

[<< Back to Dashboard](#)

ColdBox Configuration File

Contents

Overview

The ColdBox configuration file is the heart of your ColdBox application. It contains the initialization variables for your application and extra information used by the ColdBox software aspects and how your application boots up. Below is an overview of every section of the file and how to configure an application. To provide code hinting and simple validation, I have included `config.xml` file for your usage. This file can be found at `/coldbox/system/config/config.xml`. You can then use any of the tools below to help you get coding hints, color coding and validation.

XML & ColdFusion Tools

- [Eclipse](#)
 - [Plugins](#)
 - [CFEclipse](#)
 - [Web Tools Project](#)
 - [XMLBuddy](#)
 - [Oxvgen XML](#)
- [Altova's XML Spv](#)
- [Dreamweaver](#)
- [Good Old Notepad](#).

Note About Security: Your `config.xml` or `coldbox.xml` file is actually a `cfm` template. I did this in order to protect the download of the file with the use of an Application `cfm` template (Thanks to Raymond Camden). However, for added security I also use an apache access file: `.htaccess`. You will find the `.htaccess` file on the distribution archive for the `config` directory. This is used by Apache for executing directory security.

[Read more on .htaccess.](#)

In IIS, you will have to configure the directory's security from the IIS Manager.

[Good article on securing IIS](#)

Sample Coldbox.xml.cfm

```
<?xml version= "1.0" encoding= "UTF-8" ?>
<Config xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation= "http://www.coldbox.org/schema/config_2.6.0.xsd" >
  <Settings>
    <!--The name of your application.-->
    <Setting name= "AppName" value= "Your App Name here" />
    <!--ColdBox set-up information for J2EE installation.
    As context-root are actually virtual locations which does not correspond to physical location of files.
    /openbd /var/www/html/tomcat/deploy/bluedragon

    AppMapping setting will adjust physical location of Project/App files and coldbox will load handlers,plugins
    Create a cf mapping and enable this value.
    /MyApp /var/www/html/tomcat/deploy/bluedragon/MyAppFolder

    If you are using a coldbox app to power flex/remote apps, you NEED to set the AppMapping also. In Summary,
    the AppMapping is either a CF mapping or the path from the webroot to this application root. If this setting
    is not set, then coldbox will try to auto-calculate it for you. Please read the docs.

    <Setting name= "AppMapping" value= "/MyApp"/>

    <!--
    <!--Default Debugmode boolean flag (Set to false in production environments)-->
    <Setting name= "DebugMode" value= "true" />
    <!--The Debug Password to use in order to activate/deactivate debugmode,activated by url actions -->
    <Setting name= "DebugPassword" value= "" />
    <!--The fwreinit password to use in order to reinitialize the framework and application.Optional, else leave blank.
    <Setting name= "ReinitPassword" value= "" />
    <!--Default event name variable to use in URL/FORM etc. -->
    <Setting name= "EventName" value= "event" />
    <!--This feature is enabled by default to permit the url dumpvar parameter-->
    <Setting name= "EnableDumpVar" value= "true" />
    <!--Log Errors and entries on the coldfusion server logs, disabled by default if not used-->
    <Setting name= "EnableColdfusionLogging" value= "false" />
    <!--Log Errors and entries in ColdBox's own logging facilities. You choose the location, finally per application.
    <Setting name= "EnableColdboxLogging" value= "true" />
    <!--The absolute or relative path to where you want to store your log files for this application-->
    <Setting name= "ColdboxLogsLocation" value= "logs" />

    <!--
    The default logging level to use. Defaults to all = 4 valid numbers are 0-4
    0-fatal,1-error,2-warning,3-information,4-debug
    <Setting name= "DefaultLogLevel" value= "4" />
    <!--

    <!--Default Event to run if no event is set or passed. Usually the event to be fired first (NOTE: use event name= "general.index" />
    <Setting name= "DefaultEvent" value= "general.index" />
    <!--Event Handler to run on the start of a request, leave blank if not used. Emulates the Application.cfc onRequestStart method -->
    <Setting name= "RequestStartHandler" value= "main.onRequestStart" />
    <!--Event Handler to run at end of all requests, leave blank if not used. Emulates the Application.cfc onRequestEnd method -->
    <Setting name= "RequestEndHandler" value= "" />
    <!--Event Handler to run at the start of an application, leave blank if not used. Emulates the Application.cfc onRequestStart method -->
    <Setting name= "ApplicationStartHandler" value= "main.onAppInit" />
    <!--Event Handler to run at the start of a session, leave blank if not used.-->
    <Setting name= "SessionStartHandler" value= "" />
    <!--Event Handler to run at the end of a session, leave blank if not used.-->
    <Setting name= "SessionEndHandler" value= "" />
    <!--The event handler to execute on all framework exceptions. Event Handler syntax required.-->
    <Setting name= "ExceptionHandler" value= "" />
    <!--What event to fire when an invalid event is detected-->
    <Setting name= "onInvalidEvent" value= "" />
    <!--Full path from the application's root to your custom error page, else leave blank. -->
    <Setting name= "CustomErrorTemplate" value= "" />
    <!--The Email address from which all outgoing framework emails will be sent. -->
    <Setting name= "OwnerEmail" value= "" />
    <!-- Enable Bug Reports to be emailed out, set to true by default if left blank
    A sample template has been provided to you in includes/generic_error.cfm
    <Setting name= "EnableBugReports" value= "false" />
    <!--UDF Library To Load on every request for your views and handlers -->
    <Setting name= "UDFLibraryFile" value= "includes/helpers/ApplicationHelper.cfm" />
    <!--Messagebox Style Override. A boolean of wether to override the styles using your own css.-->
    <Setting name= "MessageboxStyleOverride" value= "" />
  </Settings>
</Config>
```

• [ColdBox Configuration File](#)

- [Overview](#)
- [Sample Coldbox.xml.cfm](#)
- [Generated Schema Documentation](#)
- [Loading another configuration File](#)
- [Interacting With The Loaded Settings](#)
 - [Methods](#)
 - [Code Samples](#)
- [Setting Replacement](#)
- [Element](#)
 - [Settings Chart](#)
 - [AppName](#)
 - [AppMapping](#)
 - [ApplicationStartHandler](#)
 - [CustomErrorTemplate](#)
 - [CustomEmailBugReport](#)
 - [ColdBoxLogsLocation](#)
 - [ConfigAutoReload](#)
 - [DebugMode](#)
 - [DebugPassword](#)
 - [DefaultEvent](#)
 - [DefaultLogLevel](#)
 - [EventName](#)
 - [ExceptionHandler](#)
 - [EnableBugReports](#)
 - [EnableColdfusionLogging](#)
 - [EnableDumpVar](#)
 - [EnableColdboxLogging](#)
 - [EventCaching](#)
 - [FlashURLPersistScope](#)
 - [HandlersIndexAutoReload](#)
 - [HandlersExternalLocation](#)
 - [HandlerCaching](#)
 - [IOCFramework](#)
 - [IOCFrameworkReload](#)
 - [IOCDefinitionFile](#)
 - [IOCOBJECTCaching](#)
 - [MessageboxStyleOverride](#)
 - [ModelsExternalLocation](#)
 - [ModelsObjectCaching](#)
 - [ModelsSetterInjection](#)
 - [ModelsDebugMode](#)
 - [ModelsDFCompletedPDF](#)
 - [ModelsStopReursion](#)
 - [MyPluginsLocation](#)
 - [onInvalidEvent](#)
 - [OwnerEmail](#)
 - [ProxyReturnCollection](#)
 - [RequestStartHandler](#)
 - [RequestEndHandler](#)
 - [RequestContextDecorator](#)
 - [ReinitPassword](#)
 - [SessionStartHandler](#)
 - [SessionEndHandler](#)
 - [UDFLibraryFile](#)
 - [ViewsExternalLocation](#)
 - [Complex Variable Syntax: JSON](#)
 - [How Do I Use Them?](#)
 - [Default Resource Bundle](#)
 - [Default Locale](#)
 - [Locale Storage](#)
 - [Unknown Translation](#)
 - [I18N Resources](#)
 - [Object Default Timeout](#)
 - [Object Default Last Access Timeout](#)
 - [Use Last Access Timeouts](#)
 - [Reap Frequency](#)
 - [Max Objects](#)
 - [JVM Free Memory Percentage Threshold](#)
 - [EvictionPolicy](#)
 - [throwOnInvalidStates Attribute](#)
 - [Custom Interception Points](#)
 - [Interceptor Element](#)
 - [RequestStartAttribute](#)
 - [Property Elements](#)
 - [Related Guides](#)

```

<!--F<lag to Auto reload the internal handlers directory listing. False for production. -->
<Setting name="HandlersIndexAutoReload" value="true" />
<!--F<lag to auto reload the config.xml settings. False for production. -->
<Setting name="ConfigAutoReload" value="false" />
<!-- <Declare the custom plugins base invocation path, if used. You have to use dot notation.Example: mymapping.myplugins -->
<Setting name="MyPluginsLocation" value="" />
<!-- <Declare the external views location. It can be relative to this app or external. This in turn is used to do cfincludes. -->
<Setting name="ViewsExternalLocation" value="" />
<!-- <Declare the external handlers base invocation path, if used. You have to use dot notation.Example: mymapping.myhandlers -->
<Setting name="HandlersExternalLocation" value="" />
<!-- <Declare the external models base invocation path, if used. You have to use dot notation.Example: mymapping.mymodels -->
<Setting name="ModelsExternalLocation" value="" />
<!-- <Caching of model objects via model integration -->
<Setting name="ModelsObjectCaching" value="true" />

<!-- <Uncomment More Model Integration Settings:
<Setting name="ModelsSetterInjection" value="false" />
<Setting name="ModelsDICompleteUDF" value="onDIComplete" />
<Setting name="ModelsStopRecursion" value="" />
<Setting name="ModelsDebugMode" value="true" />
-->

<!--F<lag to cache handlers. Default if left blank is true. -->
<Setting name="HandlerCaching" value="false" />
<!--F<lag to cache events if metadata declared. Default is true -->
<Setting name="EventCaching" value="false" />
<!--IOC Framework if Used, else leave blank-->
<Setting name="IOCFramework" value="" />
<!-- <Reload IOC Framework on every request, usually for development or debugging -->
<Setting name="IOCFrameworkReload" value="false" />
<!--IOC Definition File Path, relative or absolute -->
<Setting name="IOCDefinitionFile" value="" />
<!--IOC Object Caching, true/false. For ColdBox to cache your IoC beans-->
<Setting name="IOXObjectCaching" value="false" />
<!--Request Context Decorator, leave blank if not using. Full instantiation path -->
<Setting name="RequestContextDecorator" value="" />
<!--F<lag if the proxy returns the entire request collection or what the event handlers return, default is false -->
<Setting name="ProxyReturnCollection" value="false" />
<!-- <What scope are flash persistence variables using. -->
<Setting name="FlashURLPersistScope" value="session" />
</Settings>

<!-- <Your Settings can go here, if not needed, use <YourSettings />. You can use these for anything you like.
<YourSettings>
  <Setting name="MySetting" value="My Value"/>

  whether to encrypt the values or not
  <Setting name="cookiestorage_encryption" value="true"/>

  The encryption seed to use. Else, use a default one (Not Recommended)
  <Setting name="cookiestorage_encryption_seed" value="mykey"/>

  The encryption algorithm to use (According to CFML Engine)
  <Setting name="cookiestorage_encryption_algorithm" value="CFMX_COMPAT or BD_DEFAULT"/>

  MessageBox Plugin (You can now override the storage scope without affecting all framework applications)
  <Setting name="messagebox_storage_scope" value="session or client" />

  Complex Settings follow JSON Syntax. www.json.org.
  *IMPORTANT: use single quotes in this xml file for JSON notation, ColdBox will translate it to double quotes.
</YourSettings>
-->
</YourSettings>
<!-- @YOURSETTINGS@ -->
</YourSettings>

<!-- <Custom Conventions : You can override the framework wide conventions of the locations of the needed objects
<Conventions>
<handlersLocation></handlersLocation>
<pluginsLocation></pluginsLocation>
<layoutsLocation></layoutsLocation>
<viewsLocation></viewsLocation>
<eventAction></eventAction>
<modelsLocation></modelsLocation>
</Conventions>
-->

<!--
<
Control the ColdBox Debugger. The panels are self explanatory. The other settings are explained below.
PersistentRequestProfiler : Activate the event profiler across multiple requests
maxPersistentRequestProfilers : Max records to keep in the profiler. Don't get greedy.
maxRCPanelQueryRows : If a query is dumped in the RC panel, it will be truncated to this many rows.
-->
<DebuggerSettings>
  <PersistentRequestProfiler> true </PersistentRequestProfiler>
  <maxPersistentRequestProfilers> 10 </maxPersistentRequestProfilers>
  <maxRCPanelQueryRows> 50 </maxRCPanelQueryRows>

  <TracerPanel show="true" expanded="true" />
  <InfoPanel show="true" expanded="true" />
  <CachePanel show="true" expanded="false" />
  <RCPanel show="true" expanded="false" />
</DebuggerSettings>

<!--Optional,if blank it will use the CFMX administrator settings.-->
<MailServerSettings>
<MailServer> </MailServer>
<MailPort> </MailPort>
<MailUsername> </MailUsername>
<MailPassword> </MailPassword>
</MailServerSettings>

<!--E<mails to Send bug reports, you can create as many as you like -->
<BugTracerReports>
  <-- <<BugEmail>myemail@gmail.com</BugEmail> -->
</BugTracerReports>

<!--Web<service declarations your use in your application, if not use, leave blank
Note that for the same webservice name you can have a development url and a production url.-->
<WebServices>
  <-- <<WebService name="TESTWS1" URL="http://www.test.com/test1.cfc?wsdl" DevURL="http://dev.test.com/test1.cfc?wsdl" /> -->
  <-- <<WebService name="TESTWS2" URL="http://www.test.com/test2.cfc?wsdl" DevURL="http://dev.test.com/test2.cfc?wsdl" /> -->
</WebServices>

<!--D<eclare Layouts for your application here-->
<Layouts>
  <--D<eclare the default layout, MANDATORY-->
  <DefaultLayout> Layout.Main.cfm </DefaultLayout>

```

```

<!--Default View, OPTIONAL
<DefaultView>home</DefaultView>
-->

<--
<
Declare other layouts, with view/folder assignments if needed, else do not write them
<Layout file="Layout.Popup.cfm" name="popup">
  <View>vwTest</View>
  <View>vwMyView</View>
  <Folder>tags</Folder>
</Layout>
-->
</Layouts>

<!--Internationalization and resource Bundle setup:
<i18N>
<DefaultResourceBundle>includes/main</DefaultResourceBundle>
<DefaultLocale>en_US</DefaultLocale>
<LocaleStorage>session</LocaleStorage>
<UnknownTranslation></UnknownTranslation>
</i18N>
-->
<i18N />

<!--DataSource Setup, you can then retrieve a datasourceBean via the getDataSource("name") method: -->
<DataSources>
  <-- <DataSource alias="MyDSNAlias" name="real_dsn_name" dbtype="mysql" username="" password="" /> -->
</DataSources>

<!--ColdBox Object Caching Settings Overrides the Framework-wide settings -->
<Cache>
  <ObjectDefaultTimeout> 60</ObjectDefaultTimeout>
  <ObjectDefaultLastAccessTimeout> 30</ObjectDefaultLastAccessTimeout>
  <UseLastAccessTimeouts> true</UseLastAccessTimeouts>
  <ReapFrequency> 1</ReapFrequency>
  <MaxObjects> 100</MaxObjects>
  <FreeMemoryPercentageThreshold> 0</FreeMemoryPercentageThreshold>
  <EvictionPolicy> LRU</EvictionPolicy>
</Cache>

<-- <Interceptor Declarations
<Interceptors throwOnInvalidStates="true">
<CustomInterceptionPoints>comma-delimited list</CustomInterceptionPoints>
<Interceptor class="full class name">
  <Property name="myProp">value</Property>
  <Property name="myArray">[1,2,3]</Property>
  <Property name="myStruct">{ key1:1, key2:2 }</Property>
</Interceptor>
<Interceptor class="no property" />
</Interceptors>
-->

<Interceptors>
  <-- <USE ENVIRONMENT CONTROL -->
  <Interceptor class="coldbox.system.interceptors.environmentControl" >
    <Property name='configFile' >config /environments.xml.cfm </Property>
  </Interceptor>
  <-- <USE AUTOWIRING -->
  <Interceptor class="coldbox.system.interceptors.autowire" >
    <Property name='enableSetterInjection' >true</Property>
  </Interceptor>
  <-- <USE SES -->
  <Interceptor class="coldbox.system.interceptors.ses" >
    <Property name="configFile" >config /routes.cfm </Property>
  </Interceptor>
  <-- @@SIDEBAR@ -->
</Interceptors>

</Config>

```

Generated Schema Documentation

Please use the link below to view the generated documentation about the schema file you will use to build your configuration file. This generated documentation will explain every single element and configuration you need in order to use this configuration file.

- <http://www.coldbox.org/documents/SchemaDocs/>

Loading another configuration File

The framework by default will look for a `coldbox.xml.cfm` or `config.xml.cfm` file in your `config` folder or as specified by your [Naming Conventions](#). However, you can override the conventions and tell the framework which configuration file to load manually. This is important for multi-tiered environments, unit tests or for whatever reason you like. In order to do this, you will have to alter a processing variable in your application.cfc or index.cfm right before including the framework or processing a framework request.

Interacting With The Loaded Settings

As you learned from the [Framework Settings Guide](#) this xml file is converted into a structure and stored in the ColdBox Application Controller alongside the Framework Settings. The framework provides you with a set of methods to interact with the structures and you can see all of them in the [CFC API](#). However, below I layout the main methods you will call from your handlers, layouts, views, plugins, etc.

Methods

- `getSetting([string name], [boolean FWSetting])`
- `setSetting([string name], [any value])`
- `settingExists([string name], [boolean FWSetting])`
- `getSettingStructure([boolean FWSetting], [boolean DeepCopyFlag])`
- `controller.getConfigSettings()`
- `controller.getColdboxSettings()`
- `getSettingsBean()` : construct a `coldbox.system.beans.ConfigBean` with all the application settings.

FWSetting = Use the framework settings structure instead of the application settings structure

Code Samples

```

//Get a setting
<cfset getSetting( "key" )>

//Get the settings structure using controller getter method
<cfset getSettingStructure( false )>

```

```
//Set a new setting
<cfset setSetting( "key" ,value) >

//Check if setting exists
<cfset settingExists( "key" )>

//get the framework structure settings
<cfset frameworkSettings = controller.getColdboxSettings() >

//get the application settings
<cfset appSettings = controller.getConfigSettings() >

//Construct a configuration bean with all of my configuration settings
<cfset configBean = getSettingsBean() >
```

If you would like to reload your settings, you will need to reinitialize the framework by using the `fwreinit=1` URL action (See [URL Actions Guide](#)). This is a very important url action, as most of the time your settings are cached and you need to reload them. So please read the guide.

#{setting} Replacement

This feature is available since version 2.6.2. You can use the `#{setting}` notation, almost anywhere in the coldbox.xml file to retrieve and create dynamic settings, class paths or anything you like. You can use any setting that can be found in the coldbox.xml. The following are the locations within the coldbox.xml that you can use dynamic notation:

- Within any `<Settings>` element (As long as the setting was declared before the target setting)
- Within any `<YourSettings>` element (As long as the setting was declared before the target setting)
- Interceptor Custom Interception Points
- Interceptor class paths
- Interceptor Properties

<---< Samples --->

```
<Settings>
<Setting name= "AppMapping" value= "/myApp" />
</Settings>

<YourSettings>
<Setting name= "MyType" value= "MSSQL" />
<Setting name= "MyFilePath" value= "${ApplicationPath}/customfiles" />
</YourSettings>

<interceptor name= "myInterceptor" class= "${AppMapping}.model.interceptors.#{MyType}Gate" />
```

As you can see from the samples, you can do multiple `{}` notations in one value and you can even use nested values.

<Settings> Element

We will start off with the **Settings** Element. This element is used to define the framework settings, you can look at the schema to see which settings are required and which ones are optional ([Here is the generated Configuration File Documentation](#)). As a rule of thumb, I always define all of them and if I do not use the setting, the value is blank. Below is an overview table and then I will go into each setting.

Settings Chart

Setting	Description	Type	Required	Default
AppMapping:	The path of your application either relative from the web root or cfm mapping. Very Important to set this setting if using the coldbox proxy or remote operations, instead of letting the framework auto calculate it for you.	String	FALSE	[Auto Calculated]
AppName:	The unique name of your application.	String	TRUE	n/a
ApplicationStartHandler	The name of the onApplicationStart handler to run: ehGeneral.onAppStart, leave blank if not used. To trigger again this handler, you must reinitialize the framework: fwreinit=1	String	FALSE	blank
CustomErrorTemplate	ColdBox comes with its own error template for exceptions and bug reports. However, if you wish to customize your errors and bug reports, which you should, then just place the location of your custom error template. For example: includes/errorpage.cfm or /mymapping/templates/error.cfm Then in order to retrieve the error, you will need to get the Exception Bean from the request collection: getValue("ExceptionBean"). You can then use this bean to render your error page. Please look at the API to get a better understanding of the bean.	String	FALSE	blank
CustomEmailBugReport	ColdBox comes with its own template for bug reporting. However, you can override it and use your own. For example: includes/bugreport.cfm or /mymapping/templates/bugreport.cfm Then in order to retrieve the error, you will need to get the Exception Bean from the request collection: getValue("ExceptionBean"). You can then use this bean to render your error page. Please look at the API to get a better understanding of the bean.	String	FALSE	blank
ColdboxLogsLocation	The location of the ColdBox logs. Relative if using a directory within the application. Absolute location if outside of the application.	String (Path)	FALSE	logs
ConfigAutoReload	This is a flag mostly used during development. It will reload your config file settings, clear the cache, reset the app to first request state, on every request. Else you will have to manually reload the structures using fwreinit=1. Be careful when using this as performance decreases.	Boolean	FALSE	FALSE
DebugMode	Enable/Disable ColdBox debugging mode application wide.	Boolean	FALSE	FALSE
DebugPassword	The password you would like to use to go into debugmode. You will need to pass this string as a URL param combined with the DebugMode.	String	FALSE	blank
DefaultEvent:	The name of the event handler for the default event to run: General.dspHome	String	TRUE	n/a
DefaultLogLevel:	The default log level to use when logging to log files.0-fatal,1-error,2-warning,3-information,4-debug	numeric (0-4)	false	4
EventName	The name of the event variable to use in the URL/FORM, default is 'event'.	String	FALSE	event (or framework wide setting)
ExceptionHandler	The custom exception handler to run on all framework exceptions. You decide what to do, YOU HAVE THE POWER!	String	FALSE	blank
EnableBugReports:	Enable/Disable the emailing of bug reports to the emails in the BugReports element.	Boolean	FALSE	TRUE
EnableColdfusionLogging	Enable/Disable Coldfusion Error Logs.	Boolean	FALSE	FALSE
EnableDumpVar	Enable/Disable the use of the URL action to dump variables in debug mode.	Boolean	FALSE	TRUE
EnableColdboxLogging	Enable/Disable Coldbox Logging facilities.	Boolean	FALSE	FALSE
EventCaching	Enable/Disable Event Caching	Boolean	FALSE	TRUE
FlashURLPersistScope	The storage location of the flash RAM the framework uses. Available possibilities are session and client.	String	FALSE	session (or framework wide settings)

HandlersIndexAutoReload	This is a flag mostly used during development. ColdBox onApplication Start will read your handlers directory and store the names of the available handlers. When requests are made and handlers get instantiated, they are instantiated using the internal syntax. Thus, if you are developing and are adding handlers, with this flag set to TRUE, then ColdBox will reload the list. Else, you will have to manually reload the structures using the <code>fwreinit=1 url action</code>	Boolean	FALSE	FALSE
HandlersExternalLocation	This is the base instantiation path for an external location for event handlers. The framework will read all the handlers from that directory and register them as external events. This should be a dot formatted path: <code>myMapping.externalHandlers.blog</code>	String	FALSE	blank
HandlerCaching	This is useful to be set to false in development and true in production. This tells the framework to cache your event handlers. If you do not specify this setting or leave it blank, then the framework will use the default value of TRUE	boolean	FALSE	TRUE
IOCFramework	This setting is used if you will be using an IoC framework like coldspring or lightwire. The only possible values are "coldspring" and "lightwire"	String	FALSE	blank
IOCFrameworkReload	This setting is used to tell the framework to reload the entire IOC Framework on every request. Great for development.	Boolean	FALSE	false
IOCDefinitionFile	This is the path to the main configuration file for your IoC framework. This can either be relative to the root of your application or an absolute path.	String (path)	FALSE	blank
IOEObjectCaching	This boolean variable tells the IoC plugin to cache objects created by the IoC framework in the ColdBox Cache.	boolean	FALSE	FALSE
MessageboxStyleOverride	Whether you want to override the messagebox's css or not.	boolean	FALSE	FALSE
ModelsExternalLocation	The instantiation base path of the location of external model objects	string	FALSE	blank
ModelsObjectCaching	Setting that tells the bean factory to persist model objects or not. Great for development.	Boolean	FALSE	true
ModelsSetterInjection	Use setter injection alongside metadata injection	Boolean	FALSE	false
ModelsDebugMode	Logs model creation and injections	Boolean	FALSE	false
ModelsDICompleteUDF	The global name of the UDF to call after injections (if found in cfc)	string	FALSE	onCompleteDI
ModelsStopRecursion	A comma-delimited list of class names where the factory should stop recursion. Ex: <code>transfer.com.TransferDecorator</code>	string	FALSE	blank
MyPluginsLocation	Declare the custom plugins base invocation path, if used. You have to use dot notation. Example: <code>mymapping.myplugins, myapplication.customplugins.</code> When custom plugins are called, this invocation location will be pre-pended.	String	FALSE	blank
onInvalidEvent	The event to execute if an invalid event is detected by the framework. For example, if garbage is sent in the event variable, then the event registered here will fire instead.	String	FALSE	blank
OwnerEmail	The email that will be used to send all email communications by the framework.	String	TRUE	n/a
ProxyReturnCollection	A boolean flag that tells the ColdBox proxy to return the entire request collection structure or what the event handlers return	boolean	FALSE	FALSE
RequestStartHandler	The name of the <code>onRequestStart</code> handler to run: <code>ehGeneral.onRequestStart</code> , leave blank if not used.	String	FALSE	blank
RequestEndHandler	The name of the <code>onRequestEnd</code> handler to run: <code>ehGeneral.onRequestEnd</code> , leave blank if not used	String	FALSE	blank
RequestContextDecorator	The instantiation path to the request context decorator you would like to use.	String	FALSE	blank
ReinitPassword	The password you would like to use to reinitialize the framework using the <code>fwreinit</code> URL Action.	String	FALSE	blank
SessionStartHandler	The name of the <code>onSessionStart</code> handler to run: <code>ehGeneral.onSessionStart</code> , leave blank if not used.	String	FALSE	blank
SessionEndHandler	The name of the <code>onRequestEnd</code> handler to run: <code>ehGeneral.onSessionEnd</code> , leave blank if not used	String	FALSE	blank
UDFLibraryFile	The location of your UDF library if in use, else leave blank. ColdBox will first look in your <code>includes</code> directory, so you can just place the name of the UDF here, or the full path you write including CFMX Mappings. Ex: <code>/ColdBoxSamples/includes/udf.cfm</code>	String	FALSE	blank
ViewsExternalLocation	The external location to use for view renderings. This location will be checked second for views to render by using the <code>renderView()</code> commands or setting a view. This is a great way to declare a repository of views for rendering.	String	FALSE	blank

We will now go over each setting and describe what exactly they do, so you can get a better understanding of what switches to toggle on or off. In order to know if they are required or not or their default values, please refer to the chart above.

AppName

The first setting is **AppName**, this is just used for internal references and you can use it throughout your code using the `getSetting("AppName")` method.

AppMapping

This is an optional setting of the complete path to your application from either the web root or from a coldfusion mapping. If you do not use this setting, then ColdBox will try to auto-calculate the path to your application from the web root. Therefore, you should fill out the setting with your path to make sure that ColdBox can find and register your application for usage. This is the most important setting in your application, as this is how everything is calculated from.

Important Please make sure to use the AppMapping setting when using features such as the ColdBox Proxy

ApplicationStartHandler

The **ApplicationStartHandler** simulates the `onApplicationStart` method. You can use this to setup your application variables, initial configurations, etc. The cool thing is that in order to fire it again, you can just call it from any event handler method or you can fire it by using the framework `reinit` action parameter. `fwreinit=true`.

CustomErrorTemplate

The **CustomErrorTemplate** is a setting that simulates the `cfcerror` tag. However, you have the full Exception information stored in a bean (object) in the request collection that the framework places for you. In this setting you place the template to display on all errors. However, the template must be inside of the application root. Once inside the template, you can access the `ExceptionBean` by retrieving it from the request collection:

```
<cfset myException = Event.getValue( "ExceptionBean" )>
```

Then you can use the bean's methods to display the exception information in any way you want. See [CFC API](#). I truly recommend using this feature, since it enables you to present well formatted error displays to users, without showing all the stacktrace, etc.

CustomEmailBugReport

The **CustomEmailBugReport** is a setting to declare which template to use as the bug report for emailing it out if using the bug report feature. The template must be within the application or using cf mappings. Once inside the template, you can access the `ExceptionBean` by retrieving it from the request collection:

```
<cfset myException = Event.getValue( "ExceptionBean" )>
```

Then you can use the bean's methods to display the exception information in any way you want. See [CFC API](#).

ColdBoxLogsLocation

The **ColdBoxLogsLocation** is where you define the path to where you want to store the ColdBox generated log files. If you leave this setting blank and the **EnableColdboxLogging** setting is set to true, then ColdBox will create a directory called **logs** for you in the root of your application or whatever default you have placed in the framework's settings.xml file. You have two options when choosing a path:

- **Relative Path:** Set the folder where you will store the log files relative to your application root. So for example I have a directory under my application root called **logs** my value for this setting will be plain and simply **logs**
- **Absolute Path:** Let's say you want to store your log files in another location outside of your application root. Well you can! Just enter the full path in this value.
 - Unix Sample: /data/work/mywebsitelogs/myblog
 - Windows Sample: c:\applicationlogfiles\myblog

ConfigAutoReload

The **ConfigAutoReload** setting tells the framework to reload the handlers index but also all of the configuration settings, resource bundles, cleans the cache and it fires the Application Start Handler. Again, if you have errors due to your config, then reload it by using **thefwreinit=1** or set this flag to true for development. Be aware that the application will be reloaded in each request while this setting is turned on.

Note: Setting this setting to TRUE on production environments is not recommended. For performance issues.

DebugMode

The **DebugMode** setting is used to declare wether your entire application will show the ColdBox debugging information or not. This is a boolean setting. You can also activate the ColdBox debug mode for a single session by using the url parameters. See [ColdBox's URL Actions](#)

The debug cookie expires when the user closes the browser, that is, the cookie is "session only"

DebugPassword

This setting is used to set the debug mode on or off via the [ColdBox's URL Actions](#).

DefaultEvent

In this setting, we go a foot deeper into implicit invocations and arrive at the **DefaultEvent** Setting. This is where you declare your event handler and method combination that will be invoked as the first request to the system when no event is defined in the request collection. This follows the ColdBox lifecycle, in which the first request to the application, where no event has been defined by either url or form post, the framework will execute this event. That is why you can go to index.cfm and the default event will be fired. Remember that the value must be in event handler syntax.

For a more in depth look at event handlers, look at the [Event Handlers Guide](#). So basically you are declaring what event to fire when no event has been defined or its the user's first request.

DefaultLogLevel

This setting allows for user customization of what errors/entries to log by setting it to an integer between 0 and 4. The default log level the logger uses is **4**, which means it will log EVERYTHING. Please see the severity log levels table below:

severity	log level
debug	4
information	3
warning	2
error	1
fatal	0

EventName

The event name is the variable used to fire events in the framework. The default variable is **event**, but you can choose your own such as **do,fa,e**. If you do not set a variable in your config, the framework will fall back to the default value set in the **settings.xml** file, which is the configuration file for the framework. The default shipped variable name is **event** but you can also change the framework wide one.

ExceptionHandler

The **ExceptionHandler** setting is used to define the event handler method combination that will take care of your exceptions. You can use this to filter out certain exceptions, relocate to certain events, log the errors, etc. Please note that if this setting is defined, then ColdBox will not automatically log the errors for you, you will have to do it yourself. How? Well, an **ExceptionBean** is placed in the request collection for you to determine what to do with the exception and the **CustomErrorTemplate** is for you to determine how to display the exception.

EnableBugReports

The **EnableBugReports** setting is a boolean flag that tells ColdBox whether to email bug reports to the emails declared in the **BugTracerReports** sections, or not. So if you do not want to send bug reports on every exception, then you **MUST** set this flag to false.

EnableColdfusionLogging

The **EnableColdfusionLogging** setting is a boolean setting that tells the framework to log all types of errors and exceptions in the ColdFusion logging facilities using **cflog**. ColdBox will create a new log file named after the **AppName** setting value. However, as you all know, we do not always have access to the coldfusion log locations and we cannot change their locations. Thus, the introduction of ColdBox's own logging facility.

EnableDumpVar

The **EnableDumpVar** is a pretty cool feature that you can enable or disable. You can use the **DumpVar** parameter in your url in order to well, dump variables at the end of the request. **However, you can only use this feature if you are in the ColdBox debug mode.** You can send a variable name, scope, etc. or even a comma delimited list of variables to dump. Example:

http://myapp.com/index.cfm?dumpvar=myvariable,session

ColdBox will dump the the myvariable variable, and the session scope. **It is recommended to have this setting set to FALSE in production**

EnableColdboxLogging

The **EnableColdboxLogging** setting is a boolean setting that tells the framework to log all types of errors and exceptions in the location specified by the **ColdBoxLogsLocation** element (Discussed below). You can then use the **logEntry()** method found in the **logger** plugin, to log entries to this facility. You can also use the **logError()** and **logErrorWithBean()** methods, also found in the logger plugin. This is ColdBox's AOP Error Logging. See [CFC API](#)

So now you have full control of your logging per application. Not only that, using the settings.xml of the framework, you can control the maximum size in kbytes for each log file. Once this limit is reached, ColdBox auto-archives your log files and creates a new archive. You can choose the file encoding of the log files and the maximum number of archives to keep in rotation. Well ladies and gentlemen, isn't that easy. Two values to fill out and you have logging supported in your application. You can then use the following methods to interact with the logs. However, I encourage you to view the full API of the logger plugin to understand the full power you have.

- **logErrorWithBean(** exceptionBean)
- **logError(** Message, [ExceptionStruct (Coldfusion cfcatch structure)], [ExtraInfo (Anything you like to log extra)])
- **logEntry(** Severity=['error', 'information', 'warning', 'fatal'], Message, [ExtraInfo (Simple Value extra to log)])

EventCaching

A simple boolean variable that tells the framework to activate event caching or not.

FlashURLPersistScope

This is a simple setting that tells the framework where to store the flash RAM data it uses. The only possible options right now are:

- session
- client

If you do not have this setting defined, the framework will look for the default in its internal settings.

HandlersIndexAutoReload

The **HandlersIndexAutoReload** is a developer tool to facilitate the developer from constantly using the **fwreinit=1** action flag. When ColdBox initialized the application, it introspects your handlers for metadata. It then creates an internal registry of each handler. This way, only registered handlers can be called, and with the correct syntax for any operating system. However, when you are developing, you are constantly adding/removing/editing methods inside of these handlers or even adding new handlers. That is why you can get the Event Handler Not registered exceptions. So set this flag to true, to reload the index on every request. This is done incredibly fast, so don't worry about performance hits on development.

Note: Setting this setting to TRUE on production environments is not recommended. For performance issues.

HandlersExternalLocation

This to me is the most interesting aspect of ColdBox, EXTENSIBILITY. In this element you define the base invocation path to your external handlers repository. You can package them as you want or just point to a specific package. The location will be registered and all found event handlers will be parsed and registered as external events. This means that if an event is already registered in the local location, that one will take precedence over the external location. This is a great setting for you to develop reusable handlers where one update affects all events throughout.

HandlerCaching

This is useful to be set to **false** in development and **true** in production. This tells the framework to cache your event handlers. If you do not specify this setting or leave it blank, then the framework will use the default value of **TRUE**

IOCFramework

Here is where you can declare which IoC framework you would like to use in your application: **coldspring** or **lightwire**. Those are the only two options as of now. If you are not using an IoC framework, then omit this setting or leave it blank. You can then use the framework via the **ioc** plugin. [Please look at the API for more information.](#)

IOCFrameworkReload

Are you in development and have to be constantly hitting the reinitt button because you made changes to your ioc services? Look no further. This setting will auto-reload the ioc factory on each development request so you know that you are getting the latest code changes produced for you. Of course, the default value is **false**.

IOCDefinitionFile

This is the location of your IoC configuration file. You can use either a relative or absolute path. However, if the path is invalid, the framework will throw an exception. If you set this value, but not the IOCFramework setting, then this setting is omitted. If you use **Lightwire** then this setting is the instantiation path to your config bean or a valid coldspring configuration file. For more information look at the Lightwire sample application and guide.

IOCObjectCaching

This setting tells the IoC plugin to actually cache the objects created by the IoC framework in the ColdBox cache. This is a great way for ColdBox to do the created object's persistence for you in a dynamic way. By using the IoC plugin's **getBean()** method, you will be retrieving from the object cache and not creating an object again. Not only do you need to set this setting to true for the objects to get cached, but you need to add coldbox cache metadata to their **cfcomponent** tags in order for them to be cached. By default, the ioc plugin will NOT cache objects, unless their metadata specifies it. See an example below:

Important: This setting is for advanced users only. You will be potentially messing with object persistence.

```
//Cache the userService indefinitely by setting a 0
<cfcomponent name="UserService" output="false" cache="true" cachetimeout="0">

//Cache my mail service for 20 minutes
<cfcomponent name="MailService" output="false" cache="true" cachetimeout="20">

//Cache my bugservice for 5 minutes
<cfcomponent name="bugService" output="false" cache="true" cachetimeout="5">

//Do not cache my cheapService
<cfcomponent name="cheapService" output="false" cache="false">
```

MessageboxStyleOverride

The **MessageboxStyleOverride** setting is used in conjunction with the messagebox plugin. This is a plugin that actually builds a visual representation of a message box for errors, warnings, and informational purposes. Since it is a visual component, you can skin it using this setting. By setting this boolean setting to true, you are saying to the plugin that the css will be available already.

ModelsExternalLocation

This to me is the most interesting aspect of ColdBox, EXTENSIBILITY. In this element you define the base invocation path to your external models repository. You can package them as you want or just point to a specific package. The location will be registered and all found model objects can be used. This is a great setting for you to develop reusable model objects. Please note that model objects by convention take precedence over external model objects with the same name. We highly encourage you to use [Model Object Mappings](#) to create distinctions.

ModelsObjectCaching

Tells the bean factory to cache model objects if cache metadata is found. This is great for development only as you are making every model object get recreated.

ModelsSetterInjection

This tells the bean factory to use setter injection alongside metadata injection. If you prefer not to use the metadata injections, then you can use the old method of setter injection.

ModelsDebugMode

If turned on, the bean factory will log all model creation and injections to the logging facilities.

ModelsDICompleteUDF

The global name of the UDF to call after injections (if found in cfc). Basically, after all dependency injection has been done on a target model object, the bean factory will look for a UDF with **this** name in the cfc. If found, it will call it. This is great for post dependency injection processing.

ModelsStopRecursion

A comma-delimited list of class names where the factory should stop recursion. Ex: transfer.com.TransferDecorator It stops recursion so no dependencies will be evaluated from that inheritance class.

MyPluginsLocation

Again, EXTENSIBILITY. You can create your own plugins according to the [ColdBox Plugin Specification](#) and reuse them in your applications. In this element you define the base invocation path to your custom plugins. So for example you have all your custom plugins under a coldfusion mapping called `coldboxcustomplugins` and inside that mapping you have organized all your plugins into packages and so forth. On this element you would just define `coldboxcustomplugins` as the value. Then you would use ON-DEMAND these plugins by using the `getMyPlugin('plugin_path')` method.

```
<cfset myFeed = getMyPlugin( "utilities.rss.agggregator" ).parseFeed(rc.feedURL) >
```

or

```
<cfset myFeed = getPlugin(plugin= "utilities.rss.agggregator" ,customPlugin= true ).parseFeed(rc.feedURL) >
```

You can also define the base path without using coldfusion mappings. However, you must make sure that the path is correct. Remember, this is the base instantiation path to your plugin.

onInvalidEvent

This is a quick way to declare what to do if an invalid event is detected. What is an invalid event? An invalid event is an event that does not exist in your application. So here is where you declare what event to fire instead of the garbage event. I usually set this to my default event. If you do not set this setting, then the framework will throw an invalid event exception.

OwnerEmail

The **OwnerEmail** setting is used to declare the email address that will be used to send email notifications from the framework to the outside world.

ProxyReturnCollection

The **ProxyReturnCollection** is a boolean setting used when calling the ColdBox proxy's process() method. If this setting is set to true, the proxy will return back to the remote call the entire request collection structure. If set to false, it will return, whatever the event handler returned.

RequestStartHandler

The **RequestStartHandler** is where you declare the event handler and method combination that will be invoked at the start of every user request. This simulates the Application.cfc onRequestStart method, but enclosed within the framework. You can use this for security, variable settings, etc. From within a request start handler method, you can call other events, override incoming events, filter events, etc.

RequestEndHandler

The **RequestEndHandler** is where you declare the event handler and method combination that will be invoked at the end of every user request. This simulates the onRequestEnd method, but enclosed within the framework.

RequestContextDecorator

The **RequestContextDecorator** is used to decorate the framework's event context object. This is an advanced setting and please read the [Request Context Decorators Guide](#) for an in depth view of this setting. The value for this setting, should be the instantiating path to your decorator.

```
<Setting name= "RequestContextDecorator" value= "myApp.model.myRequestContext" />
```

ReinitPassword

The reinit password is a new setting since version 2.0 and it helps you set a reinitialization password that works in conjunction with the fwreinit=1 URL Action. This means that if you set this setting, you need to pass it via the URL or FORM to reinitialize the framework. [ColdBox's URL Actions](#).

```
fwreinit=myspass
```

SessionStartHandler

The **SessionStartHandler** is where you declare the event handler and method combination that will be invoked at the start of every user session. This simulates the Application.cfc onSessionStart method, but enclosed within the framework. You can use this for security, variable settings, etc.

SessionEndHandler

The **SessionEndHandler** is where you declare the event handler and method combination that will be invoked at the end of every user session. This simulates the onSessionEnd method, but enclosed within the framework. You will receive a copy of the session structure in the request collection called **sessionReference**

UDFLibraryFile

The **UDFLibraryFile** is a cool setting where you can declare the path to a UDF template to load for usage in views, layouts and event handlers. You can use relative paths to your applications root or ColdFusion mappings. So for example if I have my udf template called udf.cfm in my includes directory under my application root, I would use: `value="includes/udf.cfm"`, if I have them in a mapping called: myudfs, then I would use: `value="/myudfs/udf.cfm"`. Your very own runtime method injection mechanism. You can also do runtime mixins by using the `includeUDF()` method in your handlers/plugins/interceptors/views/layouts.

ViewsExternalLocation

This setting determines a location of where you want to have external views. So when you do a renderView() or setView() methods, the framework will search first your default convention for the view, if it cannot find it there, it will look in this external location and render it for you. This is a great way to create modules or extensible applications.

<YourSettings>

This element is used by the developer to set any values he/she would like to use in the application. This can be configuration settings, etc.

```
<YourSettings>
  <Setting name= "BlogDSN" value= "my_blog" />
  <Setting name= "Maintenance_mode" value= "true" />
  <Setting name= "Publish_emails" value= "lmajano@gmail.com" />
  <Setting name= "MyArray" value= "[1,2,3,4,5]" />
  <Setting name= "MyStructure" value= "{ 'owner': 'Luis Majano', 'smart': 'false', 'funny': 'maybe' }" />
</YourSettings>
```

As you can see above, I use it to declare a datasource, a flag for maintenance, a variable to hold my publish_emails, a custom number array and a structure.

Complex Variable Syntax: JSON

All complex variable syntax is based on JSON notation. So you can go crazy and do all kinds of embedded structures, queries, etc. An important aspect in the configuration

file is that instead of using double quotes (") for your JSON separators, you will have to use single quotes ('). The framework will then convert them to double quotes once it deserializes them. For an in-depth tutorial in JSON syntax, please visit this guide: <http://www.json.org/>

How Do I Use Them?

You can then use all these settings in your applications by simply calling the `getSetting("setting").{key}` Method. You can even change the settings that are loaded in memory using the `setSetting("key", {value})`. This element is a great way to securely store your applications configurations.

<Conventions>

This element defines custom conventions for your application. By default, the framework has a default set of conventions that you need to adhere too. However, if you would like to implement your own conventions for a specific application, you can use this setting:

```
<Conventions>
  <handlersLocation> _handlers </handlersLocation>
  <pluginsLocation> _plugins </pluginsLocation>
  <layoutsLocation> _layouts </layoutsLocation>
  <viewsLocation> _views </viewsLocation>
  <eventAction> go </eventAction>
  <modelsLocation> model </modelsLocation>
</Conventions>
```

- **handlers location** : This element is to declare where your handlers are stored within the application root. In my sample its the directory `_handlers`
- **plugins location** : This element is to declare where your custom plugins are stored within the application root. In my sample it's the directory `_plugins`
- **layouts location** : This element is to declare where your layouts are stored within the application root. In my sample it's the directory `_layouts`
- **views location** : This element is to declare where your views are stored within the application root. In my sample it's the directory `_views`
- **event action** : This element defines what is the default event action an event handler will use if an incoming event has no defined action. If there is no action then the framework will look for this action in the handler and execute it.
- **models location** : This element is to declare where your model objects are stored within the application root. In my sample it's the directory `model`

<DebuggerSettings>

The debugger settings element is used to control how the ColdBox debugger works. Below are the main element that you will interact with.

- **PersistentRequestProfiler** : This setting is a boolean setting that you use to tell the debugger to persist request profilers across requests.
- **maxPersistentRequestProfilers** : This is the number of request profilers to keep stored in the debugger.
- **maxRCPanelQueryRows** : This is how many rows to dump when dumping queries.

The following are elements that have attributes that you can use to help you render the debugger.

- **TracerPanel** : The panel that shows you all the code tracers
 - @show : boolean (Actually render the panel)
 - @expanded : boolean (Have the panel expanded or not by default)
- **InfoPanel** : The panel that shows you all request information and request profiler
 - @show : boolean (Actually render the panel)
 - @expanded : boolean (Have the panel expanded or not by default)
- **CachePanel** : The panel that shows you the cache statistics
 - @show : boolean (Actually render the panel)
 - @expanded : boolean (Have the panel expanded or not by default)
- **RCPanel** : The panel that shows you the request collection at the end of the request.
 - @show : boolean (Actually render the panel)
 - @expanded : boolean (Have the panel expanded or not by default)

<MailServerSettings>

This element is used to declare the mail server, username and password to use for the application. This is an optional element, due to the fact that if it is not declared, then ColdBox will send emails out using the values in the ColdFusion administrator.

```
<MailServerSettings>
  <MailServer> mail.server.com </MailServer>
  <MailPort> </MailPort>
  <MailUsername> myusername <MailUsername>
  <MailPassword> mypassword <MailPassword>
</MailServerSettings>
```

In the MailServer element you declare the dns or ip address of the mail server. The default MailPort is 25, but you can override it here. MailUsername is the username to use to log in to the mail server defined. MailPassword is the Mail Password, duh!

<BugTracerReports>

This element works in conjunction with the `EnableBugReports` setting in the first tutorial. You must have this setting flag enabled in order for ColdBox to send you any bug report. In this element you are going to declare all the email addresses that will receive bug reports by using the BugEmail element.

```
<BugTracerReports>
  <BugEmail> cfcoldbox@coldbox.com </BugEmail>
  <BugEmail> joe@coldbox.com </BugEmail>
  <BugEmail> luis@gmail.com </BugEmail>
</BugTracerReports>
```

<DevEnvironments>

* This setting will be deprecated in 2.7 to favor the environment interceptor.

Since we all work in development environments, a common setting in development might be different in production. Thus, the nature of this element. In this element you will declare url snippets that the framework will test at initialization. Once a match is produced, the frameworks `Environment` setting is set to `DEVELOPMENT` or `PRODUCTION` if none are found. This is a great way for you to declare your development urls, not the full url just parts of it, and ColdBox sets the correct internal environment.

```
<DevEnvironments>
  <url> dev </url>
  <url> localhost </url>
  <url> mydev </url>
</DevEnvironments>
```

In the example above, I set three url fragments that the framework will test for: `dev`, `localhost`, `mydev` If at least one of them is found in the `cgi.http_host`, then ColdBox will set the `Environment` setting to `DEVELOPMENT`, else to `PRODUCTION`. I also recommend you use an environment interceptor for your environment settings.

<WebServices>

Since we are now in a service oriented web architecture, this was a great way to declare all the web services my application that will be used. Not only that, I can declare two url's for each webservice, one for development and one for production. ColdBox reads these elements and places them in the configuration structure. You can then use the webservices plugin to get the webservice's url, get a webservice object already instantiated or refresh the web services's stubs. So for example if I call `getWS("luis")`, ColdBox will retrieve the correct web service URL according to the environment I am on, `DEVELOPMENT` or `PRODUCTION`. I no longer have to create tons of if's statements. ColdBox does it for me by sensing its environment.

* The devURL will be deprecated by version 2.7 to favor the environment interceptor

```
<WebServices>
  <WebService name="newsWS" DevURL="http://devurl.com/newsWS.cfc?wsdl" URL="http://prourl.com/newsWS.cfc?wsdl" />
  <WebService name="emailValidator" URL="http://someip.com/validator.cfc?wsdl" />
</WebServices>
```

In the second web service declaration you see that there is no DevURL attribute defined. Well, that is an optional attribute. If not used, then ColdBox will use the url in the URL attribute for both DEVELOPMENT and PRODUCTION.

<Layouts>

The layouts element is used to declare the Default Layout that all views will be rendered in and to declare other Layouts and their view/folder combinations. This is the core of the ColdBox layout manager capabilities. For an in-depth guide, please visit the [layouts and views guide](#).

```
<Layouts>
  <DefaultLayout> Layout.Main.cfm </DefaultLayout>
  <DefaultView> home </DefaultView>
  <Layout name="popup_black" file="Layout.popup_black.cfm" >
    <View vwEmails </View>
    <View vwPhones </View>
    <Folder> Tags </Folder>
  </Layout>
  <Layout name="print" file="Layout.print.cfm" >
    <View vwContactBook </View>
  </Layout>
  <Layout name="blog" file="Layout.blog.cfm" >
    <Folder> blog </Folder>
  </Layout>
</Layouts>
```

As you can see from the code above, the **DefaultLayout** element is mandatory and you define the name of the file. This file will need to be in the layouts directory under your application root. You can then continue to declare multiple Layout elements to declare how other views will be rendered by the framework. You can see the first Layout declaration has a name of "popup_black" and a file of "Layout.popup_black.cfm". It then defines two child View elements: vwEmails, vwPhones and **Folder** Element called Tags. This tells the framework that the views: vwEmails, vwPhones and all the views inside of the folder tags will be rendered in this specific layout.

The Folder element is used to declare that all views inside that folder and within will be tied to a specific layout. Please note that a Folder can only exist within one layout only.

The **DefaultView** element is optional and can be used to tell the framework to always render this view (located in the views directory or your convention) when the event handler does not set a view via the `event.setView(view)` method. This is a great way to prototype websites. See Adam Fortuna's [Include Views with ColdBox](#) as a sample of why to use this setting.

Important Note: All view declarations must NOT include the extension. ColdBox automatically appends the .cfm extension.

This information gets loaded in the configuration structure of ColdBox, so when an event handler sets any of these views, it will automatically render them in the declared layout and not in the default layouts. So if I have in an event handler: `setView("vwEmails")` ColdBox will render the vwEmails view in the Layout.popup_black.cfm template. You can have as many layouts as you want. And also remember that you can override a view's layout by programmatically setting it in your event handlers. A perfect example of this feature is if you want to provide a cfdocument capability throughout your entire site, for any view, you simply can override the **DefaultLayout** of the view by using the `setLayout("template")` function. This tells ColdBox to render the view in the layout that was just set and not in the one in the configuration structure. The possibilities are endless.

<i18N>

ColdBox has **FULL INTERNATIONALIZATION SUPPORT** built in thanks to Paul Hastings. It makes use of java resource bundles and java locales. If you need more information about internationalization, please visit the following sites:

- [i18Gurus](#)
- [Open Source Listing of i18N by Brian Rinaldi](#)

Now, some people might not be that familiar with this concept, I am not an expert, so I tried to make ColdBox give applications i18N support without a big headache or a lot of configuration. So basically the way to add i18N support for a ColdBox application is to fill out the following configuration (3 LINES ONLY!!!) and create your resource bundles with your key-value pairs using UTF-8 encoding if you are using translations. At the end of the post is a resources section that can help you out even more.

```
<i18N>
  <DefaultResourceBundle> includes /main </DefaultResourceBundle>
  <DefaultLocale> en_US </DefaultLocale>
  <LocaleStorage> session </LocaleStorage>
  <UnknownTranslation> UNKNOWN </UnknownTranslation>
</i18N>
```

Default Resource Bundle

In the first element: **DefaultResourceBundle** you declare the location of your resource bundles without the locale part: **includes/main**. This is not mandatory and can be left blank if your application will only use the i18n locales and methods, no translations. If used, this tells ColdBox that in the includes directory there will be a file that starts with *main* in the includes directory.

Default Locale

The second element **DefaultLocale** is used to define the default locale to use: **en_US**. Please remember that this is using standard java locale syntax. Some examples are *es_SV*, *es_DO*

This settings gets appended to the first element (default resource bundle) to define the default file to load: **includes/main_en_US.properties**. The text in italics is what is defined in the xml, the rest is appended by ColdBox. So make sure your files are in the following format:

```
{name}_{java standard locale}.properties
```

Locale Storage

The third element is **LocaleStorage** and you have three possibilities:

- session
- client
- cookie

This tells ColdBox in what scope to store the locale of each user, not THE APPLICATION. This way, you can have a users running the application in different locales (languages)

Unknown Translation

The fourth element is **UnknownTranslation** which is used by the resource bundle plugin whenever a translation cannot be found. Instead of returning the default bogus **_UNKNOWN** you can choose your own.

There is no big deal about resource bundles, they are files that have key-value pairs, look at the example below:

```
help_button=Help
close=<a href="javascript:window.close()">Close Me</a>
relocate_button=Relocate Now!
intro_message=Welcome to my website.
search_button=Search It!
search_test=<i>Search for postings now.</i>
```

So how do I change from one locale to another? look at the sample below:

```

<--- Then use the following in any of your event handlers --->
<--- rc.thisLocale is a form variable holding the locale --->
<cfset getPlugin( "i18n" ).setfwLocale(rc.thisLocale) >

<cfset getPlugin( "i18n" ).setfwLocale(getvalue( "newlocale" ))>

<cfset getPlugin( "i18n" ).setfwLocale( "en_US" )>

```

i18N Resources

- Paul Hastings cfe: <http://www.sustainablelegis.com/unicode/resourceBundle/rb.cfm>
- For ISO Language Code information look at: <http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>
- For ISO Country Code information look at: http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html
- For locale information look at <http://java.sun.com/j2se/1.4.2/docs/guide/intl/locale.doc.html>
- A free open source java translation editor <http://attesoro.org/>

<Datasources>

The datasources element is very useful when defining datasources for your application. You can then retrieve the settings via the `getDataSource("alias")` method. This method will create a datasource bean with the appropriate information.

```

<Datasources>
<DataSource alias= "mysite" name= "MyDSN" dbtype= "mysql" username= "" password= "" />
<DataSource alias= "blog" name= "MyBlog" dbtype= "oracle" username= "" password= "" />
</Datasources>

```

<Cache>

The cache element is used to configure the ColdBox object cache system. These settings will override the framework-wide settings. See [ColdBox Cache](#)

```

<Cache>
<ObjectDefaultTimeout> 20</ObjectDefaultTimeout>
<ObjectDefaultLastAccessTimeout> 5</ObjectDefaultLastAccessTimeout>
<UseLastAccessTimeouts> true</UseLastAccessTimeouts>
<ReapFrequency> 1</ReapFrequency>
<MaxObjects> 0</MaxObjects>
<FreeMemoryPercentageThreshold> 0</FreeMemoryPercentageThreshold>
<EvictionPolicy> LFU</EvictionPolicy>
</Cache>

```

Object Default Timeout

The timeout in minutes in which the objects will live. After object goes beyond the timeout, they will be removed from the cache.

Object Default Last Access Timeout

The timeout in minutes in which the objects have not been accessed then they will be removed from the cache.

Use Last Access Timeouts

This setting is a boolean setting that tells the cache to use the expiration algorithms that use a default last access timeout. The default setting is true.

Reap Frequency

The frequency in minutes, in which the cache manager will reap the cache for expired objects. For example, if I set this to 5 (minutes). Then whenever a request comes in and the 5 minutes have passed, the cache manager will reap the cache. So reaping occurs at a frequency of at least 5 minutes.

Max Objects

The number of maximum objects for the cache manager to cache.

JVM Free Memory Percentage Threshold

This tells the cache manager the percentage amount needed of free memory in order to cache objects. If the JVM's free memory goes below this threshold, then the cache manager stops caching objects.

EvictionPolicy

The only possible eviction policies as of now are:

- LFU Least Frequently Used
- LRU Least Recently Used
- FIFO First In First Out

The default policy is LFU. This means that the object with the least usage will be purged from the cache if space is needed. However, if LRU is used, then the object that was last used will be purged.

<Interceptors>

The interceptors element is where you declare the interceptors your application will use. Please see the [Interceptor's Guide](#) for an in-depth view of interceptors.

Important: The order of declaration is very important, since if they contain the same execution points, they will be chained in the order they are declared

So let's look at a sample:

```

<Interceptors throwOnInvalidStates= "true" >
<CustomInterceptionPoints> comma-delimited list </CustomInterceptionPoints>
<Interceptor class= "full class name" >
<Property name= "myProp" >value </Property>
<Property name= "myArray" >[1,2,3] </Property>
<Property name= "myStruct" >{ key1:1, key2=2 } </Property>
</Interceptor>
<Interceptor class= "no property" />
</Interceptors>

```

throwOnInvalidStates Attribute

The main `Interceptors` element has one boolean attribute called `throwOnInvalidStates`.

- The default is `true`
- This tells the interceptor service to throw an exception if the state announced for interception is not valid or does not exist.

CustomInterceptionPoints

The CustomInterceptionPoints element is a comma delimited list of custom interception points you will be registering for execution. This is the way to provide an observer-observable pattern to your applications. Again, please see the [wiki:cbInterceptorsGuide Interceptor's Guide] for more information. Just note that here is where you declare the custom interception points separated by a comma if more than one.

Interceptor Element

This is the element used to declare an interceptor for execution.

@Class Attribute

The `class` attribute is the instantiation path for this interceptor, *example: coldbox.system.interceptors.ses*

Property Elements

Each interceptor can have zero, one or more property elements and they can be simple or complex value, using the complex syntax used for the settings shown above. It contains one attribute: `@name` which is the name of the property.

Related Guides

- [The ColdBox Configuration File](#)
- [Event Handlers](#)
- [The ColdBox Request Context](#)
- [Layouts & Views](#)
- [URL Mapping & Rewriting](#)
- [Model Integration](#)
- [ColdBox Plugins](#)
- [ColdBox Interceptor](#)
- [Extending The Request Context](#)
- [ColdBox Unit Testing & Integration Testing](#)
- [ColdBox Proxy: Powering Flex/AIR/Remote Applications](#)
- [ColdBox-Ajax Integration](#)

Categories:

- [level-beginners](#)