# Creating a Custom Configuration Section in C#

[Dennis Fazekas](http://www.codeproject.com/script/Membership/View.aspx?mid=2259174), 18 Sep 2007

Creation of a custom configuration section similar to AppSettings. Stores settings for both Development and Production environments, and returns the appropriate settings based on the machine’s configuration in which the application is being executed.

## Introduction

The purpose of this project is to demonstrate the creation of a custom configuration section using the Configuration Manager API. My sample is based on a common issue that many companies experience when deploying applications from a development environment to a production environment. Many times, the configuration settings between environments are different. The custom configuration section implemented by this project will store both the development and production configuration settings and return the setting based on a codebase setting.

This demonstration will walk you through the creation of an assembly (DLL) to implement the Configuration Manager API classes that are required to read the custom configuration section.

It is recommended that the sample application be downloaded and opened using Visual Studio 2005. Doing so will save time and help with the understanding of the different aspects of this demonstration as they are mentioned.

What to Expect from this Article

This article will explain how to create a custom configuration section. It explains how to implement the Configuration Manager API base classes that are required for reading the custom configuration section. Then, it describes how to create a wrapper class that mimics the AppSettings class, with a slight twist to make it environment aware based on the "codebase" setting. This article will explain how to do the following:

Configure a custom configuration section in the application configuration file.

How to implement the Configuration Manager API base classes to read the custom configuration section.

Implementing System.Configuration.ConfigurationElement

Implementing System.Configuration.ConfigurationElementCollection

Implementing System.Configuration.ConfigurationSection

Creating a wrapper class that mimics System.Configuration.ConfigurationManager.AppSettings

Configuring Visual Studio for intellisence and validation of the XML tags for the custom configuration section in the configuration file.

How it Works

A new appSettings setting called Codebase will need to be created in the Machine.Config file. This setting tells the SHI.WebTeam.Common.Configuration.ConfigurationSettings.AppSettings class which setting to retrieve and store for the environment the application is being executed.

The SHI.WebTeam.Common.Configuration.ConfigurationSettings.AppSettings class also retrieves the settings stored in the default appSettings section of the configuration file. All settings in the shiSettings section will override the settings in the appSettings section. The functionality of the appSettings section has not been modified. All settings in the appSettings section of the configuration file are the same regardless of the Codebase value.

The SHI.WebTeam.Common.Configuration wrapper class has a static property called AppSettings which returns a NameValueCollection mimicking Microsoft's implementation of System.Configuration.ConfigurationManager.AppSettings.

Implementing System.Configuration.ConfigurationElement

The code listed below implements the System.Configuration.ConfigurationElement. It represents an element within a configuration file.

An element within a configuration file refers to a basic XML element or a section. The basic element is a simple XML tag with related attributes. A section coincides with a basic element. Complex sections can contain one or more basic elements, a collection of elements, and other sections.

This class represents the XML data stored in the shiSettings section in the configuration file. Basically, this is creating the map between the class and the XML. Rather than having to deal with the XML directly, the ConfugationManager handles it. If new attributes are added to the shiSettings "Add" element in the configuration file, it must be defined in this class too.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png  Copy Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description : This class maps the attributes elements in the

\* configuration file to this class. Represents the

\* <add /> element.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using System;

namespace SHI.WebTeam.Common.ConfigurationManager

{

/// <summary>

/// This class represents the structure of the SHI settings structure

/// in the WebConfig.Conf or App.Conf. These are the attributes.

/// </summary>

public class ShiSettingElements: System.Configuration.ConfigurationElement

{

/// <summary>

/// Returns the key value.

/// </summary>

[System.Configuration.ConfigurationProperty("key", IsRequired = true)]

public string Key

{

get

{

return this["key"] as string;

}

}

/// <summary>

/// Returns the setting value for the production environment.

/// </summary>

[System.Configuration.ConfigurationProperty("prod", IsRequired = true)]

public string Prod

{

get

{

return this["prod"] as string;

}

}

/// <summary>

/// Returns the setting value for the development environment.

/// </summary>

[System.Configuration.ConfigurationProperty("dev", IsRequired = true)]

public string Dev

{

get

{

return this["dev"] as string;

}

}

/// <summary>

/// Returns the setting description.

/// </summary>

[System.Configuration.ConfigurationProperty("desc",IsRequired = false)]

public string Desc

{

get

{

return this["desc"] as string;

}

}

}

}

Implementing System.Configuration.ConfigurationElementCollection

The ConfigurationElementCollection represents a collection of elements within a configuration file. This class is a collection of all the shiSettings stored in the configuration file.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png  Copy Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description : This class is the collection of settings loaded

\* from the WebConfig.Conf or App.Conf.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using System;

namespace SHI.WebTeam.Common.ConfigurationManager

{

public class ShiSettingCollection :

System.Configuration.ConfigurationElementCollection

{

public ShiSettingElements this[int index]

{

get

{

return base.BaseGet(index) as ShiSettingElements;

}

set

{

if (base.BaseGet(index) != null)

{

base.BaseRemoveAt(index);

}

this.BaseAdd(index, value);

}

}

protected

override System.Configuration.ConfigurationElement CreateNewElement()

{

return new ShiSettingElements();

}

protected override object GetElementKey(

System.Configuration.ConfigurationElement element)

{

return ((ShiSettingElements)element).Key;

}

}

}

Implementing System.Configuration.ConfigurationSection

The ConfigurationSection class implements the custom section type. The ConfigurationSection class provides custom handling and programmatic access to custom configuration sections. This class retrieves the shiSettings from the "shiConfiguration" section in the configuration file.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png  Copy Code

using System;

using System.Web.Configuration;

namespace SHI.WebTeam.Common.ConfigurationManager

{

/// <summary>

/// This class is actually what loads the custom settings.

/// </summary>

public class ShiConfiguration : System.Configuration.ConfigurationSection

{

private static string sConfigurationSectionConst = "shiConfiguration";

/// <summary>

/// Returns an shiConfiguration instance

/// </summary>

public static ShiConfiguration GetConfig()

{

return (ShiConfiguration)System.Configuration.ConfigurationManager.

GetSection(ShiConfiguration.sConfigurationSectionConst) ??

new ShiConfiguration();

}

[System.Configuration.ConfigurationProperty("shiSettings")]

public ShiSettingCollection shiSettings

{

get

{

return (ShiSettingCollection)this["shiSettings"] ??

new ShiSettingCollection();

}

}

}

}

Creating the Wrapper Class

This class loads the settings from the configuration file using the classes created above into a static NameValueCollection based on the current Codebase value. The code has been commented to try to explain what's going on. Take notice of the "AppSettings" property; it returns a static indexer using the [Singleton](http://en.wikipedia.org/wiki/Singleton_pattern) technique. There is also a private sub class called ShiSettingsSection which loads the shiSettings and the AppSettings sections into the NameValueCollection based on the codebase.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png  Copy Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Description : This class is the wrapper which loads the settings

\* stored in the shiSettings section in the

\* configuration file. (Web.Config or App.Config)

\* Settings returned by this class are based on the

\* value of the Codebase setting in the machine.conf

\* file. The AppSettings property has been designed

\* to mimic Microsoft's AppSettings.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using System;

using SHI.WebTeam.Common.Configuration.Enum;

using System.Collections.Specialized;

using System.Collections;

namespace SHI.WebTeam.Common.Configuration.Enum

{

public enum CodeBases { Development, Production, Invalid, NotSet };

}

namespace SHI.WebTeam.Common.Configuration

{

public class ConfigurationSettings

{

private static object sLockingObject = new object();

private static CodeBases sCodebase = CodeBases.NotSet;

private static ShiSettingsSection sSettings = null;

/// <summary>

/// Gets the SHI.Configuration.AppSettingsSection data based on the

/// machine's codebase for the current application's

/// codebase default settings.

/// In addition gets the System.Configuration.AppSettingSection data for

/// the current application's default settings if the setting does not

/// exist in the SHI.Configuration.AppSettingsSection.

/// </summary>

/// <param name="name">The name of the setting to be retreived.</param>

/// <returns>Returns the setting specified.</returns>

[System.Diagnostics.DebuggerNonUserCode()]

public static NameValueCollection AppSettings

{

lock (sLockingObject)

{

//If the settings weren't loaded then load them.

if (sSettings == null)

{

sSettings = new ShiSettingsSection();

sSettings.GetSettings();

}

}

return (NameValueCollection)sSettings;

}

/// <summary>

/// Gets the Codebase setting in which the application is

/// being executed.

/// </summary>

static public CodeBases CodeBase

{

get

{

if (ConfigurationSettings.sCodebase == CodeBases.NotSet)

{

//Get the codebase value from the config file.

string ConfigCodeBase =

System.Configuration.ConfigurationManager.

AppSettings["Codebase"].ToLower();

//Convert the codebase string to the enum value.

if (ConfigCodeBase == "prod")

ConfigurationSettings.sCodebase = CodeBases.Production;

else if (ConfigCodeBase == "dev")

ConfigurationSettings.sCodebase = CodeBases.Development;

else

ConfigurationSettings.sCodebase = CodeBases.Invalid;

}

return ConfigurationSettings.sCodebase;

}

}

/// <summary>

/// Validates the value of the codebase setting. If the value is not

/// supported an exception is thrown.

///

/// The codebase settings should be set in the machine.config file

/// usually located at

/// \WINDOWS\Microsoft.NET\Framework\v2.X.X\Config\Machine.Config.

///

/// In the appSettings section create a key called

/// "codebase" and a value of

/// "Prod" or "Dev". Missing values or other values will be concidered

/// invalid.

/// <returns>Returns the value of the codebase

///setting machine.config file.</returns>

[System.Diagnostics.DebuggerNonUserCode()]

static private CodeBases validateCodebase()

{

if (ConfigurationSettings.CodeBase == CodeBases.NotSet)

{ //The codebase setting is not configured throw an exception.

throw new Exception(

"Missing codebase value in the machine.config " +

"file under the appSettings. Allowed values are \"

prod\" and \"dev\"");

}

else if (ConfigurationSettings.CodeBase == CodeBases.Invalid)

{ //The codebase isn't the expected value throw an exception.

throw new Exception(

"Invalid codebase value in the machine.config file " +

"under the appSettings. Allowed values are \"

prod\" and \"dev\"");

}

//The codebase was set and value, so return the

//current machine codebase.

return ConfigurationSettings.CodeBase;

}

#region Private shiSettingsSection Class

private class ShiSettingsSection : NameValueCollection

{

/// <summary>

/// Populates the collection with the SHI and Application Settings

/// based on the current codebase.

/// </summary>

[System.Diagnostics.DebuggerNonUserCode()]

public void GetSettings()

{

//If the settings collection is not populated, populate it.

if (base.Count == 0)

{

//Validate the codebase and get the current Codebase.

CodeBases codebase = validateCodebase();

//Load the ShiConfiguration section from the .Config file.

SHI.WebTeam.Common.ConfigurationManager.ShiConfiguration

ConfigSettings =

SHI.WebTeam.Common.ConfigurationManager.

ShiConfiguration.GetConfig();

//Only populate if the section exists.

if (ConfigSettings != null)

{

//Add the setting for the current machine's codebase to the

//settings collection. Current Codebases

//values are "Production"

//and "Development". The validateCodebase method inforces

//this.

for (int i = 0; i < ConfigSettings.shiSettings.Count; i++)

{

if (ConfigurationSettings.CodeBase ==

CodeBases.Production)

{

base.Add(ConfigSettings.shiSettings[i].Key,

ConfigSettings.shiSettings[i].Prod);

}

else if (ConfigurationSettings.CodeBase ==

CodeBases.Development)

{

base.Add(ConfigSettings.shiSettings[i].Key,

ConfigSettings.shiSettings[i].Dev);

}

else

{

throw new Exception(

"The configured codebase value is " +

"not currently implemented.");

}

}

}

// Load System.ConfigurationManager.AppSettings for

//all settings

// not loaded in the SHI Configuration Setting Section.

NameValueCollection appSettings =

System.Configuration.ConfigurationManager.AppSettings;

for (int i = 0; i < appSettings.Count; i++)

{

string key = appSettings.Keys[i];

//If the Key does not exist in the SHI settings add it.

if (base[key] == null)

{

base.Add(key, appSettings[i]);

}

}

}

}

#region Overrides

/// <summary>

/// This configuration is read only and calling

/// this method will throw an exception.

/// </summary>

[System.Diagnostics.DebuggerNonUserCode()]

public override void Clear()

{

throw new Exception("The configuration is read only.");

}

/// <summary>

/// This configuration is read only and calling this

/// method will throw an exception.

/// </summary>

[System.Diagnostics.DebuggerNonUserCode()]

public override void Add(string name, string value)

{

throw new Exception("The configuration is read only.");

}

/// <summary>

/// This configuration is read only and calling this method

/// will throw an exception.

/// </summary>

[System.Diagnostics.DebuggerNonUserCode()]

public override void Remove(string name)

{

throw new Exception("The configuration is read only.");

}

/// <summary>

/// This configuration is read only and calling this

/// method will throw an exception.

/// </summary>

[System.Diagnostics.DebuggerNonUserCode()]

public override void Set(string name, string value)

{

throw new Exception("The configuration is read only.");

}

#endregion

}

#endregion

}

}

Configurating the Configuration File for the Custom Configuration Section

In order for the .NET Framework to understand the new configuration section, the <configSections> element will need to be added to the Web.Config, App.Config, or Machine.Config file. Since the included sample application is an ASP.NET application, we will add these settings to the web.config file. Add the following XML into the application configuration file inside the <configuration>...</configuration> elements.

Hide   Copy Code

<!-- This tells the Configuration Manager

API about the section handler implementation. -->

<section name="shiConfiguration"

type="SHI.WebTeam.Common.ConfigurationManager.ShiConfiguration,

ShiConfigurationSectionHandler,

Version=1.0.0.0,

Culture=neutral"

restartOnExternalChanges="false"

requirePermission="false"

/>

<!-- Place all environment specific setting in this section. -->

<shiConfiguration>

<shiSettings>

<add key="SettingName"

prod="Production Setting"

dev="Development Setting"

desc="Some description, but not required."

/>

<add key="testing"

prod="Hello production world!"

dev="Hello development world!"

/>

</shiSettings>

</shiConfiguration>

XML Tag Intellisence and Validation in the Configuration File

Visual Studio 2005 will need to be configured on how to interpret the new configuration section in the configuration file. Once this has been completed, intellisence and tag validations will begin working for the new configuration section. A new schema section will need to be added to the DotNetConfig.xsd file.

Edit the DotNetConfig.xsd usually located in the C:\Program Files\Microsoft Visual Studio 8\Xml\Schemas\DotNetConfig.xsd folder. If the file cannot be located, do a search on all drives on the system. I recommend making a backup copy of the file prior to modifying it. Once the file has been located, open it in Notepad.exe and add the following XML snippet:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png  Copy Code

<xs:element name="shiConfiguration" vs:help="configuration/shiConfiguration">

<xs:complexType>

<xs:choice minOccurs="0" maxOccurs="unbounded">

<xs:element name="shiSettings"

vs:help="configuration/shiConfiguration/shiSettings">

<xs:complexType>

<xs:choice minOccurs="0" maxOccurs="unbounded">

<xs:element name="add"

vs:help="configuration/shiConfiguration/shiSettings/add">

<xs:complexType>

<xs:attribute name="key" type="xs:string"

use="required" />

<xs:attribute name="prod"

type="xs:string" use="required" />

<xs:attribute name="dev"

type="xs:string" use="required" />

<xs:attribute name="desc"

type="xs:string" use="optional" />

</xs:complexType>

</xs:element>

<xs:element name="remove"

vs:help="configuration/shiConfiguration/shiSettings/remove">

<xs:complexType>

<xs:attribute name="key" type="xs:string"

use="optional" />

</xs:complexType>

</xs:element>

<xs:element name="clear"

vs:help="configuration/shiConfiguration/shiSettings/clear">

<xs:complexType>

<!--tag is empty-->

</xs:complexType>

</xs:element>

</xs:choice>

</xs:complexType>

</xs:element>

</xs:choice>

</xs:complexType>

</xs:element>

Important Side Notes

If this assembly is used on a web server, I recommend installing it into the GAC (Global Assembly Cache). I would also suggest putting the <configSections> into the machine.confg file. This way, it's global to the entire server.

Do not put the CodeBase setting into the App.Config or Web.Config file. Put it into the Machine.Config. Isn't the whole idea to avoid putting environment specific settings into the wrong environment?

The Configuration Section must be added to the configuration file.

Points of Interest

In C#, there isn't a way to create static indexers. The workaround is quite simple, and accomplishes the same task. It's called [Singleton pattern](http://en.wikipedia.org/wiki/Singleton_pattern). Check out the link for more details. This is a cool trick to know.

Using the Code

To retrieve settings from the custom configuration section, simply do the following.

Hide   Copy Code

<!-- Put this code into the web.config or app.config file. -->

<shiConfiguration>

<shiSettings>

<add key="SettingName" prod="Production Setting"

dev="Development Setting"

desc="Some description, but not required."/>

<add key="testing" prod="Hello production world!"

dev="Hello development world!"/>

</shiSettings>

</shiConfiguration>

Below is an example of a public class called "Something". Notice that when getting settings using the SHI implementation, it's similar to the way Microsoft's AppSettings works. Simply changing the using directive from System.Configuration to SHI.WebTeam.Common.Configuration makes the code environment aware.

Hide   Copy Code

using SHI.WebTeam.Common.Configuration;

public class Something

{

public Somthing()

{

//Gets the setting from the SHI Custom Configuration

//Section, NOT Microsoft's

string Testing = ConfigurationSettings.AppSettings["testing"];

string SettingName = ConfigurationSettings.AppSettings["SettingName"];

//Remember the SHI implementation also loads the setting from the

//AppSettings section

string AnAppSettingsSetting =

ConfigurationSettings.AppSettings["appSettings"];

...

}

}

Conclusion

Let me finish by saying this is the first time I've posted anything to the site. So I hope I did a good job and explained everything thoroughly enough. I hope you enjoyed this example and find it useful. If there are any questions or comments, please feel free to contact me.