Download demo website configuration - 1.34 MB

Introduction

The time you start developing your web application until you finish the application, you will more often use the *Web.config* file not only for securing your application but also for wide range of other purposes which it is intended for. ASP.NET *Web.config* file provides you a flexible way to handle all your requirements at the application level. Despite the simplicity provided by the .NET Framework to work with *web.config*, working with configuration files would definitely be a task until you understand it clearly. This could be one of the main reasons that I started writing this article.

This article would be a quick reference for the professional developers and for those who just started programming in .NET. This article would help them to understand the ASP.NET configuration in an efficient way. The readers may skip the reading section "Authentication, Authorization, Membership Provider, Role Provider and Profile Provider Settings", as most of them are familiar with those particular settings.

Background

In this article, I am going to explain about the complete sections and settings available in the *Web.config* file and how you can configure them to use in the application. In the later section of the article, we will see the .NET classes that are used to work with the configuration files. The contents of the articles are summarized below:

1. *Web.config* sections/settings
2. Reading *Web.config*
3. Writing or manipulating *Web.config*
4. Encrypting the *Web.config* and
5. Creating your own Custom Configuration Sections

Points to be Remembered

ASP.NET *Web.config* allows you to define or revise the configuration settings at the time of developing the application or at the time of deployment or even after deployment. The following are brief points that can be understood about the *Web.config* file:

- *Web.config* files are stored in XML format which makes us easier to work with.
- You can have any number of *Web.config* files for an application. Each *Web.config* applies settings to its own directory and all the child directories below it.
- All the *Web.config* files inherit the root *Web.config* file available at the following location *systemroot\Microsoft.NET\Framework\versionNumber\CONFIG\Web.config* location

- IIS is configured in such a way that it prevents the *Web.config* file access from the browser.
- The changes in *Web.config* don't require the reboot of the web server.

Web.config Settings

Before we start working with configuration settings of ASP.NET, we see the hierarchy of the *Web.config* file.

XML

```xml
<configuration>

        <configSections>
            <sectionGroup>
            </sectionGroup>
        </configSections>

        <system.web>
        </system.web>

        <connectionStrings>
        </connectionStrings>

        <appSettings>
        </appSettings>
        …………………………………………………………………………………………………………
        …………………………………………………………………………………………………………
        …………………………………………………………………………………………………………
        …………………………………………………………………………………………………………
        …………………………………………………………………………………………………………

</configuration>
```

So from the above tree structure, we can understand that the configuration tag is the root element of the *Web.config* file under which it has all the remaining sub elements. Each element can have any number of attributes and child elements which specify the values or settings for the given particular section. To start with, we'll see the working of some of the most general configuration settings in the *Web.config* file.

system.web

In the configuration hierarchy, the most common thing we will work with is the *system.web* section. Now we look at some of the child sections of the *system.web* section of *Web.config* file.

Compilation Settings

If you are using Visual Studio 2010, probably the only available section of *Web.config* file by default is `Compilation` section. If you want to specify the target framework or if you need to add an assembly from the Global Assembly Cache (GAC) or if you want to enable the debugging mode of the application, you can take `Compilation` settings as granted for these tasks. The following code is used to achieve the discussed settings:

XML

```xml
<system.web
    <compilation
                 debug="true" strict="true" explicit="true" batch="true"
                 optimizeCompilations="true" batchTimeout="900"
                 maxBatchSize="1000" maxBatchGeneratedFileSize="1000"
                 numRecompilesBeforeAppRestart="15" defaultLanguage="c#"
                 targetFramework="4.0" assemblyPostProcessorType="">
    <assemblies>
        <add assembly="System, Version=1.0.5000.0, Culture=neutral,
                  PublicKeyToken=b77a5c561934e089"/>
    </assemblies>

</compilation>
</system.web>
```

Under the `assemblies` element, you are supposed to mention the type, version, culture and public key token of the assembly. In order to get the public key token of an assembly, you need to follow the below mentioned steps:

1. Go to Visual Studio tools in the start menu and open the Visual Studio command prompt.
2. In the Visual Studio command prompt, change the directory to the location where the assembly or *.dll* file exists.
3. Use the following command, `sn –T itextsharp.dll`.
4. It generates the public key token of the assembly. You should keep one thing in mind that only public key token is generated only for the assemblies which are strongly signed.

Example

```
C:\WINNT\Microsoft.NET\Framework\v3.5> sn -T itextsharp.dll
Microsoft (R) .NET Framework Strong Name Utility Version 3.5.21022.8
Copyright (c) Microsoft Corporation.  All rights reserved.

Public key token is badfaf3274934e0
```

Explicit and sample attributes are applicable only to VB.NET and C# compiler however ignores these settings.

Page Settings

Ok, by this time, we have got familiar with the *Web.config* file and we have seen the settings of Compilation Sections, now we will see the settings of a page. As an ASP.NET application consists of several number of pages, we can set the general settings of a page like `sessionstate`, `viewstate`, `buffer`, etc., as shown below:

XML

```xml
<pages buffer ="true" styleSheetTheme="" theme ="Acqua"
              masterPageFile ="MasterPage.master"
              enableEventValidation="true">
```

By using the `MasterPageFile` and theme attributes, we can specify the master page and theme for the pages in web application.

Custom Error Settings

The next section of *Web.config* file, we are going to look around is Custom Error settings, by the name itself it is clear that we can configure the settings for the application level errors in these section. Now we will see the description of the `customErrors` section of the *Web.config* from the below mentioned code snippet.

XML

```xml
<customErrors defaultRedirect ="Error.aspx" mode ="Off">
   <error statusCode ="401" redirect ="Unauthorized.aspx"/>
</customErrors>
```

The `customErrors` section consists of `defaultRedirect` and `mode` attributes which specify the default redirect page and the on/off mode respectively.
The subsection of `customErrors` section allows redirecting to specified page depending on the error status code.

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 408 Request Timeout

For a more detailed report of status code list, you can refer to this URL:

- http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Location Settings

If you are working with a major project, probably you might have numerous numbers of folders and sub-folders, at this kind of particular situation, you can have two options to work with. First thing is to have a *Web.config* file for each and every folder(s) and Sub-folder(s) and the second one is to have a single *Web.config* for your entire application. If you use the first approach, then you might be in a smoother way, but what if you have a single *Web.config* and you need to configure the sub-folder or other folder of your application, the right solution is to use the "Location" tag of "system.web" section of *Web.config* file. However you can use this tag in either of the discussed methods.

The following code shows you to work with Location settings:

XML

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

XML

```
<location path ="Uploads">
    <system.web>
    <compilation debug = "false">
    </system.web>
</location>
```

In a similar way, you can configure any kind of available settings for any file/folder using the location tag.

Session State and View State Settings

As we all know, the ASP.NET is stateless and to maintain the state we need to use the available state management techniques of ASP.NET. View state and session state are among them. For complete information about view state and Session State and how to work with, there are some excellent articles in CodeProject, which you can refer here:

- Beginners Introduction to State Management Techniques in ASP.NET
- Exploring Session in ASP.NET

Now we'll see the *Web.config* settings of View State and Session State:
View State can be enabled or disabled by using the following page settings in the *web.config* file.

XML

```
<Pages EnableViewState="false" />
```

Session state settings for different modes are as shown below:

XML

```
<sessionState mode="InProc" />
```

XML

```xml
<sessionState mode="StateServer"
stateConnectionString= "tcpip=Yourservername:42424" />
```

XML

```xml
<sessionState mode="SQLServer" sqlConnectionString="cnn" />
```

HttpHandler Settings

`HttpHandler` is a code that executes when an http request for a specific resource is made to the server. For example, request an *.aspx* page the ASP.NET page handler is executed, similarly if an *.asmx* file is requested, the ASP.NET service handler is executed. An HTTP Handler is a component that handles the ASP.NET requests at a lower level than ASP.NET is capable of handling.

You can create your own custom http handler, register it with IIS and receive notice whenever a request is made. For doing this, you just need to create a class which implements `IHttpHanlder` and then you need to add the following section of configuration settings in the *web.config* file. For this demonstration, I have created a sample `imagehandler` class which displays a JPG image to the browser.You can go through the `imagehandler` class code in the sample download code.

XML

```xml
<httpHandlers>
    <add verb="*" path="*.jpg" type="ImageHandler"/>
    <add verb="*" path="*.gif" type="ImageHandler"/>
</httpHandlers/>
```

HttpModule Settings

`HttpModule` is a class or an assembly that implements the `IHttpModule` interface that handles the application events or user events. You can too create your own custom `HttpModule` by implementing the interface and configure it with ISS. The following settings show the `HttpModules` configuration in the *web.config*.

XML

```xml
<httpModules>
      <add type ="TwCustomHandler.ImageHandler"
           name ="TwCustomHandler"/>
      <remove name ="TwCustomHandler"/>
      <clear />
</httpModules>
```

Authentication, Authorization, Membership Provider, Role Provider and Profile Provider Settings

These settings are directly available in the *web.config* file if you have created the ASP.NET application by using the Visual Studio 2010. I'm not going to elaborate them as there are lot of articles in CodeProject describing the functionality and use of these settings and for further information you can refer to them. Some of the links are here:

- ASP.NET Membership and Role Provider
- Developing custom ASP.NET Membership and Role providers reading users from custom section in the web.config
- ASP.NET Membership
- Membership and Role providers for MySQL
- Custom Membership, Role Providers, Website administration tool, and Role based access to individual files

Authentication Settings

XML

```xml
<authentication mode="Forms">
     <forms cookieless="UseCookies" defaultUrl="HomePage.aspx"
                    loginUrl="UnAuthorized.aspx" protection="All" timeout="30">
      </forms>
</authentication>
```

Authorization Settings

XML

```xml
<authorization
        <allow roles ="Admin"/>
        <deny users ="*"/>
</authorization>
```

Membership Provider Settings

XML

```xml
<membership defaultProvider="Demo_MemberShipProvider">
    <providers>
       <add name="Demo_MemberShipProvider"
            type="System.Web.Security.SqlMembershipProvider"
            connectionStringName="cnn"
            enablePasswordRetrieval="false"
            enablePasswordReset="true"
            requiresQuestionAndAnswer="true"
            applicationName="/"
            requiresUniqueEmail="false"
            passwordFormat="Hashed"
            maxInvalidPasswordAttempts="5"
            minRequiredPasswordLength="5"
            minRequiredNonalphanumericCharacters="0"
           passwordAttemptWindow="10" passwordStrengthRegularExpression="">
    </providers>
</membership>
```

Role Provider Settings

XML

```
<roleManager enabled="true" cacheRolesInCookie="true"
cookieName="TBHROLES" defaultProvider="Demo_RoleProvider">
              <providers>
                  <add connectionStringName="dld_connectionstring"
                  applicationName="/" name="Demo_RoleProvider"
                  type="System.Web.Security.SqlRoleProvider, System.Web,
                  Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
             </providers>
</roleManager>
```

Profile Provider Settings

XML

```
<profile defaultProvider="Demo_ProfileProvider">
    <providers>
    <add name="Demo_ProfileProvider" connectionStringName="cnn"
    applicationName="/" type="System.Web.Profile.SqlProfileProvider,
    System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
    </providers>
    <properties>
    <add name="Name" type="String"/>
    <add name="DateofBirth" type="DateTime"/>
    <add name="Place" type="string"/>
    </properties>
</profile>
```

AppSettings

In the above section, we have seen the settings available in `system.web` tag, now we will see the available settings in `appSettings` section.
`appSettings` element helps us to store the application settings information like connection strings, file paths, URLs, port numbers, custom key value pairs, etc.
The following code snippet shows the example of `appSettings` Section:

XML

```
<appSettings>
    <add key="AppKey" value="APLJI12345AFAFAF89999BDFG"/>
</appSettings>
```

connectionStrings

The most common section of *web.config* file the `connectionStrings` sections allows you to store multiple connection strings that are used in the application. The `connectionStrings` tag consists of child element with attributes name and `connectionstring` which is used to identify the `connectionstring` and the other is used to connect to the database server respectively.

The general `connectionstring` settings are shown below:

XML

```xml
<connectionStrings>
    <add name ="cnn" connectionString ="Initial Catalog = master;
        Data Source =localhost; Integrated Security = true"/>
</connectionStrings>
```

ConfigSections

`ConfigSections` helps you to create your own custom configuration section that can be used with the *web.config* file. We look at this in the later section of the article, for the time being, we can have look at the `configsection` settings. `ConfigSections` should be declared just below the configuration (parent element) otherwise it is going through you an error.

XML

```xml
<configSections>
    <sectionGroup name="pageAppearanceGroup">
      <section
        name="pageAppearance"
        type="PageAppearanceSection"
        allowLocation="true"
        allowDefinition="Everywhere"
      />
 </sectionGroup>
 </configSections>
```

Programmatically Accessing the Web.config File

We can use the C# classes to read and write the values to the *Web.config* file.

Reading appSettings values

The following code is used to read the `appSettings` values from *Web.config* file. You can use either of the methods shown below:

C#

```
//Method 1:
        string key = ConfigurationManager.AppSettings["AppKey"];
        Response.Write(key);

//Method 2:
        Configuration config = WebConfigurationManager.OpenWebConfiguration("~/");
        KeyValueConfigurationElement Appsetting = config.AppSettings.Settings["AppKey"];
        Response.Write(Appsetting.Key + " <br/>" + "Value:" + Appsetting.Value);
```

Reading connectionstring values

The following code is used to read the `connectionstring` values from *Web.config* file. You can use either of the methods shown below:

C#

```
//Method 1:
        string cnn = ConfigurationManager.ConnectionStrings["conn"].ConnectionString;

//Methods 2:
        Configuration config = WebConfigurationManager.OpenWebConfiguration("~/");
        ConnectionStringSettings cnnstring;

        if (config.ConnectionStrings.ConnectionStrings.Count > 0)
        {
            cnnstring = config.ConnectionStrings.ConnectionStrings["conn"];
            if (cnnstring != null)
                Response.Write("ConnectionString:" + cnnstring.ConnectionString);
            else
                Response.Write(" No connection string");
        }
```

Reading configuration section values

The following code is used to read the configuration section values from *Web.config* file. The comments in the code will help you to understand the code:

C#

```
// Intialize System.Configuration object.
Configuration config = WebConfigurationManager.OpenWebConfiguration("~/");
//Get the required section of the web.config file by using configuration object.
CompilationSection compilation =
    (CompilationSection)config.GetSection("system.web/compilation");
//Access the properties of the web.config
Response.Write("Debug:"+compilation.Debug+"<br/>""+
        "Language:"+compilation.DefaultLanguage);
```

Update the configuration section values

The following code is used to read the configuration section values from *Web.config* file:

C#

```
Configuration config = WebConfigurationManager.OpenWebConfiguration("~/");
//Get the required section of the web.config file by using configuration object.
CompilationSection compilation =
    (CompilationSection)config.GetSection("system.web/compilation");
//Update the new values.
compilation.Debug = true;
//save the changes by using Save() method of configuration object.
if (!compilation.SectionInformation.IsLocked)
{
    config.Save();
    Response.Write("New Compilation Debug"+compilation.Debug);
}
else
{
    Response.Write("Could not save configuration.");
}
```

Encrypt Configuration Sections of Web.config File

As we have already discussed that IIS is configured in such a way that it does not serve the *Web.Config* to browser, but even in some such situation to provide more security, you can encrypt some of the sections of *web.config* file. The following code shows you the way to encrypt the sections of *web.config* file:

C#

```
Configuration config = WebConfigurationManager.OpenWebConfiguration
            (Request.ApplicationPath);
ConfigurationSection appSettings = config.GetSection("appSettings");
if (appSettings.SectionInformation.IsProtected)
{
    appSettings.SectionInformation.UnprotectSection();
}
else
{
    appSettings.SectionInformation.ProtectSection("DataProtectionConfigurationProvider");
}
config.Save();
```

Custom Configuration Section in Web.config

I have thought twice before I could put this section of content in this article, as there are a lot of wonderful articles explaining this topic, but just to make this article as complete, I have included this topic too.

Create Custom Configuration Section

The `ConfigurationSection` class helps us to extend the *Web.config* file in order to fulfill our requirements. In order to have a custom configuration section, we need to follow the below steps:

Before we actually start working with it, we will have a look at the `section` settings. We need to have a `ProductSection` element with child elements `girdSettings` and `color`. For this purpose, we will create two classes with the child elements which inherits `ConfigurationElement` as shown below:

C# Shrink ▲

```
public class GridElement : ConfigurationElement
{
    [ConfigurationProperty("title", DefaultValue = "Arial", IsRequired = true)]
    [StringValidator(InvalidCharacters = "~!@#$%^&*()[]{}/;'\"|\\",
            MinLength = 1, MaxLength = 60)]
    public String Title
    {
        get
        {
            return (String)this["title"];
        }
        set
        {
            this["title"] = value;
        }
    }

    [ConfigurationProperty("count", DefaultValue = "10", IsRequired = false)]
    [IntegerValidator(ExcludeRange = false, MaxValue = 30, MinValue = 5)]
    public int Count
    {
        get
        { return (int)this["count"]; }
        set
        { this["size"] = value; }
    }
}

public class ColorElement : ConfigurationElement
{
    [ConfigurationProperty("background", DefaultValue = "FFFFFF", IsRequired = true)]
    [StringValidator(InvalidCharacters = "~!@#$%^&*()[]{}/;
        '\"|\\GHIJKLMNOPQRSTUVWXYZ", MinLength = 6, MaxLength = 6)]
    public String Background
    {
        get
        {
            return (String)this["background"];
        }
        set
        {
            this["background"] = value;
        }
    }

    [ConfigurationProperty("foreground", DefaultValue = "000000", IsRequired = true)]
    [StringValidator(InvalidCharacters = "~!@#$%^&*()[]{}/;
        '\"|\\GHIJKLMNOPQRSTUVWXYZ", MinLength = 6, MaxLength = 6)]
    public String Foreground
    {
        get
        {
            return (String)this["foreground"];
        }
        set
        {
            this["foreground"] = value;
        }
    }

}
```

... and then we will create a class called `ProductSection`, for the root element which includes the above child elements.

C# Shrink ▲

```
public class ProductSection : ConfigurationSection
{
    [ConfigurationProperty("gridSettings")]
    public GridElement gridSettings
    {
        get
        {
            return (GridElement)this["gridSettings"];
        }
        set
        { this["gridSettings"] = value; }
    }

    // Create a "color element."
    [ConfigurationProperty("color")]
    public ColorElement Color
    {
        get
        {
            return (ColorElement)this["color"];
        }
        set
        { this["color"] = value; }
    }
}
```

Then finally, we will configure these elements in *Web.config* file as shown below:

XML

```
<configSections>
     <section name ="ProductSection" type ="<ProductSection"/>
 </configSections>
```

```
<ProductSection>
  <gridSettings title ="Latest Products" count ="20"></gridSettings>
  <color background="FFFFCC" foreground="FFFFFF"></color>
</ProductSection>
```

Access Custom Configuration Section

The following code is used to access the custom configuration section:

C#

```
ProductSection config = (ProductSection)ConfigurationManager.GetSection("ProductSection");
string color =config.Color.Background;
string title =config.gridSettings.Title;
int count = config.gridSettings.Count;
```

Conclusion

In this article, we have learned about the ASP.NET configuration file and we have seen almost all the available and frequently used settings of *web.config* file. I hope you enjoyed reading this article and this article might have helped you in completing your tasks in some way. Any comments, suggestions and feedback are always welcome, which will help me to write more articles and improve the way in which I present the articles.