

Contents

- [New Features in Railo 3.2](#)
- [AJAX Functions](#)
- [AjaxOnLoad function](#)
- [CfAjaxImport tag](#)
- [CfAjaxProxy tag](#)
- [CfDiv tag](#)
- [CfMap / CfMapitem tag](#)
- [Bind Syntax](#)
- [Application.cfc Enhancements](#)
- [Application-wide Datasource](#)
- [Define DSN in Application.cfc](#)
- [onCFRequest\(\) method in Application.cfc](#)
- [Auto Imported CFC's](#)
- [Caching Features in Railo 3.2](#)
- [New Cache Functions](#)
- [Cache Extensions](#)
- [Named Caches](#)
- [Caches Per Object Type](#)
- [New Functions](#)
- [Directory Access Functions](#)
- [File Access Functions](#)
- [Other Functions](#)
- [ORM Related Functions](#)
- [Component Related Functions](#)
- [Misc Functions](#)
- [Array Related Functions](#)
- [List Related Functions](#)
- [Date Related Functions](#)

New Features in Railo 3.2

AJAX Functions

Thanks to the efforts of Andrea Campologni (URL), Railo 3.2 features a massive set of functions and tags that allow you to add AJAX functionality to your applications with ease.

AjaxOnLoad function

The AjaxOnLoad function allows you to programatically call a JavaScript function from Railo once the page has finished rendering.

If you would like to know more about this function checkout the [AjaxOnLoad documentation](#).

CfAjaxImport tag

The cfajaximport tag allows to import the javascript resources to be used into pages that use the Ajax Railo library.

```
tag="CFWINDOW"/>
```

More on cfajaximport [here](#).

CfAjaxProxy tag

Creates a JavaScript proxy for a ColdFusion component or create a proxy between a specific dom element and a cfc method a url or a javascript function.

```
cfc="ajaxproxy.cfc.test" jsclassname="proxyObj" onSuccess="successCallback" onError="errorCallback"/>
<script></span> type="text/javascript">
var myProxy = new proxyObj();
myProxy.getData();
/* call a amethod passing parameters */
var myProxy = new proxyObj();
myProxy.getData(200,'text');
/* You can also pass parameters like a js literal object*/
var myProxy = new proxyObj();
var args = {arg:100,arg2:200};
myProxy.getData(args);
</script>
```

More on cfajaxproxy [here](#).

CfDiv tag

Creates an HTML div tag or other HTML container tag and lets you use asynchronous form submission or a bind expression to dynamically control the tag contents.

```
bind="url:file.cfm?name={myForm:name}&age={myForm:age}"
onBindError="onError" bindonload="false" id="mydiv"/>
```

More on cfdiv [here](#).

CfMap / CfMapitem tag

Embed a geo map into the web page. Available support only for google maps.

```
name="myMap"
centeraddress="345 Park Avenue, san jose, CA 95110-2704, USA"
zoomlevel="10"/>
```

More on cfmap [here](#). More on cfmapitem [here](#).

Bind Syntax

Many ajax tags like `cfdiv` and `cfajaxproxy` make use of a particular binding syntax. Read more about [the supported bind syntax](#).

Application.cfc Enhancements

Controlling your Railo Application has been enhanced with features added to the lifecycle of the `Application.cfc` component.

Application-wide Datasource

In most applications, when you used a datasource, you had to repeatedly specify it in your `cfquery` tag, sometimes even adding it to the application scope as:

```
application.dsn = "myDSN">
...
name="getPosts" datasource="#application.dsn#">
Select ...
```

In Railo 3.2, you can now define this application wide in your `Application.cfc`:

```
component {
  this.name = "MyApplication";
  this.datasource = "mdblog";
}
```

And in any query, you will use the default datasource without having to define it:

```
name="getPosts">
SELECT * FROM mng_entry

query="getPosts">
...
```

Define DSN in Application.cfc

TODO

onCFCRequest() method in Application.cfc

In Railo 3.2 we have added the `onCFCRequest` method to the `Application.cfc`. If you implement this method, any remote call to a CFC will be intercepted by this method. You can then implement your own processing of the Component (such as logging before it is called).

```
component {
  this.name = "MyApplication";
  this.datasource = "mdblog";

  function onCFCRequest(String component,String method,Struct arguments) {
```

```
//Do something  
  
return true;  
}  
}
```

Auto Imported CFC's

Railo now includes a number of components that are auto imported into your components and scripts. Each script has the "org.railo.cfml" package which gives you access to the following components in your scripts:

- [Feed](#)
- [FTP](#)
- [HTTP](#)
- [Mail](#)
- [Query](#)

An example use would be to get a JSON string:

```
getInfo = new HTTP(url="http://localhost/sample.json");  
Results = getInfo.send();  
myStructure = DeserializeJSON(Results.getPrefix().filecontent);
```

Caching Features in Railo 3.2

In Railo 3.2 we have added a number of innovative caching features that allow you to build scalable applications. We have enhanced the cache* functions to allow you to name the cache you want to add information to. See the related documentation for:

- [cacheclear](#)
- [cachecount](#)
- [cachedelete](#)
- [cacheget](#)
- [cachegetall](#)
- [cachegetallids](#)
- [cachegetdefaultcachename](#)
- [cachegetmetadata](#)
- [cachegetproperties](#)
- [cachekeyexists](#)
- [cacheput](#)
- [cacheremove](#)
- [cachesetproperties](#)

New Cache Functions

- [cachegetdefaultcachename](#)

Buy default all these functions use the built-in RAM cache.

Cache Extensions

As part of Railo 3.2, we have added a number of server extensions that allow you to add different types of cache storage, rather than just relying on the built in RAM cache. This means that you can now use EHCatch (lite), CouchDB and MongoDB as cache stores.

To install these, simply head to your server administrator (e.g. <http://localhost/railo-context/admin/server.cfm>) click on the Extension/Applications section and you can install each of these caches.

Named Caches

With the addition of a number of cache extensions you can now use named caches to store your data. You can set a cache to be the default for Objects, Queries, Templates or Resources as well as storing data into a specific cache.

To store data to a specific named cache you can do:


```
cachePut("myItem", "test", createTimeSpan(0,0,10,0), createTimeSpan(0,0,10,0), "myCache")>
cacheItem = cacheget("myItem", false, "myCache")>
eval=cacheItem>
```

Which will put the value "test" into the cache called "myCache".

Caches Per Object Type

List of existing cache connection

List of all existing connection for this enviroment

<input type="checkbox"/>	Name	Type	Check
<input type="checkbox"/> 	myCache	RamCache	

Default cache connection

Define the default cache connection for tempates (cfcache) and object (cacheGet, cachePut ...), this connection is used when no cache name is explicit defined

Object	<input type="text" value="myCache"/> <small>This cache connection is used for all cache operations (cacheGet,cachePut ...)</small>
Template	<input type="text" value="-----"/> <small>This cache connection is used for the tag cfcache</small>
Query	<input type="text" value="-----"/> <small>This cache connection is used for the caching of the tag cfquery</small>
Resource	<input type="text" value="-----"/> <small>This cache connection is used for the Ram Resource (ram://...)</small>
<input type="button" value="update"/>	

In Railo 3.2, apart from being able to name each cache connection, you can assign each cache

connection to be the storage for different types of caches. For example you could set up a connection to CouchDB for your objects, a connection to ram for your templates and use a distributed EHCACHE connection to store your queries.

This allows for scaling and clustering of applications that can then share the objects from the cache repositories.

New Functions

In Railo 3.2 we have added a number of functions and CFScript language enhancements to help your productivity whilst coding in [CFSCRIPT](#) blocks.

Directory Access Functions

Railo 3.2 has added the following Directory based functions:

- [directorycreate](#)
- [directorydelete](#)
- [directorylist](#)
- [directoryrename](#)

File Access Functions

Railo 3.2 has added and enhanced the following File based functions:

- [fileseek](#)
- [fileskipbytes](#)

Other Functions

Railo 3.2 has added a number of other helpful functions

ORM Related Functions

- [entitydelete](#)
- [entityload](#)
- [entityloadbyexample](#)
- [entityloadbypk](#)
- [entitymerge](#)
- [entitynamearray](#)
- [entitynamelist](#)
- [entitynew](#)
- [entityreload](#)
- [entitysave](#)
- [entitytoquery](#)
- [ormclearsession](#)
- [ormclosesession](#)
- [ormejectcollection](#)
- [ormejectentity](#)
- [ormejectqueries](#)
- [ormexecutequery](#)

- [ormflush](#)
- [ormgetSession](#)
- [ormgetSessionfactory](#)
- [ormreload](#)
- [urlencode](#)

Component Related Functions

- [componentcacheclear](#)
- [componentcachelist](#)

Misc Functions

- [getvfsmetadata](#)
- [isipv6](#)

Array Related Functions

- [arraydelete](#)

List Related Functions

- [listcompact](#)
- [listitemtrim](#)

Date Related Functions

- [lsofweek](#)
- [lsweek](#)