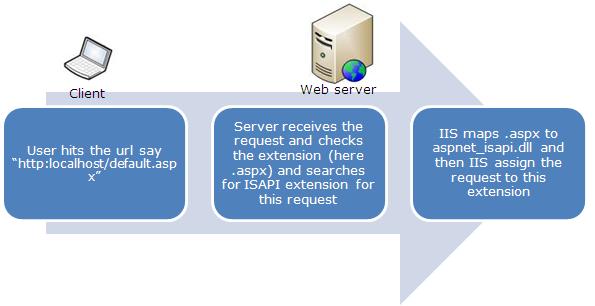
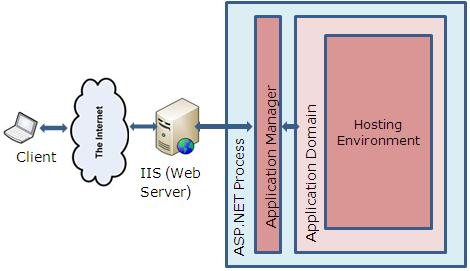
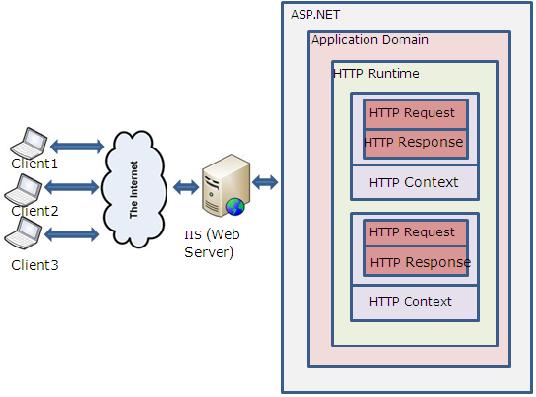
**Benefits of XML-based Configuration files**

* ASP.NET Configuration system is extensible and application specific information can be stored and retrieved easily. It is human readable.
* You need not restart the web server when the settings are changed in configuration file. ASP.NET automatically detects the changes and applies them to the running ASP.NET application.
* You can use any standard text editor or XML parser to create and edit ASP.NET configuration files.

**What Web.config file contains?**  
  
There are number of important settings that can be stored in the configuration file. Some of the most frequently used configurations, stored conveniently inside Web.config file are:

* Database connections
* Caching settings
* Session States
* Error Handling
* Security

|  |
| --- |
| * + 1. **Global.asax file** |
| "The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events raised by ASP.NET." The Global.asax file is parsed and dynamically compiled by ASP.NET into a .NET Framework class and the first time any resource or URL within its application namespace is activated or requested. Whenever the application is requested for the first time, the Global.asax file is parsed and compiled to a class that extends the HttpApplication class. When the Global.asax file changes, the framework reboots the application and the Application\_OnStart event is fired once again when the next request comes in. Note that the Global.asax file does not need recompilation if no changes have been made to it. There can be only one Global.asax file per application and it should be located in the application's root directory only. |
| **Events in the Global.asax file** |
| The following are some of the important events and their purpose in the Global.asax file.   * 1. **Application\_Init:** The Application\_Init event is fired when an application initializes the first time.   2. **Application\_Start:** The Application\_Start event is fired the first time when an application starts.   3. **Session\_Start:** The Session\_Start event is fired the first time when a user’s session is started. This typically contains for session initialization logic code.   4. **Application\_BeginRequest:** The Application\_BeginRequest event is fired each time a new request comes in.   5. **Application\_EndRequest:** The Application\_EndRequest event is fired when the application terminates.   6. **Application\_AuthenticateRequest:** The Application\_AuthenticateRequest event indicates that a request is ready to be authenticated. If you are using Forms Authentication, this event can be used to check for the user's roles and rights.   7. **Application\_Error:** The Application\_Error event is fired when an unhandled error occurs within the application.   8. **Session\_End:** The Session\_End Event is fired whenever a single user Session ends or times out.   9. **Application\_End:** The Application\_End event is last event of its kind that is fired when the application ends or times out. It typically contains application cleanup logic. |

* + 1. **Using Application State**  
         
       Application State is a state management technique. Application State is stored in the memory of the server and is faster than storing and retrieving information in a database. Session sate is specific for a single user session, but Application State is for all users and sessions. Application State does not have a default expiration period. When we close the worker process the application object will be lost. Technically the data is shared amongst users by a HTTPApplcationState class and the data can be stored here in a key/value pair. It can also be accessed using the application property of the HTTPContext class.  
         
       **Application State Life Cycle**  
         
       **Step 1 :**When the Browser sends a request to the web server and the server receives the the request it first checks the extension to determine whether or not it is ISAPI because this request can only be handled by the ISAPI extension; if the extension is different then the request is handled by the server itself.  
         
         
         
       **Step 2 :** After receiving the request the Application Manager creates an application domain. In the application domain an instance of the class HostingEnvironment is created that provides access to information about all application resources.   
         
         
         
       **Step 3 :** After creating the application domain, ASP.NET initializes the basic objects as HTTPContext, HTTPRequest and HTTPResponse. HTTPContext holds objects to the specific application request as HTTPRequest and HTTPResponse.HTTPRequest contains all the information regarding the current request like cookies, browser information and so on and the HTTPResponse contains the response that is sent to the client.  
         
       **Step 4 :**Here all the basic objects are being initialized and the application is being started with the creation of the HTTPApplication class.  
         
         
         
       **Step 5 :** Then events are executed by the HTTPApplication class for any specific requirement. These events are of Global asax.file.
    2. **Working with User session**

1. ***Disconnect a User from a Session***
2. In Remote Desktop Services Manager, in the left pane, click the Remote Desktop Session Host (RD Session Host) server on which the user session is running.
3. To disconnect the user from the session, do either of the following:
   * On the **Users** tab, right-click the user session, and then click **Disconnect**.
   * On the **Sessions** tab, right-click the user session, and then click **Disconnect**.
4. Click **OK** to disconnect the user from the session.
   1. ***Connect to a User Session***
5. In Remote Desktop Services Manager, in the left pane, click the Remote Desktop Session Host (RD Session Host) server on which the user session is running.
6. To connect to the user session, do either of the following:
   * On the **Users** tab, right-click the user session, and then click **Connect**.
   * On the **Sessions** tab, right-click the user session, and then click **Connect**.
7. Click **OK** to connect to the user session.
   1. ***Log Off a User from a Session***
8. In Remote Desktop Services Manager, in the left pane, click the Remote Desktop Session Host (RD Session Host) server on which the user session is running.
9. On the **Users** tab, right-click the user that you want to log off, and then click **Log Off**.
10. Click **OK** to log off the user from the session.

**5. Using Cache objects**

Caching is one of the most interesting concepts and operations in ASP.NET. If you can handle it, you can run any web application by applying the caching concept depending on the requirements.  
  
Currently a majority of websites/portals (or I can say simply web pages) are dynamic. In very common words we can define dynamic pages as including the following:

- Pages that directly interact with people  
- Communication (on page)  
- Any media content  
- Any type of graphic interaction

***There are 3 types of caching***

1. **Page Caching:** To cache an entire page's output we need to specify a directive at the top of our page, this directive is the @ OutputCache.  
     
   Let's figure out a simple demo of it.

<%@ OutputCache Duration = 5 VaryByParam = "ID" %>

Here, in that statement **Duration** and **VarByParam** are the two attributes of the **OutputCache ………**directive.

* **Duration Attribute**: This attributes represents the time in seconds of how long the output cache should be stored in memory. After the defined duration the content stored in the memory will be cleared automatically.
* **VarByParam Attribute**: This is the most important attributes; you can't afford to miss that in the OutputCache directory statement. It generally defines the query string parameters to vary the cache (in memory).

1. **Fragment caching**

In some scenarios we only need to cache only a segment of a page. For example a contact us page in a main page will be the same for all the users and for that there is no need to cache the entire page.

So for that we prefer to use fragment caching option. For example:

1. <%@ OutputCache Duration = 10 VaryByParam = "None" %>
2. **Data Caching**  
   Data caching is slightly different from the 2 other caching types. It's much more interesting to see how data caching actually works.  
     
   As we know in C# everything is about classes and objects. So ASP.NET supports data caching by treating them as small sets of objects. We can store objects in memory very easily and use them depending on our functionality and needs, anywhere across the page.   
     
   Insertion of a string value in the cache as:
3. Cache["Website"] = "CSharpCorner";

## Using Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With ASP, you can both create and retrieve cookie values.

## *How to Create a Cookie?*

The "Response.Cookies" command is used to create cookies.

**Note:** The Response.Cookies command must appear BEFORE the <html> tag.

In the example below, we will create a cookie named "firstname" and assign the value "Alex" to it:

<%  
Response.Cookies("firstname")="Alex"  
%>

It is also possible to assign properties to a cookie, like setting a date when the cookie should expire:

<%  
Response.Cookies("firstname")="Alex"  
Response.Cookies("firstname").Expires=#May 10,2012#  
%>

***b. How to Retrieve a Cookie Value?***

The "Request.Cookies" command is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "firstname" and display it on a page:

<%  
fname=Request.Cookies("firstname")  
response.write("Firstname=" & fname)  
%>

**Output:** Firstname=Alex