**ASP.NET 4.**

* Output cache extensibility
* Session state compression
* View state mode for individual control
* *Page.MetaKeyword* and *Page.MetaDescription* properties
* *Response.RedirectPermanent* method
* Routing in ASP.NET
* Increase the URL character length
* New syntax for Html Encode
* Predictable Client IDs
* *Web.config* file refactoring
* Auto-Start ASP.NET applications
* Improvements on Microsoft Ajax Library

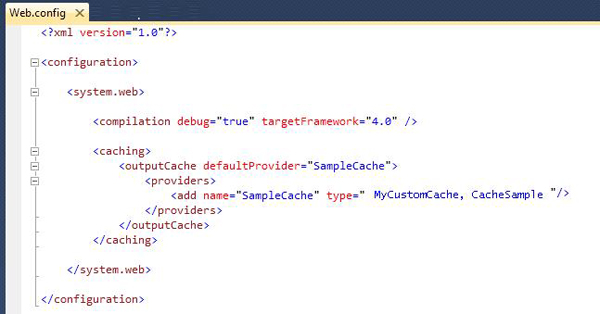
I’ll describe the details of each of these features in the following sections.

**Output Cache extensibility**

Output caching, or Page-Level Caching, caches the entire rendered markup of an ASP.NET web page for a specific time-period. This has always been one of the essential features for ASP.NET that is used extensively to increase application performance. However there have been some limitations on the feasible extent of caching, because cached content always had to be stored in-memory.

But with ASP.NET 4.0 developers can extend their caching by using **Output-cache providers**. Developers can now create ‘output-cache providers’ that store the cache contents to any persistence mechanism such as databases, disks, cloud storage and distributed cache engines.

To create a custom output-cache provider, a class which derived from **System.Web.Caching.OutputCacheProvider** has to be created in ASP.NET 4.0. There are four public methods which you have to override in order to provide your own implementation for add, remove, retrieve and update functionality. Also, the output-cache provider has to be registered in the **web.config** file as shown in the following screen capture.



You can also set this custom output-cache provider as your default cache mechanism. So once you add the page cache directives all of your contents will be stored using the custom output-cache provider.

http://www.simple-talk.com/iwritefor/articlefiles/1012-LS2.JPG

Moreover, developers can also dynamically configure which output-cache Provider is used. For example you might want to cache the frequently access pages in the memory for faster access and less frequent pages on disk. By overriding the **GetOutputCacheProviderName()** method you can configure which output cache provider to use for different requests. These additions to the output-cache can enable developers to write extensible and more efficient cache mechanisms to their web application and thereby improve its responsiveness.

**Session State compression**

The ASP.NET session state is a mechanism to maintain session-specific data through subsequent requests. In some instances, you may wish to store your session state data in a session-state server or in Microsoft SQL server. However, these two options require you to store data out of the web application’s worker process. To send across to the relevant sources, (State server or Microsoft SQL Server), session-state data has to be serialized. This can take a significant time if the size of the data to be serialized grows significantly. This will increase the latency of the application.

This latency can be reduced if the size of the data is lessened by compression. ASP.NET 4.0 introduces a new mechanism to compress your session state data for both Session-state server and Microsoft SQL server. Compression can be enabled by setting the **compressionEnable** to true in the **web.config** file. In this example, the session-state data will be serialized/desterilized using **System.IO.Compression.GZipStream.**

<sessionState mode="SqlServer" sqlConnectionString="data source=DB;Initial Catalog=LudmalDB" allowCustomSqlDatabase="true" compressionEnabled="true"/>

With this compression feature, developers can often reduce the time it takes for a web application to respond by reducing the size of session data.

**View State mode for Individual Controls**

View state is a mechanism to maintain page controls’ state on subsequent post backs. ASP.NET stores the view state data for controls that are in the page, even if it’s not necessary. Since the view state data is stored in the pages’ html, the size of the request object will be increased, and make performance worse.

In ASP.NET 4.0, each web control will include a **ViewStateMode** property which lets developers disable view-state by default, and enable it just for the controls for which a persistence of state is required. *ViewStateMode* has the following three values;

* Enabled – enables the view state for this control and any child control.
* Disabled – disable the view state.
* Inherits – this specify the control uses the settings from its parent control.

By setting these values in page controls accordingly, a significant performance improvement can be gained in response-time.

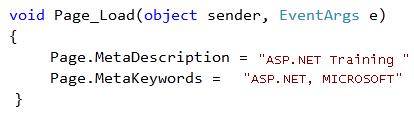
***Page.MetaKeywords* and *Page.MetaDescription* properties**

To increase the relevance of pages in searches, developers should include relevant “keyword” and “description” meta tags in the html <head> section. Unfortunately, it takes some time to add these tags for each and every page, and the alternative of adding these tags programmatically was difficult.

But with ASP.NET 4.0, there are two new properties in the code behind file;

* *Page.MetaDescription* – equivalent to meta name “description”
* *Page.MetaKeywords* – equivalent to meta name “keywords”

This will enable developers to easily and programmatically add the relevant keywords and description.



This will even be useful for Master pages—where you only have to add these properties in the master page. In addition to “keywords” and “description” settings in the code behind, developers can also set these values within the @Page directive.

***Response.RedirectPermanent* Method**

ASP.NET 4.0 has improved SEO (Search-engine Optimization) facilities. Typically developers use **Response.Redirect(string url***)* to handle requests for old URLs. However, this leads to an extra round trip to access the old URLs and so will negatively affect your page-ranking in search-engines.

ASP.NET 4.0 introduces a new **Response.RedirectPermanent(string url)** helper method to be used as HTTP 301 (Moved permanently) to handle requests. This will enable search-engines to index URLs and content efficiently and thus improve the page rankings.

**Routing in ASP.NET**

Routing will let developers serve meaningful URLs to users and map them with the actual physical files. This URL-rewriting mechanism enables developers to write high ranking, search-engine optimized web applications. For example, URL for a page which displays an actual product might look like the following;

http://www.ludmal.net/showproducts.aspx?prodId=24

By using routing the URL will look like the following

http://www.ludmal.net/products/ipod

In this way, the URLs will be more easily remembered by users. It will also significantly improve the search-engine page rankings of the web site.

The following example shows how to implement routing behavior in ASP.NET 4 using new *MapPageRoute* in *Route* class.

public class Global : System.Web.HttpApplication { void Application\_Start(object sender, EventArgs e) { RouteTable.Routes.MapPageRoute("ProductsRoute", "product/{prodId}", "~/products.aspx"); } }

**Increase the URL character length**

In previous versions of ASP.NET, URLs were limited to 260 characters in length. But in ASP.NET 4.0 developers have the option of increasing or decreasing the length of URLs by using the new **maxRequestPathLength** and **maxQueryStringLength**. I’ll illustrate this in an example.

<httpRuntime maxRequestPathLength="360" maxQueryStringLength="1024" />

In previous versions of ASP.NET you were limited to a fixed set of characters but in v4, developers can also validate the invalid characters by specifying values in the **requestPathInvalidChars** attribute.

**New syntax for Html Encode**

Html Encode method encodes a particular string to be displayed in a browser. It is important to encode strings prior it’s rendering in the page, mainly to avoid cross-site script injection (XSS) and HTML injection attacks. However, developers so often forget to call the encode function.

In previous .NET versions, **Server.HtmlEncode()** or **HttpUtility.Encode()** methods has been used for string encoding as shown in the following example.

http://www.simple-talk.com/iwritefor/articlefiles/1012-LS5.JPG

ASP.NET 4.0 introduced new code expression syntax for encoding a particular string. While the syntax will render the output it also encodes the relevant string as shown below. Note “**:**” character after opening tag (“<%”).

http://www.simple-talk.com/iwritefor/articlefiles/1012-LS6.JPG

The new encoding syntax provides an easy and concise way of encoding a particular string.

**Predictable Client IDs**

ASP.NET 4 now supports a new **ClientIDMode** property for server control. This property indicates how the Client ID should be generated to a particular control when they render. Client ID has been an important property of the server controls recently—especially with the success of jQuery and other Ajax scripting technologies. The ClientIDMode property has four values;

* **AutoID** – This renders the output as it was before (example: *ctl00\_ContentPlaceholder1\_ListView1\_ctrl0\_Label1*)
* **Predictable (Default)**– Trims all “ctl00” strings in the Client Id property.
* **Static** – Full control over the Client ID (developer can set the Client Id and it will not be changed after the control renders)
* **Inherit** – Allow control to inherit the behavior from its parent control

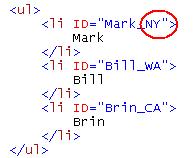
Client ID property can be set in three different ways;

* Directly on individual control
* On the container control. (All the child controls will inherit the settings from parent/container control)
* Page or User Control level using <%@ Page%> or <%@ Control %> directives.
* Directly in the *web.config* file. All the controls within the web application will inherit the settings.

New **ClientIDRowSuffix** property on databound controls also gives a similar functionality when rendering an each data item. Once you set the relevant databound property to ClientIDRowSuffix, the value will be added as a suffix to individual row elements.

http://www.simple-talk.com/iwritefor/articlefiles/1012-LS7.JPG

After the control renders the “State” value will be added as a suffix to each data row element.



**Web.config refactoring**

Over the past few years **web.config** file has grown significantly as ASP.NET has used it for more and more features such as routing, Ajax, IIS 7 and version compatibility. This has made it trickier to maintain even with the Visual Studio environment.

With ASP.NET 4, most of the major elements have been moved to the **machine.config** file. This has enabled developers to maintain a cleaner, less cluttered, *web.config* file. The new **web.config** file is either empty, or includes just the .NET framework version details as shown in the following example.

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

**Auto-Start ASP.NET Applications**

Most application requires initial data load or caching operations to be done before serving the client requests. Typical this happens only when the first user request a page. However, often developers and web administrators write fake requests to keep the application alive to increase the response time. To overcome this issue, ASP.NET 4 introduce new Auto-Start feature. Auto-start feature available with IIS 7.5 and it initialize the ASP.NET application to accept requests.

To configure the Auto-start, you need to configure the “Application pool” worker process by setting the startMode attribute to “AlwaysRunning” in the *applicationHost.config* file. (C:\Windows\System32\inetsrv\config\applicationHost.config)



As soon you save the *applicationHost.config* file the worker process will start and initialize the required application operations before the first user has been served.

**Improvements on Microsoft Ajax Library**

Microsoft Ajax library is client side library which includes high performance server –based user controls and asynchronous page rendering controls. Ajax Library enables developers to easily and quickly write responsive database-driven applications.

There are some significant improvements in the Ajax Library in the ASP.NET 4. of which the most important seem to be...

* **Scrip Loader** – the new script loader control enable developers to load all the required scripts only once, thereby eliminating the unnecessary subsequent requests to the server. It supports the ‘lazy load’ pattern which loads scripts only when necessary, and loads scripts in combination, in order to increase the performance of loading a page. It also supports the *jQuery* script and custom scripts.
* ***JQuery* Integration** – *JQuery* is very popular third party javascript library. ASP.NET 4 extensively supports the integration for *jQuery* by mixing the *jQuery* and Ajax plug-ins seamlessly.
* **Client Data Access** – by using pre-defined client controls inside the Ajax Library, developers can easily build asynchronous data-driven applications. For example client **DataView** control will display one or more records by consuming a WCF service. All the relevant time-consuming operations will be handled by the Ajax library asynchronously.

**Conclusion**

ASP.NET 4 includes plethora of new features which will enable developers to write high performance, search-engine friendly web application quickly. The features I’ve mentioned seem to be the most important of all the new features in ASP.NET 4. By upgrading your existing web applications to up-coming ASP.NET 4, you are likely to see an improvement in performance and search-engine optimization.