

[<< Back to Dashboard](#)

# ColdBox's Plugins Guide

Covers up to version 3.5.0

## Introduction

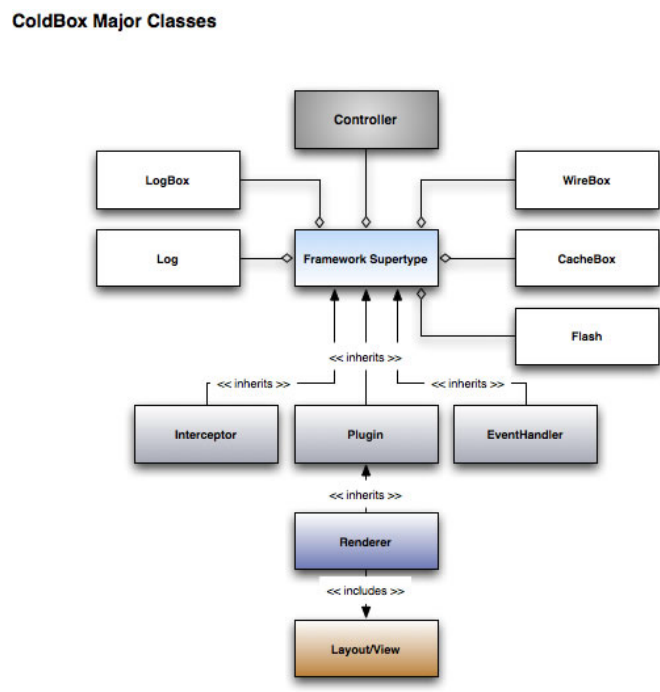
Welcome to the ColdBox plugins guide. This guide will give you a brief overview of what ColdBox plugins are, what are the core plugins, how you can extend them and how you can create your own custom plugins. This guide expects that you at least have a working knowledge of ColdBox and some of its internal workings.

## ForgeBox Plugins

Please remember that in our ColdBox developer community site, [ForgeBox](#), you can find a big collection of [Plugins](#) and so much more. Anybody can contribute and automatically all contributions will be accessible via our ColdBox application templates and via ColdFusion Builder.

## What Are Plugins?

Plugins are CFC's that are built to extend the core framework or application functionality without hindering system performance. The ColdBox Platform contains several useful plugins that not only makes ColdBox a framework, but also a developer toolkit. The idea behind the plugin concept is to create a reusable and extensible architecture for developers to build applications or framework specific tasks that could be easily shared and loaded on demand within your applications and keeping true to the framework's regulations and part of the framework life cycle.



This means that you can easily code plugins, completely extend your application and even share them with your friends. You can create as many plugins as you want and build a plugins' library that can be used in your applications by just specifying it on the configuration file or by using the [conventions](#). You will then be able to get the custom plugin and use it on any of your ColdBox applications. Not only can you use them, ColdBox can maintain persistence for you and even help you do dependency injection via [WireBox](#).

Any CFC can automatically be a Plugin if dropped in the conventions folder.

## Core Plugins

In this section I will cover the core plugins that come pre-bundled with ColdBox and have a brief explanation of each. In the next sections we will cover usage and techniques:

## Contents

- [ColdBox's Plugins Guide](#)
  - [Introduction](#)
    - [ForgeBox Plugins](#)
  - [What Are Plugins?](#)
  - [Core Plugins](#)
  - [How do I use plugins?](#)
    - [Injecting Plugins](#)
    - [Code Samples](#)
  - [Custom Plugins](#)
    - [Conventions Location](#)
    - [External Location](#)
    - [Full Path Location](#)
    - [Injecting Custom Plugins](#)
  - [Anatomy Of A Plugin](#)
    - [Component Declaration](#)
    - [Persistence](#)
  - [Extending/Overriding Core Plugins](#)

Plugin	Description	Persistence
<a href="#">AntiSamy</a>	Allows you to use the <a href="#">OWASP</a> anti forgery and XSS cleanup library in your ColdBox applications	Singleton
<a href="#">ApplicationStorage</a>	A facade to the <i>application</i> scope with some extra functionality	Cached With Default Timeouts
<a href="#">BeanFactory</a>	A facade to <a href="#">WireBox</a> so you can leverage object retrievals or populations	Singleton
<a href="#">ClientStorage</a>	A facade to the <i>client</i> scope. It will also take care of json serializations for complex variables for you and some extra goodies	Cached with Default Timeouts
<a href="#">ClusterStorage</a>	A facade to the Railo <i>cluster</i> scope. It will also take care of json serializations for complex variables for you and some extra goodies	Cached with Default Timeouts
<a href="#">CookieStorage</a>	A facade to the <i>cookie</i> scope. It will also take care of json serializations for complex variables for you. It can also use encryption and other goodies	Cached with Default Timeouts
<a href="#">DateUtils</a>	A useful collection of date related utility methods	Cached with Default Timeouts
<a href="#">FeedGenerator</a>	Can generate RSS feeds with true timezone, itunes, podcast and so much more support.	Cached with Default Timeouts
<a href="#">FeedReader</a>	A component that enhances ColdFusion's <i>cffed</i> tag to read RSS/ATOM feeds with true timezone support. It can also cache the parsing results in the ColdBox cache or files, with expiration policies and more.	Cached with Default Timeouts
<a href="#">FileUtils</a>	A useful collection of file related utility methods	Cached with Default Timeouts
<a href="#">HTMLHelper</a>	Allows you to build uniform HTML elements with ORM binding capabilities, data capabilities, HTML5 integrations, asset management and so much more.	Singleton
<a href="#">i18n</a>	Allows you to track user locales in the application and also help you with a tremendous amounts of functions when dealing with localization and internationalization	Singleton
<a href="#">IOC</a>	This plugin allows your application to talk and use third party dependency injection frameworks like ColdSpring and LightWire	Singleton
<a href="#">JavaLoader</a>	A jewel by Mark Mandel to allow you to easily integrate with third party java libraries or classes	Singleton
<a href="#">JSON</a>	A JSON utility that allows you to serialize/deserialize JSON with some extra capabilities	Cached with Default Timeouts
<a href="#">JVMUtils</a>	A useful collection of JVM related utility methods	Cached with Default Timeouts
<a href="#">Logger</a>	Allows you to talk to global application <a href="#">LogBox</a> logger object for useful logging.	Cached with Default Timeouts
<a href="#">MailService</a>	Allows you to abstract the usage of <i>cfmail</i> by giving you an OO approach to sending email with token replacement, different mail protocols and so much more. Why settle for the basics!	Cached with Default Timeouts
<a href="#">MessageBox</a>	A UI plugin to help render message boxes on your applications for errors, warnings, and informational messages. It can store its values in ColdBox's <a href="#">FlashRAM</a> so they can survive relocations.	Cached with Default Timeouts
<a href="#">ORMService</a>	A facade to our ColdBox enhanced <a href="#">ORM Services</a> that will allow you to leverage a virtual service layer for ORM entity operations	Singleton
<a href="#">QueryHelper</a>	A utility helper to sort, filter, append, union, join queries on the fly	Cached with Default Timeouts
<a href="#">Renderer</a>	ColdBox utilizes this plugin for all layout and view renderings	Transient (NOT CACHED)
<a href="#">ResourceBundle</a>	Allows you to talk to language resource bundles in your application and be able to switch and load according to user's locales	Singleton
<a href="#">SessionStorage</a>	A session scope manager and facade with some extra functionality	Cached with Default Timeouts
<a href="#">Timer</a>	A developer gem, that can help you time code and send it to the ColdBox debugger	Cached with Default Timeouts
<a href="#">Utilities</a>	A great plugin to help on all kinds of system utility functions	Cached with Default Timeouts

<a href="#">Validator</a>	A collection of useful validation methods	Cached with Default Timeouts
<a href="#">Webservices</a>	A library to help you get per tier WSDL's and Web Service Objects, Refresh Stubs and more.	Cached with Default Timeouts
<a href="#">XMLParser</a>	Our very own XML serializer for objects and data	Cached with Default Timeouts
<a href="#">Zip</a>	A nice wrapper to some java zip and unzipping utilities	Transient (NOT CACHED)

These are the core plugins that are shipped with ColdBox and as you can see they are highly configurable and will help you build your applications faster. What is also important to note, is that you can extend these plugins and create your own versions of them. I know I have!! The possibilities are endless.

## How do I use plugins?

The easiest way to use core plugins in your handlers/layouts/views and interceptors is via one method `getPlugin()` which is dissected below:

```
getPlugin([string plugin], [boolean customPlugin="false"], [boolean newInstance="false"], [module=''])
```

### Arguments:

- **plugin**: string : The name of the plugin to create or retrieve
- **customPlugin**: [boolean = false] : The boolean flag if this is a core or custom plugin.
- **newInstance**: [boolean = false] : The boolean flag if the factory should create a new instance of the named plugin.
- **module**: string : The name of the module where the plugin is located

The other method you can use is the `getMyPlugin` method for retrieving only **custom plugins**:

```
getMyPlugin(string plugin, string module="")
```

## Injecting Plugins

You can also inject plugins into other plugins, handlers, interceptors or model objects using our [WireBox](#) Injection DSL:

```
property name= "logger" inject= "coldbox:plugin:Logger" ;
property name= "zipUtil" inject= "coldbox:plugin:Zip" ;
```

## Code Samples

Below you can see some code snippets of useful plugin calls:

```
function preHandler(event, rc, rc, action){
    getPlugin( "Logger" ).info( "I am just inside the pre-handler method executing #arguments.action#" );
    getPlugin( "Timer" ).start( "Executing #arguments.action#" );
}

<--- Switch the i18n Locale --->
<cfset getPlugin( "i18n" ).setfwLocale( rc.locale ) >

<--- Java loader setup --->
<cfset getPlugin( "JavaLoader" ).setup( "includes/helloworld.jar" )>

<--- Set a message Box --->
<cfset getPlugin( "MessageBox" ).setMessage( "error", "Error retrieveing news feed" )>
<cfset getPlugin( "MessageBox" ).setMessage( "warning", "Warning message" )>
<cfset getPlugin( "MessageBox" ).setMessage( "info", "User created successfully" )>

<--- Get a Web Service Object declared in your config --->
<cfset wsObj = getPlugin( "Webservices" ).getWsobj( "newsfeed" )>

<--- Clear ClientStorage News --->
<cfset getPlugin( "ClientStorage" ).deleteVar( "newsfeed" )>

<--- Clean XSS attacks --->
<cfset rc.title = getPlugin( "AntiSamy" ).clean( rc.title ) >
```

The best place to learn about the functionalities of the core plugins is to visit the [Online API](#).

## Custom Plugins

So what Can I use custom plugins for:

- As helper classes, loaded on demand and used on demand
- UI elements for reusable tasks
- For reusable tasks
- Security

- AJAX enhancing features
- Query related
- ANT
- Anything you can imagine.

## Conventions Location

All custom plugins should be placed in the **plugins** directory of your application. This is the location where ColdBox will look for custom plugins.

```

ApplicationRoot
|--+ plugins (Custom Plugins Directory)
   |--+ DateUtil.cfc

// Reference
var util = getMyPlugin( "DateUtil" );

```

## External Location

You can also declare a **coldbox.PluginsExternalLocation** setting in your configuration file. This will be a dot notation path or instantiation path where more external custom plugins can be found (You can use coldfusion mappings). So this way you can bring in an entire library of plugins for your usage on-demand. You can share plugins with your entire server and even create packages for them. Once you declare the setting and you inject or call for a plugin, the ColdBox plugin service will search your local conventions first and then your external location for the plugin object.

**Note:** Please note that the custom convention takes precedence over the external location. So if two of the same plugins exists on both locations, the custom convention location will take precedence.

## Full Path Location

You can also call the **getMyPlugin()** with the full instantiation path to the plugin object as well:

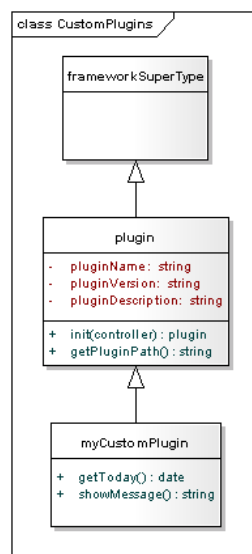
```
var util = getMyPlugin( "myapp.model.util.plugins.MyUtil" );
```

## Injecting Custom Plugins

You can also inject custom plugins using our [WireBox](#) Injection DSL into other plugins, interceptors, handlers and/or model objects:

```
property name= "logger" inject= "coldbox:myPlugin:DateUtil" ;
property name= "zipUtil" inject= "coldbox:myPlugin:package.Zip" ;
```

## Anatomy Of A Plugin



- Can be simple CFCs or inherit from our base **coldbox.system.Plugin**
  - If they inherit from our base, then they must have a specific *init()* signature
- You can call some setter methods to describe your plugin:
  - setPluginName()**
  - setPluginAuthor()**
  - setPluginAuthorURL()**
  - setPluginVersion()**
  - setPluginDescription()**

## Component Declaration

**With Inheritance:**

```

component extends= "coldbox.system.Plugin" {

    function init(){
        setPluginName( "My Awesome Plugin" );
        setPluginAuthor( "Luis Majano" );
        setPluginAuthorURL( "http://www.luismajano.com" );
        setPluginVersion( "1.0" );
        setPluginDescription( "A very cool plugin" );

        return this;
    }
}

```

**No Inheritance:**

```

component{

    function init(){
        // ANYTHING YOU LIKE HERE
        return this;
    }

}

```

When using the no-inheritance approach, calling the setter methods for the plugin properties is absolutely optional but recommended.

**Persistence**

Persistence for Plugins is done via our dependency injection framework [WireBox](#), so any persistence annotation or configuration via the configuration *binder* will apply to the plugin. However, by default, all plugins are **not cached**. Here are some of those annotations:

Attribute	Type	Description
<b>singleton</b>	none	Marks an object as a singleton object, which lives forever in the application
<b>scope</b>	Valid <a href="#">WireBox</a> Scope	Places the object in any valid <a href="#">WireBox</a> Scope
<b>cache</b>	boolean	A true or false will let the framework know whether to cache this object or not in the <b>default</b> cache region
<b>cacheBox</b>	string	Caches the object in the region you put here e.g <i>cachebox="cluster"</i>
<b>cachetimeout</b>	numeric	The timeout of the object in minutes. This is an optional attribute and if it is not used, the framework defaults to the default object timeout in the cache settings. You can place a 0 in order to tell the framework to cache the handler for the entire application timeout controlled by coldfusion.
<b>cacheLastAccessTimeout</b>	numeric	The last access timeout the object will have. This means that if the object has not been used for this amount of minutes, the cache will purge it.

```

// Singleton plugin
component singleton{}

// Cached Plugin
component cache= "true" {}

// Transient Plugin
component{}
component cache= "false" {}

// Cached with timeouts
component cache= "true" cachetimeout= "5" {}

// Cached with timeouts in a named cache
component cachebox= "cluster" cachetimeout= "10" {}

// Cached in request scope
component scope= "request" {}

```

So what are you waiting for? Let's get cooking with custom plugins!

**Extending/Overriding Core Plugins**

You can very easily override the functionality of core plugins without touching them and you can also add more plugins to the core by using our new **ColdBox Extensions** configuraiton setting. You can create a setting called **coldbox.ColdBoxExtensionsLocation** that points to a location in your server where the ColdBox extensions will be stored. This

setting is a dot notation path or ColdFusion mapping path.

```
coldbox = {  
  coldboxExtensionsLocation = "shared.extensions"  
};
```

The folder **extensions** must have a **plugins** folder in it. Any plugin you place here will take preference than the core plugins. So if you wanted to override our **JSON** plugin for example, you can drop one here called **JSON**:

```
/shared  
  /extensions  
    /plugins  
      JSON.cfc
```