

[<< Back to Dashboard](#) | [<< Projects Viewer](#)

# The Groovy Loader Project

## Contents

## Introduction

This plugin is thanks to Barney Boisevert for his cfgroovy inspiration.

This plugin dynamically loads a coldbox application with the groovy runtime and other java libraries you so desire. You can configure a classpath within your application that will act as the root of what you want to execute or you can dynamically use the groovy scripting tags.

The groovy loader plugin also leverages the coldbox cache in order to store the java class files the groovyloader creates from script. Internally, the groovy class loader also caches the parsed classes so they are not re-parsed at runtime. The plugin includes two methods to interact with these caches:

- `clearClazzCache()` : Cleans the coldbox cache of loaded groovy scripts
- `clearClassLoaderCache()` : Cleans the actual classloader's cache, use sparingly

Why would I want to clear the class loader cache? You want to do this, whenever you make changes to the groovy files on disk. However, be warned that classloading is a very tempestuous beast and it can lead to memory leaks or JVM permGen errors as class definitions do not get garbage collected or disposed of. However, this side effect is only visualized on development.

## Download

You can download the code from its forgebox entry: <http://coldbox.org/index.cfm/forgebox/view/Groovy-Loader-Project>

## Release Notes

### VERSION 2.0

- Added ability to load more than 1 location for groovy scripts
- Added ability to load more than 1 location for java libraries alongside groovy language
- Added the GroovyStarter interceptor for easy loading of the runtime, fully configurable

### VERSION 1.0

- Initial groovy integration

## Installation

### Requirements:

- ColdFusion 8 and above
- Railo 3.0 and above

### Install:

Copy the plugin folder: GroovyLoader to your custom plugins or bring them in via the PluginsExternalLocation setting.

Then either configure the class path and plugin like in the sample application:

```
<cfset getMyPlugin("GroovyLoader.GroovyLoader").configureClassPath(getSetting("ApplicationPath") & "model/groovy")>
```

Or just use the GroovyStarter interceptor, way easy:

```
<!-- <Groovy Starter: Creates & configures the GroovyLoader -->
<Interceptor class="{AppMapping}.plugins.GroovyLoader.GroovyStarter"
  <!-- <Paths that hold groovy libs -->
  <Property name="groovyLibPaths"/>{AppMapping}model/groovy,/{AppMapping}model/anotherPath/</Property>
  <!-- <Paths that hold jar's for us to load automagically -->
  <Property name="javaLibPaths"/>{AppMapping}model/lib/</Property>
</Interceptor>
```

## Interceptor Properties

Property	Description
<code>groovyLibPaths</code>	A comma-delimited list of relative/absolute path locations for groovy scripts or classes
<code>javaLibPaths</code>	A comma-delimited list of relative/absolute path locations for the java loader to load java libraries alongside the core groovy language
<code>pluginClassPath</code>	The location of the groovy loader plugin. By Default: <i>GroovyLoader.GroovyLoader</i>

- [The Groovy Loader Project](#)
  - [Introduction](#)
  - [Download](#)
  - [Release Notes](#)
    - [VERSION 2.0](#)
    - [VERSION 1.0](#)
  - [Installation](#)
    - [Requirements:](#)
    - [Install:](#)
    - [Interceptor Properties](#)
  - [Class Path Usage](#)
  - [Loading Groovy Classes](#)
  - [Executing Groovy File Scripts](#)
  - [Executing Groovy Source](#)
  - [Conclusion](#)

## Class Path Usage

If you will be using a groovy folder for entities or for script executions, then you will have to configure the plugin's `groovyLibPaths` to point to the locations where your groovy scripts/classes are located. The Groovy Loader will then let you create scripts and groovy classes at runtime by just using their instantiation name from the root directory downwards. The groovy loader plugin will search in all configured class paths for the scripts/classes, once it locates it it will create it. This way, you can have more than 1 location for the scripts.

Ex:

```
/model
 /groovy (is your in your groovyLibPaths)
   - Hello.groovy
 /util
   - DateCompare.groovy
```

To create these classes you would use their dot notation:

```
create("Hello")
create("util.DateCompare")
```

## Loading Groovy Classes

You can load any class via the `create(clazz)` method in the plugin. You can use dot notation to refer to packages, starting from the root of your class path as you defined in the section above.

```
rc.oHello = getMyPlugin(GroovyLoader.GroovyLoader).create("Hello");
```

## Executing Groovy File Scripts

You can also execute groovy as scripts by using the `runScript()` method and passing the name of the script using the dot-notation path as described above.

```
getMyPlugin("GroovyLoader.GroovyLoader").runScript("This is my script here...");
```

However, you can pass an optional argument called: **varCollection**. This is a structure that the groovy script will be binded with at runtime, so you can use it from within your script and basically create a data bridge between your app and groovy.

By default, the groovy script will be binded with the following variables/objects:

Groovy Variable Name	Description
<code>varCollection</code>	the <code>varCollection</code> argument.
<code>cf_pageContext</code>	the j2ee page context object
<code>cf_application</code>	the application scope
<code>cf_session</code>	if defined, the session scope
<code>cf_server</code>	the server scope
<code>coldbox_rc</code>	the coldbox request collection

Example:

```
var varCollection = {name:Luis Majano,age="31"};
getMyPlugin("GroovyLoader.GroovyLoader").runScript("Test",varCollection);
```

## Executing Groovy Source

You can execute dynamic groovy source by using two methods below:

1. executing the `runSource()` method
2. using the custom tag: `groovy:script`

The plugin will also bind the dynamically executed code with the same bindings as described above. The plugin will also create a unique hash according to your source in order to keep track of your compiled sources. The compiled sources will be stored in the coldbox cache for easy retrieval and usage. Once the script changes, a new entry is created in the cache and the groovy class loader will also cache the java representation. Again, be careful with permGen problems as object definitions keep increasing in development environments.

**Important Note:** If you get permgen errors, just restart the CF server and you are done.

To execute on the fly source, just call the `runSource()` method on the plugin. You can also pass in the optional `varCollection` argument.

```
//Declare the source in CF
var source = "varCollection.GroovyArray = [1,2,3,4]";
//Execute the source
getMyPlugin("GroovyLoader.GroovyLoader").runSource(source);
```

You can also use the tag `import` to execute runtime groovy. All you need to do is add the following `cfimport`

```
<--- Import GScript --->
<cfimport prefix="groovy" taglib="../plugins/GroovyLoader/tags"/>
```

Then you can script using the following tag `groovy:script`


```
<groovy:script>
<cfoutput>
def today = coldbox_rc[today]
def SubjectLine = [Hello,"my","name","is","Luis Majano.,"Expressed at",today]
//Place in ColdBox event context
coldbox_rc["SubjectLine"] = SubjectLine.join("_")
</cfoutput>
</groovy:script>
```

You can also add the `varCollection` optional attribute to the script tag

```
<groovy:script varCollection##MyStruct#>  
</groovy:script>
```

## Conclusion

You can now see the power of both Java and ColdFusion. ColdBox's GroovyLoader really makes it easy to integrate groovy in any ColdBox application. So now you can get Groovy with ColdBox!

 Categories:

- [level-advanced](#)