ContextConfig C# Library

This C# library allows you to set configuration values dependent upon which web server environment the code is currently running. Environments are defined using the hostname (url) from which the page is operating using a basic XML config file. Multiple hostnames can be set for each environment. You can also add an optional "catch-all" wildcard domain indicating which environment should be assumed if the current domain doesn't match any that have been predefined.

Contents

Getting Started

The only files needed to use this library in your .Net website are the compiled .dlls and an XML configuration file.

Note There is a dependency on the "System.Configuration" namespace. A copy of this .dll has been included in case you need it.

1. Copy the files from the desired Version folder in the "Release" folder to your site.
2. The XML config file can either be located in a root-level folder named "config" or it can be placed at the website root.
3. Customize the provided .config file as described below.
4. Reference "HLF.ContextConfig" in your code where you want to utilize configuration values.

Configuration

The example .config file should give you a good idea about how to add you own data, but here is a brief explanation of the different parts.

```
<ContextConfig version="1.1" OverrideConfigurationManager="true">
```

The "version" attribute isn't really used by the code right now, so you can set this to whatever data you'd like or just use it for your own reference (for instance, updating it whenever you update the data in the config file).

The "OverrideConfigurationManager" option specifies whether or not you want to utilize ContextConfig data to override values specified in the web.config (if the appkeys match) when using the ConfigurationManager.AppSettings["key"] command. (See the *"Override Configuration Manager" Mode* section, below, for more information.)

```
  <!-- Add a line for each domain you plan on having-->
  <Domains>
    <Domain url="mysite.local" environment="local" sitename="MySite"/>
    <Domain url="localhost:52426" environment="local" sitename="MySite"/>
    <Domain url="mysite.dev " environment="dev" sitename="MySite"/>
    <Domain url="mysite-stage.com" environment="staging" sitename="MySite"/>
    <Domain url="mysite.com" environment="live" sitename="MySite"/>
    <Domain url="*" environment="live" sitename="MySite"/><!-- handles any additional domains, delete if
you'd prefer that an error be raised -->
  </Domains>
```

Define all your domains with their hostnames and a string labelling the environment they belong to. You can use any text to represent the different environments – just be sure you have matching <Environment> elements defined. You can have multiple domains with the same environment defined. You can even use Visual Studio "localhost:port" format.

You can also include a wildcard domain ("*") which will match any other domains which haven't been specifically defined. If you exclude the wildcard, and a non-defined domain is used to lookup a config value, an exception will be thrown – which might be the desired behavior to alert you to a non-configured domain.

There is an optional attribute, “sitename”, which you can use as well. Site names do not need to be unique or match up with environments in any way. It’s more of an additional marker for your own use in code.

```
  <Environments>
    <!-- The "default" Environment holds default values, to be used if there isn't a specific value specified for an environment -->
    <Environment name="default">
      <Configs>
        <add key="MyAppKey" value="Default Value"/>
        <add key="LiveOnlyAppKey" value="Default for all env other than live"/>
      </Configs>
    </Environment>
    <Environment name="local">
      <Configs>
        <add key="MyAppKey" value="Value for Local"/>
      </Configs>
    </Environment>
    <Environment name="dev">
      <Configs>
        <add key="MyAppKey" value="Value for Dev"/>
      </Configs>
    </Environment>
    <Environment name="staging">
      <Configs>
        <add key="MyAppKey" value="Value for Staging"/>
      </Configs>
    </Environment>
    <Environment name="live">
      <Configs>
        <add key="MyAppKey" value="Value for Live"/>
        <add key="LiveOnlyAppKey" value="Live-specific Value"/>
      </Configs>
    </Environment>
  </Environments>
```

Each Environment should be configured with a name which matches those used by the Domains defined above. In addition, if you create an Environment with the name “default”, those values will be used in the event of an individual environment not having a value defined for a given key. If, when looking up a config value, the key cannot be matched, either for the environment explicitly or via a default value, an exception will be thrown.

Code Usage

There are two ways to access the configuration data – the simplest is using the functions library in the 'ContextConfig' static class, which will handle lookup values and automatically take into consideration defaults and wildcards while doing so. There are also some useful helpers – to check whether data has been configured, etc.

You can also access all the configuration data directly if you want to create your own logic and flow. Use the 'ConfigSettings' class which has properties and elements representing configured domains, environments, and key/value pairs. (See the *API Reference* section for all the details.)

“Override Configuration Manager” Mode

When enabled, any calls to ConfigurationManager.AppSettings["key"] will utilize ContextConfig data as follows:

1. If there is a value specified for the key for the current environment, it will be used.
2. If there is no matching value for this environment, but there is a default value, the specified default value will be used.
3. If there is no record of the key in the ContextConfig.config file, it will use the value specified in the web.config <appSettings> section.
4. If there is no matching key in the web.config, an empty string will be returned, as usual.

To enable "Override Configuration Manager" mode, a call needs to be made at application start to "ContextConfigOverride.ActivateOverride()."

An HttpModule class has been included, so the call can be made in a web application by simply registering the HttpModule in the application's web.config and setting the ContextConfig.config "OverrideConfigurationManager" attribute to "true".

Web.config example:

```
<configuration>
  ...
 <!-- For IIS6/Classic -->
 <system.web>
  <httpModules>
   <add name="ContextConfigOverrideModule" type="HLF.ContextConfig.ContextConfigOverrideModule,
HLF.ContextConfig" />
  </httpModules>
 </system.web>

 <!-- For IIS7+/Integrated -->
 <system.webServer>
  <modules>
   <add name="ContextConfigOverrideModule" type="HLF.ContextConfig.ContextConfigOverrideModule,
HLF.ContextConfig" />
  </modules>
 </system.webServer>
 ...
</configuration>
```

Alternately, a call in the Global.asax "Application_Start()" section can be used:

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Start(object sender, EventArgs e)
    {
        //Uses ContextConfig value
        HLF.ContextConfig.ContextConfigOverride.ActivateOverride();

        //Uses provided value
        HLF.ContextConfig.ContextConfigOverride.ActivateOverride(true);
    }
}
```

If you make the call to ActivateOverride() with no parameters provided, the value from the ContextConfig.config "OverrideConfigurationManager" attribute will be used (true=enable, false=disable). Otherwise, you can provide a value of true or false directly, and the config value will be ignored.

Examples

If you are using the "Override Configuration Manager" Mode described above, you can just use the standard "ConfigurationManager.AppSettings["key"]" code to get your config values, and when present, the ContextConfig values will be used. If you don't want to override default ConfigurationManager functionality, and/or would like to utilize some of the additional functionality included in the ContextConfig class, these examples, which have been taken from the "TestSite" project included with the source code, will be helpful.

First, reference the namespace with a using statement:

```
using HLF.ContextConfig;
```

Examples of Lookup Functions

```
string CurrentDomainHost= ContextConfig.CurrentDomain;

bool CurrentDomainIsConfiged = ContextConfig.DomainIsConfigured();

bool SpecifiedDomainIsConfiged = ContextConfig.DomainIsConfigured("xyz.com");

bool SpecifiedDomainIsConfigedExplicity  = ContextConfig.DomainIsConfigured("xyz.com", false);

string CurrentEnvironment = ContextConfig.DomainEnvironmentName();

string SpecifiedDomainEnvironment = ContextConfig.DomainEnvironmentName("xyz.com");

string CurrentDomainSiteName = ContextConfig.DomainSiteName();

string SpecifiedDomainSiteName = ContextConfig.DomainSiteName("xyz.com");

bool CurrentEnvIsConfiged = ContextConfig.EnvironmentIsConfigured();

bool SpecifiedEnvIsConfiged = ContextConfig.EnvironmentIsConfigured("testing");

bool SpecifiedEnvIsConfigedExplicity = ContextConfig.EnvironmentIsConfigured("testing", false);

string GetValueForSpecifiedKeyForCurrentEnv = ContextConfig.GetValue("MyAppKey");

string GetValueForSpecifiedKeyForSpecifiedEnv= ContextConfig.GetValue("LiveOnlyAppKey", "live");

string GetValueForSpecifiedKeyForSpecifiedEnvDefault =
ContextConfig.GetValue("LiveOnlyAppKey","dev");


try
{
    string ValueForNonExistentKey = ContextConfig.GetValue("KeyThatDoesntExist"); //Here we throw
an error
}
catch (Exception Ex)
{
    Response.Write(string.Format("ERROR: {0} : {1}", Ex.GetType().ToString(), Ex.Message));

}
```

Examples of Raw Data Functions

```
string Version = ConfigSettings.Settings.Version;

int DomainsCount = ConfigSettings.Settings.Domains.Count;

int d = 0;
foreach (DomainElement Domain in ConfigSettings.Settings.Domains)
{
    Response.Write(string.Format("Domain {0} : {1} = {2} ({3}) <br/>", d, Domain.Url,
Domain.Environment, Domain.SiteName));
    d++;
}

int EnvironmentsCount = ConfigSettings.Settings.Environments.Count;

int e = 0;
```

```
foreach (EnvironmentElement Env in ConfigSettings.Settings.Environments)
{
    Response.Write(string.Format("Environment {0} : {1} <br/>", e, Env.Name));
    e++;

    int ConfigsCount = Env.Configs.Count;
    Response.Write(string.Format("ConfigsCount: {0} <ul>", ConfigsCount));

    int c = 0;
    foreach (KeyValueElement KeyValue in Env.Configs)
    {
        Response.Write(string.Format("<li> {0} = {1} </li>", KeyValue.Key, KeyValue.Value));
        c++;
    }
    Response.Write("</ul>");
}
```

Use Cases

The obvious use for ContextConfig is to manage variables which differ depending on what environment the site is running on. For instance, api keys for 3rd party services which have “dev” and “production” versions available. It can also be used in your own code logic to test for the environment and perform different operations – for instance, logging additional debug data, or skipping or requiring authentication based on the current environment.

Another possible use case would be to manage multiple websites inside one code base (as is common in Umbraco CMS sites). Rather than using the common paradigm of “dev/staging/live” environments, you could define, for instance, “US/UK/CANADA” “environments”, etc. Since the quantity of environments is not limited, you could even use a combination such as “US-dev/US-live/UK-dev/UK-live”. Just make sure that each variant you want to use is defined as a separate environment.

API Reference

Note In the interest of brevity, I have left out of this reference many of the inherited elements from the System.Configuration class.

HLF.ContextConfig Namespace

This namespace contains both the "raw" access to the configuration data elements, as well as the static functions which can easily be used to intelligently grab config values for use in code.

Assembly: HLF.ContextConfig (in HLF.ContextConfig.dll) Version: 1.0.0.0 (1.0.0.0)

ContextConfig has been compiled against ASP.Net v. 4.5

Classes

	Class	Description

	ConfigSettings	The 'ConfigSettings' class give you direct access to all the configured data using collections and dot(.) notation.
		
		*Note: you will need to explicitly declare objects of the element types in order to use their properties.
		example:
		`foreach (DomainElement Domain in ConfigSettings.Settings.Domains)`
		`{`
		`  string EnvironmentName = Domain.Environment;`
		`}`

	ContextConfig	The 'ContextConfig' static class includes useful functions to test and return data about domains, environments, and configured key/value pairs.
	ContextConfigOverride	Handles overriding standard 'ConfigurationManager.AppSettings["key"]' operations
	ContextConfigOverrideModule	HttpModule to run 'ContextConfigOverride.ActivateOverride()' Register in Web.config file: `<add name="ContextConfigOverrideModule" type="HLF.ContextConfig.ContextConfigOverrideModule, HLF.ContextConfig" />` IIS6 or Classic Mode - place in <system.web><httpModules> section IIS7+ or Integrated Mode - place in <system.webServer><modules> section
	DomainElement	Represents a Domain from the config file
	DomainElementCollection	Represents all the Domains defined in the config file
	EnvironmentElement	Represents a defined Environment from the config file
	EnvironmentElementCollection	Represents all the Environments defined in the config file
	KeyValueElement	Represents a Key/Value pair defined for an environment in the config file
	KeyValueElementCollection	Represents all the Key/Value pair elements defined in the config file

ContextConfig Class

The 'ContextConfig' static class includes useful functions to test and return data about domains, environments, and configured key/value pairs.

Inheritance Hierarchy

System.Object

HLF.ContextConfig.ContextConfig

Methods

	Name	Description
S	AllEnvironmentConfigs(Boolean)	Get all the KeyValue elements for the current Environment *Parameters* • *IncludeDefaults* (Optional *true*) (Type: System.Boolean) : Include all 'default' KeyValue configs for keys not specifically defined for the environment. *Return Value* Type: List<KeyValueElement>
S	AllEnvironmentConfigs(String, Boolean)	Get all the KeyValue elements for the current Environment *Parameters* • *EnvironmentName* (Type: System.String) : Environment to get values from • *IncludeDefaults* (Optional *true*) (Type: System.Boolean)

		: Include all 'default' KeyValue configs for keys not specifically defined for the environment. *Return Value* Type: List<KeyValueElement>
	DomainEnvironmentName()	Get the Environment name for the current domain *Return Value* Type: String
	DomainEnvironmentName(String)	Get the Environment name for the provided domain url *Parameters* • *DomainUrl (*Type: System.String) : Url to lookup *Return Value* Type: String
	DomainIsConfigured(Boolean)	Check whether the current domain exists in the Domains list *Parameters* • *AcceptWildcard* (Optional *true*) (Type: System.Boolean) : If there is a wildcard (*) domain specified, return true? (Choose false to explicitly search for this url) *Return Value* Type: Boolean
	DomainIsConfigured(String, Boolean)	Check whether the URL exists in the Domains list *Parameters* • *DomainUrl (*Type: System.String) : Url to lookup • *AcceptWildcard* (Optional *true*) (Type: System.Boolean) : If there is a wildcard (*) domain specified, return true? (Choose false to explicitly search for this url) *Return Value* Type: Boolean
	DomainSiteName()	Get the Site Name for the current domain *Return Value* Type: String
	DomainSiteName(String)	Get the Site Name for the provided domain url *Parameters* • *DomainUrl (*Type: System.String) : Url to lookup *Return Value* Type: String
	EnvironmentIsConfigured(Boolean)	Check whether the current environment exists in the Environments list *Parameters* • *AcceptDefault* (Optional *true*) (Type: System.Boolean) : If there is a "default" domain specified, return true? (Choose false to explicitly search for this environment) *Return Value* Type: Boolean

	EnvironmentIsConfigured(String, Boolean)	Check whether the current environment exists in the Environments list *Parameters* • *EnvironmentName(*Type: System.String) : Name to lookup • *AcceptDefault* (Optional *true*) (Type: System.Boolean) : If there is a "default" domain specified, return true? (Choose false to explicitly search for this environment) *Return Value* Type: Boolean
	GetValue(String)	Get the value for a given key on the current domain *Parameters* • *ConfigKey (*Type: System.String) : Key name *Return Value* Type: String
	GetValue(String, String)	Get the value when providing a key and environment name *Parameters* • *ConfigKey (*Type: System.String) : Key name • *EnvironmentName (*Type: System.String) : Environment name *Return Value* Type: String

Fields

	Name	Description
	CurrentDomain	Current active domain url Type: String

ConfigSettings Class

The 'ConfigSettings' class give you direct access to all the configured data using collections and dot(.) notation.

*Note: you will need to explicitly declare objects of the element types in order to use their properties.

example:

```
foreach (DomainElement Domain in ConfigSettings.Settings.Domains)
{
    string EnvironmentName = Domain.Environment;
}
```

Inheritance Hierarchy

System.Object
  System.Configuration.ConfigurationElement
    System.Configuration.ConfigurationSection

HLF.ContextConfig.ConfigSettings

Constructors

	Name	Description
	ConfigSettings	Initializes a new instance of the **ConfigSettings** class

Fields

	Name	Description
	Settings	Represents all the ConfigSettings

Properties

	Name	Description
	Domains	<ContextConfig> <Domains> collection *Property Value* Type: DomainElementCollection
	Environments	<ContextConfig> <Environments> collection *Property Value* Type: EnvironmentElementCollection
	OverrideConfigurationManager	<ContextConfig> 'OverrideConfigurationManager' attribute *Property Value* Type: String-TODO
	Version	<ContextConfig> 'version' attribute *Property Value* Type: String

DomainElementCollection Class

Represents all the Domains defined in the config file

Inheritance Hierarchy

System.Object
  System.Configuration.ConfigurationElement
    System.Configuration.ConfigurationElementCollection
      HLF.ContextConfig.DomainElementCollection

Constructors

	Name	Description
	DomainElementCollection	Initializes a new instance of the **DomainElementCollection** class

Properties

	Name	Description
	CollectionType	Gets the type of the ConfigurationElementCollection. (Overrides ConfigurationElementCollection.CollectionType.) *Return Value* Type: ConfigurationElementCollectionType The ConfigurationElementCollectionType of this collection.
	Item(Int32)	*Parameters* • *Index* (Type: System.Int32) *Property Value* Type: DomainElement
	Item(String)	*Parameters* • *url* (Type: System.String) *Property Value* Type: DomainElement

DomainElement Class

Represents a Domain from the config file

Inheritance Hierarchy

System.Object
  System.Configuration.ConfigurationElement
    HLF.ContextConfig.DomainElement

Constructors

	Name	Description
	DomainElement	Initializes a new instance of the **DomainElement** class

Properties

	Name	Description
	Environment	<Domain> 'environment' attribute *Property Value* Type: String
	SiteName	<Domain> 'sitename' attribute *Property Value* Type: String
	Url	<Domain> 'url' attribute *Property Value* Type: String

EnvironmentElementCollection Class

Represents all the Environments defined in the config file

Inheritance Hierarchy

System.Object
  System.Configuration.ConfigurationElement
    System.Configuration.ConfigurationElementCollection
      HLF.ContextConfig.EnvironmentElementCollection

Constructors

	Name	Description
	EnvironmentElementCollection	Initializes a new instance of the **EnvironmentElementCollection** class

Properties

	Name	Description
	CollectionType	Gets the type of the ConfigurationElementCollection. (Overrides ConfigurationElementCollection.CollectionType.) *Return Value* Type: ConfigurationElementCollectionType The ConfigurationElementCollectionType of this collection.
	Item(Int32)	*Parameters* • *Index* (Type: System.Int32) *Property Value* Type: EnvironmentElement
	Item(String)	*Parameters* • *name* (Type: System.String) *Property Value* Type: EnvironmentElement

EnvironmentElement Class

Represents a defined Environment from the config file

Inheritance Hierarchy

System.Object
  System.Configuration.ConfigurationElement
    HLF.ContextConfig.EnvironmentElement

Constructors

	Name	Description
	EnvironmentElement	Initializes a new instance of the **EnvironmentElement** class

Properties

	Name	Description
	Configs	<Environment> <Configs> (key/values) collection *Property Value* Type: KeyValueElementCollection
	Name	<Environment> 'name' attribute *Property Value* Type: String

KeyValueElementCollection Class

Represents all the Key/Value pair elements defined in the config file

Inheritance Hierarchy

System.Object
  System.Configuration.ConfigurationElement
    System.Configuration.ConfigurationElementCollection
      HLF.ContextConfig.KeyValueElementCollection

Constructors

	Name	Description
	KeyValueElementCollection	Initializes a new instance of the **KeyValueElementCollection** class

Properties

	Name	Description
	CollectionType	Gets the type of the ConfigurationElementCollection. (Overrides ConfigurationElementCollection.CollectionType.) *Return Value* Type: ConfigurationElementCollectionType The ConfigurationElementCollectionType of this collection.
	Item(Int32)	*Parameters* • *Index (*Type: System.Int32) *Property Value* Type: EnvironmentElement
	Item(String)	*Parameters* • *key (*Type: System.String) *Property Value* Type: EnvironmentElement

KeyValueElement Class

Represents a Key/Value pair defined for an environment in the config file

Inheritance Hierarchy

System.Object
  System.Configuration.ConfigurationElement
    HLF.ContextConfig.KeyValueElement

Constructors

	Name	Description
	KeyValueElement	Initializes a new instance of the **KeyValueElement** class

Properties

	Name	Description
	Key	<Environment> <add> 'key' attribute *Property Value* Type: String
	Value	<Environment> <add>'value' attribute *Property Value* Type: String

ContextConfigOverride Class

Handles overriding standard 'ConfigurationManager.AppSettings["key"]' operations

Inheritance Hierarchy

System.Object
  HLF.ContextConfig.ContextConfigOverride

Methods

	Name	Description
S	ActivateOverride()	Re-initializes the ConfigurationManager to utilize ContextConfig values when calling 'ConfigurationManager.AppSettings["key"]' A call to this should be placed somewhere before config values are called, for instance, in App Start. Or use the 'ContextConfigOverrideModule' HttpModule for default handling **Will activate the override based on the value in ContextConfig.config**
S	ActivateOverride(Boolean)	Re-initializes the ConfigurationManager to utilize ContextConfig values when calling 'ConfigurationManager.AppSettings["key"]' A call to this should be placed somewhere before config values are called, for instance, in App Start. Or use the 'ContextConfigOverrideModule' HttpModule for default handling *Parameters* • *DoOverride* (Type: System.Boolean) : Should the override be

		activated? (False=no code will run)

ContextConfigOverrideModule Class

HttpModule to run 'ContextConfigOverride.ActivateOverride()'

Register in Web.config file:

```
<add name="ContextConfigOverrideModule" type="HLF.ContextConfig.ContextConfigOverrideModule,
HLF.ContextConfig" />
```

For IIS6 or Classic Mode - place in <system.web><httpModules> section

For IIS7+ or Integrated Mode - place in <system.webServer><modules> section

Inheritance Hierarchy

System.Object
  HLF.ContextConfig.ContextConfigOverrideModule

Methods

	Name	Description
	Dispose	Releases all resources used by the **ContextConfigOverrideModule**
	Init	Runs 'ActivateOverride()' on App Start

Version History

Current

Version 1.1

Version 1.1 was released on July 16, 2014.

Changes in This Release

- Added “Override Configuration Manager” functionality

Previous

Version 1.0

Version 1.0 was released on July 3, 2014.

Changes in This Release

- Initial Release