

Error parsing plugin definition: An error occurred while Parsing an XML document. Element type "Redirect" must be followed by either attribute specifications, ">" or "/>".

[<< Back to Dashboard](#)

# ColdBox Cache Guide

## Contents

## Introduction

The ColdBox Cache is an in-memory cache that lives in **Application Scope** designed for handlers, plugins, events, views and any other objects or data you so desire. It has various tuning parameters such as default object timeout, default last access timeout, maximum objects to have in cache, a JVM memory threshold, a reaping frequency, eviction policies, an event model and so much more. The cache is based on java soft references (explained further below), several concurrency classes (if using JDK 5 and above) and is multi-threaded, which makes it an incredibly stable and fast caching engine. The cache is also tied to an event broadcast model, implemented via ColdBox interceptors, that can announce when objects are cached, removed or expired. In this guide we will explore the concepts, configurations and how to use the ColdBox Cache.

### Useful Resources

- <http://ecet.ees.ru.acad.bg/cst05/Docs/cp/SII/II.20.pdf>
- 

- <http://www.ibm.com/developerworks/java/library/j-jtp01246.html>
- <http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ref/SoftReference.html>
- [http://jdj.sys-con.com/read/36434\\_p.htm](http://jdj.sys-con.com/read/36434_p.htm)

## ColdBox Cache Features

- Fast and Simple
  - Fast ColdFusion and Java hybrid cache
  - Simple API and basic configuration parameters
  - Small Footprint
  - No need to create it, configure it or manage it if used within a ColdBox Application
- Solid Core
  - Multi-Threaded
  - Based on Java Concurrency Classes
  - Multiple Eviction Policies: LRU, LFU and FIFO
  - Memory Management & Memory Sensitive caching based on Java Soft References
  - Production Tested
  - Fully Documented
- Extensible & Flexible
  - Cache Listeners for broadcasting when objects are:
    - inserted
    - removed
    - expired
  - Create your own custom eviction policies
  - Extensive and granular purging mechanisms (regular expressions and key snippets)
  - Cache Statistics API for creating custom cache reports
- Highly Configurable
  - JVM Threshold Checks
  - Object Limits
  - Ability to time expire objects
  - Eternal (singletons) and time-lived objects
  - Object purging based on object usage (Access Timeouts)
  - Fully configurable at runtime via dynamic configurations and hot-updates
- Visually Appealing and Useful
  - Fully featured caching monitor and commands panel
  - Complete cache performance reports

- [ColdBox Cache Guide](#)
  - [Introduction](#)
  - [ColdBox Cache Features](#)
  - [Inner Workings](#)
  - [Java Soft References](#)
  - [Configuring the cache](#)
    - [Object Default Timeout](#)
    - [Last Access Timeout](#)
    - [Use Last Access Timeouts](#)
    - [Reap Frequency](#)
    - [Max Objects](#)
    - [Free Memory Percentage Threshold](#)
    - [Eviction Policies](#)
      - [LRU](#)
      - [LFU](#)
      - [FIFO](#)
    - [Dynamically Changing Policies](#)
  - [How do I use It?](#)
  - [Cache Monitor Screenshot](#)
  - [Cache Panel Commands](#)
    - [How to open the caching panel?](#)
  - [Techniques](#)
  - [Plugin/Handler Caching](#)
    - [Enabling Handler Caching](#)
    - [Caching Parameters](#)
  - [Interceptor Caching](#)
  - [View Caching](#)
  - [Event Caching](#)
    - [Enabling Event Caching](#)
    - [Setting up an event for caching](#)
    - [Event Cache Suffix](#)
    - [Purging Events](#)
  - [Cache Monitor Alongside Application Screenshot](#)

**Note:** ColdBox Cache works on CFMX 7 and above. Multi-threaded and Concurrency classes on CFMX 8 and above, Railo 3 and

above, and BD 7 and above only.

## Inner Workings

The ColdBox Cache in versions 2.5.0 and later, have cf8/BlueDragon7/Railo3 support, in which it takes advantage of using the **cfthread** tag to create new threads for reaping and maintenance. Not only does it take advantage of coldfusion improvements but also improvements of the new java JDKs. It uses several new data structures introduced in JDK 5 that are used for concurrency and caching, that gives the cache more stability under high traffic and faster performance.

The cache is a dual-performing cache in that it will store eternal (singleton) and non-eternal (life span) objects side by side. The non-eternal objects are stored as java soft references that provide you with a memory aware caching engine. This relieves the cache from purging non-used objects or purging when the JVM needs to recover RAM. All this is provided by the java soft reference implementations, which make this cache adapt as memory is needed or reaped by the JVM. However, it is also a fixed size & dynamic configuration cache engine in which it can be size limited and also limited by several configuration options. There have been several studies in the computer science world delineating why having a mixed cache will not only improve performance as load increases, but will also increase stability. The combination of having java soft references, fixed limits, and eviction policies is what makes this caching engine unique and powerful.

## Java Soft References

"In the past, developers didn't have much control over garbage collection or memory management. Java2 has changed that by introducing the java.lang.ref package. The abstract base class Reference is inherited by classes SoftReference, WeakReference and PhantomReference. SoftReferences are well suited for use in applications needing to perform caching."

Why is this? Well, a soft reference is nothing but a wrapper class that wraps an object in memory. Whenever the JVM requires memory and runs its reference rules, it can detect these soft references and decide to purge them if it meets the garbage collector rules. If it purges them, the wrapped object inside of the soft reference is marked for collection, but the java soft reference itself still exists. Therefore, the programmer can determine if the object inside the reference exists or not. If it does not exist, it means the object was garbage collected and I have no more object. I won't go into the implementation semantics of the cache, just theory. Please visit the references to understand more of how the ColdBox cache was built. In summary, soft references are what determine if an object is still available or not. The ColdBox cache can then run maintenance on itself and clean out all of its references for you.

## Configuring the cache

The ColdBox cache can be configured framework wide by tweaking the framework's settings xml file or via the ColdBox Dashboard. However, you can override all the values in your application via its configuration file (coldbox.xml.cfm). I will list below a table of the main settings and the framework defaults.

Setting	Type	Default Value	Description
<b>ObjectDefaultTimeout</b>	numeric	60	the default lifespan of an object in minutes
<b>ObjectDefaultLastAccessTimeout</b>	numeric	30	the default last access timeout in minutes
<b>UseLastAccessTimeouts</b>	Boolean	true	whether to use last access expirations
<b>ReapFrequency</b>	numeric	1	The delay in minutes to produce a cache reap (Not guaranteed)
<b>MaxObject</b>	numeric	100	The maximum number of objects in the cache
<b>FreeMemoryPercentageThreshold</b>	numeric	0	The numerical percentage threshold of free JVM memory to have. If the jvm free memory falls below this setting, the cache will no longer cache objects but evict them. (0=Unlimited)
<b>EvictionPolicy</b>	string	LRU	The eviction policy algorithm class to use. ColdBox ships with LFU, LRU and FIFO

Sample application configuration xml for your application's configuration file:

```
<Cache>
  <ObjectDefaultTimeout> 45</ObjectDefaultTimeout>
  <ObjectDefaultLastAccessTimeout> 15</ObjectDefaultLastAccessTimeout>
  <UseLastAccessTimeouts> true</UseLastAccessTimeouts>
  <ReapFrequency> 1</ReapFrequency>
  <MaxObjects> 50</MaxObjects>
  <FreeMemoryPercentageThreshold> 0</FreeMemoryPercentageThreshold>
  <EvictionPolicy> LRU</EvictionPolicy>
</Cache>
```

As you can see, the cache is extremely configurable. So let's analyze each setting one by one.

### Object Default Timeout

This numerical setting denotes the lifespan of objects in minutes. However, this is the minimum an object should live because there is no background process reaping the objects, the framework relies on incoming requests to cleanup the cache storage. Therefore, this timeout cannot be guaranteed, but it is a minimum.

## Last Access Timeout

This numerical setting denotes the amount of time in minutes that an object if not used, should live. The best way to describe this setting is to provide an example. Let's say object A has a timeout of 40 minutes and a last access timeout of 15 minutes. This means that once the object gets created, a timer starts. If that object is not used again in the following 15 minutes, then the cache sees that it has not been used and evicts it. This is a very useful setting as it can provide an even higher rotation for your objects and provide a more dynamic cache. However, tweaking is important and monitoring also. There has to be good enough separation or analysis on object life span's in order to apply this setting correctly.

## Use Last Access Timeouts

A Boolean variable that determines if you want to activate the last access timeout expiration evictions.

## Reap Frequency

A numerical setting that denotes the minimum frequency, in minutes, that the cache should reap itself. Reaping is always costly, so you will have to analyze your application in order to determine a correct frequency. Again, this setting is not guaranteed as it relies on incoming requests. So be sure to monitor your application and determine a realistic reaping frequency. Internally, the JVM will garbage collect all the soft references it needs as it requires RAM or not. All of this is provided by the JVM at its discretion (better for us). So our cache reaping basically cleans up what the JVM already cleaned up. All the hard work is done by the JVM, in order to maintain a safe and stable system.

## Max Objects

This determines the maximum number of objects the cache can hold. If the cache has reached its limit, the new objects will not be cached until space is available. Therefore, it will evict objects from the cache using the selected eviction policy.

## Free Memory Percentage Threshold

This setting is a numerical setting denoting the free memory percentage in the JVM. This is a fail-safe setting that basically tells the cache NOT to cache anything more if the JVM is running out of memory, plus it will evict objects using the eviction policy, until the JVM threshold is clear. So it can potentially recover some needed RAM if needed. However, since the cache is based on java soft references, the JVM gives an additional boost by garbage collecting those objects too.

## Eviction Policies

ColdBox ships with the following eviction policies:

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

However, the framework provides you with an interface that you can use in order to build your very own eviction algorithm classes and use them. Please look at the following interface: `coldbox.system.cache.policies.AbstractEvictionPolicy`.

### LRU

With this eviction policy, the cache discards the least recently used items first.

### LFU

This policy counts how often an item has been accessed and it will discard the items that have been used the least.

### FIFO

This policy works as a queue, first object in will be the first object out of the cache. So older staler objects are purged.

## Dynamically Changing Policies

The ColdBox cache let's you dynamically change the eviction policy of the cache by using the method `setevictionPolicy(policy:coldbox.system.cache.policies.AbstractEvictionPolicy)`. With this functionality you can create sophisticated eviction mechanisms that can be changed according to your application's criteria, maybe traffic, memory consumption, you name it. With the ability to dynamically influence the eviction policy, you have control on how aggressive or not your policy can be.

```
<cfset getColdboxOCM().setEvictionPolicy(MyPolicyObject) >
```

## How do I use It?

Well, the main methods for using the cache can be found in the [ColdBox API](#). However, the main method that you need to memorize to use get cache manager within your application is `getColdBoxOCM()`. This method is used to retrieve the ColdBox cache manager so you can interact with the cache from within your handlers, plugins, interceptors, proxy, etc.

Below are some of the most used caching methods:

Method	Description
--------	-------------

<b>clear</b>	<b>DO NOT CALL FROM A CACHED OBJECT SUCH AS A HANDLER,PLUGIN OR INTERCEPTOR.</b> If you do, then you will receive a coldfusion 500 null error. VERY VERY bad. If you want to clear all the objects in the cache, you can either, reinitialize the application or use the expireAll() method.
<b>clearKey</b>	Clears a key from the cache.
<b>clearKeyMulti</b>	Clears multiple keys from the cache
<b>clearByKeySnippet</b>	Ability to clear multiple entries by using a key snippet.
<b>clearEvent</b>	Ability to clear event caching via snippets and more
<b>clearAllEvents</b>	Clears all the event caching
<b>clearView</b>	Clears views by key snippets
<b>clearAllViews</b>	Clears all the cached views
<b>expireAll</b>	Expires all the objects in the cache. This is safe to call from cached objects.
<b>expireKey</b>	Expires a key
<b>expireByKeySnippet</b>	Expires keys by their key snippets
<b>get</b>	Get an object from cache. If it doesn't exist it returns a blank structure.
<b>getMulti</b>	Get's multiple objects from the cache.
<b>getSize</b>	Get the cache's size in items
<b>getCachedObjectMetadata</b>	Get an object's cache stats
<b>getCachedObjectMetadataMulti</b>	Get multiple object's cache stats
<b>lookup</b>	Check if an object is in cache.
<b>set</b>	sets an object in cache.
<b>setMulti</b>	Set multiple entries into the cache.

## Cache Monitor Screenshot

What good is a cache if I can't see what is inside or monitor it. Well, that is why we built the cache monitor that you can see below. The monitor will let you know all the necessary statistics and metrics about your cache. You can even dump the contents of individual items if needed.

CacheBox

Open Cache Monitor
CacheBox ExpireAll()
CacheBox ReapAll()

**CacheBox ID** 942813171  
**Configured Caches** TEMPLATE,default  
**Scope Registration** {ENABLED={false},SCOPE={server},KEY={cachebox}}

---

**Performance Report For** default Cache Regenerate Report

**Cache Name** default [class=coldbox.system.cache.providers.CacheBoxColdBoxProvider]  
**Performance** Hit Ratio: 83.62% ==> Hits: 97 | Misses: 19 | Evictions: 0 | Garbage Collections: 0 | Object Count: 21  
**JVM Memory Stats** 34.47 % Free | Total Assigned: 236,096 KB | Max: 236,096 KB  
**Last Reap** Aug-07-2010 07:56:17 PM

**Free Memory (KB)**  
**Used Memory (KB)**

81,391.812  
154,704.188

Run Garbage Collection

**Hits**  
**Misses**  
**Evictions**

Hits: 97  
Misses: 19  
Evictions: 0

**Plugins**  
**Events**  
**Interceptors**  
**Handlers**  
**Views**  
**Other Objects**

0  
7  
7  
6  
7  
7

---

**Cache Configuration** Show/Hide

**Cache Content Report**

Reload Contents
Expire All Keys
Clear All Events
Clear All Views

Object	Hits	Timeout	Idle Timeout	Created	Last Accessed	Status	CMDS
cbox_customplugin-jsmin	2	0	30	Aug-07 07:56:18 PM	Aug-07 07:56:18 PM	Alive	DEL
cbox_interceptor-autowire	1	0	30	Aug-07 07:56:17 PM	Aug-07 07:56:17 PM	Alive	DEL
cbox_interceptor-deploy	1	0	30	Aug-07 07:56:17 PM	Aug-07 07:56:17 PM	Alive	DEL

## Cache Panel Commands

The cache content panel has some nice commands that you can use on-demand! This will help you issue commands to the cache when you are in debug mode ONLY:

- Clear cache entries
- View cache entries
- Clear all cached events
- Clear all cached views
- Expire all objects

Cache Content Report (Time: 08:41:30 AM)						
<a href="#">Expire All Keys</a>   <a href="#">Clear All Events</a>   <a href="#">Clear All Views</a>						
Object	Hits	Timeout (Min)	Created	Last Accessed	Expires On	CMDS
event-ehSamples.dspHome-6E577C46361430EBD95D509A47B992BE	4	1	Dec-17 08:41:20 AM	Dec-17 08:41:30 AM	Dec-17 08:42:20 AM	<input type="button" value="DEL"/>
handler-coldbox.samples.handlers.ehSamples	1	60	Dec-17 08:41:19 AM	Dec-17 08:41:19 AM	Dec-17 09:41:19 AM	<input type="button" value="DEL"/>
plugin-i18n	67	0	Dec-16 01:51:25 PM	Dec-17 08:41:20 AM	---	<input type="button" value="DEL"/>
plugin-resourceBundle	391	0	Dec-16 01:51:25 PM	Dec-17 08:41:20 AM	---	<input type="button" value="DEL"/>
plugin-sessionstorage	3	60	Dec-17 08:41:19 AM	Dec-17 08:41:30 AM	Dec-17 09:41:19 AM	<input type="button" value="DEL"/>
plugin-Utilities	1	5	Dec-17 08:41:30 AM	Dec-17 08:41:30 AM	Dec-17 08:46:30 AM	<input type="button" value="DEL"/>

## How to open the caching panel?

In order to open the cache debug panel you must first be in debug mode. Please look at the [wiki:cbURLActions URL Actions Guide]. Once the panel is open, you can see a cache overview report and a cache content report. You can now even click on the name of the cached element in order to see its content.

The `debugpanel` url action can ONLY BE used when in debug mode. If you are not in debug mode, then the panel will not be rendered. You can open the monitor by clicking on the **Open Monitor** button inside of the cache panel of the debugger or you can open it by going to the following URL:

```
index.cfm?debugpanel=cache
```

## Techniques

There are various ways that you can use the ColdBox cache and it all depends on the data you want to store and how to retrieve it. One method that I have used for storing queries in the cache is to use the on application start method to place the queries on the cache indefinitely.

```
//On the application start
<cfscript>
var qStates = getPlugin( "ioc" ).getBean( "lookupsService" ).getStates();
var qCountries = getPlugin( "ioc" ).getBean( "lookupsService" ).getCountries();

//Place in cache indefinitely until application reinitializes.
getColdBoxOCM().set( 'qStates', qStates, 0 );
getColdBoxOCM().set( 'qCountries', qCountries, 0 );
</cfscript>
```

```
//Then on my handlers or views/layouts I can do the following
<cfset qStates = getColdboxOCM().get( 'qStates' )>
```

This is a simple example of placing data on the cache that will persist throughout the entire application. Another example is you can place a query to persist for 10 minutes:

```
<cfset getColdboxOCM().set( 'myQuery', myQuery, 10 ) >
```

However, when you do a `get` the method can fail because the query might have expired. That is why you have the `lookup()` method which tells you if the key you sent in exists in the cache. This is useful to create facade methods where you can do the following:

```
<cffunction name="getMetadata" access="private" returntype="any" output="false">
  <cfargument name="lookup" type="string">
  <cfscript>
    var oMD = structnew();
    if ( getColdboxOCM().lookup(arguments.lookup) ){
      oMD = getColdboxOCM().get(arguments.lookup);
    }
    else {
      //Create md from Transfer
      oMD = getTransfer().getTransferMetaData(arguments.lookup);
      //set back in cache
      getColdboxOCM().set(arguments.lookup, oMD, 10);
    }
    return oMD;
  </cfscript>
</cffunction>
```

This is a great example of using the ColdBox cache for checking and verifying data. As you can see, there are several ways to tap into the power of the ColdBox cache. The following example is to create your own url action to expire objects that is part of a request start handler.

```
<cffunction name="onRequestStart" access="private" returntype="void" output="false">
  <cfargument name="event" type="any" required="true">
```

```

<cfscript>
//Request start handler.

//check if you pass in a url var called: clear cache
if ( event.valueExists( "clearcache" ) ){
    getColdBoxOCM().expireAll();
    setNextEvent();
}
</cfscript>
</cffunction>

```

## Plugin/Handler Caching

The framework can keep persistence for you for handlers and plugins. Plugin persistence is done automatically, all you need to do is set several caching parameters in your plugin's cfcomponent declaration. Handlers act the same way, in which you need to add the caching parameters to the cfcomponent tag, but also require a setting called: **HandlerCaching**. You need to activate this setting in order for handlers to be cached by the framework.

### Enabling Handler Caching

To enable handler caching, you will need to set a setting in your configuration file called **HandlerCaching** which is a Boolean variable.

```
<Setting name="HandlerCaching" value="true" />
```

If you do not enable this setting, the framework will not cache your handlers. It would be a good idea to have this set to false in development, so your changes can be reflected.

### Caching Parameters

Also, the framework detects if these parameters exist in the cfcomponent declaration. If they do not exist, it has some internal rules that will determine if the object is cached or not. Below is a table determining caching defaults:

Object	Cached By Default
<b>handlers</b>	TRUE if handler caching is enabled
<b>plugins</b>	FALSE

As you can see from above, by default handlers **WILL BE** cached, unless you specifically use the cache meta data attributes to tell the framework NOT to cache it. Caching of handlers simulates persistence, so remember this if you are planning handlers that can maintain their own persistence. This is a true flexible and awesome feature. Persistence controlled by the framework for you. Same applies to plugins, but plugins are NOT cached by default. You need to specifically use the cache metadata attributes to tell the framework to cache them. Below are the cache metadata parameters you will add to the cfcomponent tag.

Attribute	Type	Description
cache	Boolean	A true or false will let the framework know whether to cache this handler object or not.
cachetimeout	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 (Zero) in order to tell the framework to cache the handler for the entire application timeout controlled by coldfusion.
cacheLastAccessTimeout	numeric	The last access timeout of the object in minutes. This is an optional attribute and if it is not used, the framework defaults to the default last access object timeout in the cache settings. This tells the framework that if the object has not been accessed in X amount of minutes, then purge it.

```
<cfcomponent name="mainHandler" output="false" cache="true" cacheTimeout="0">
</cfcomponent>
```

```
<cfcomponent name="myPlugin" output="false" cache="false">
</cfcomponent>
```

```
<cfcomponent name="captchaPlugin" output="false" cache="true" cacheTimeout="30" cacheLastAccessTimeout="10":
</cfcomponent>
<cfcomponent name="captchaPlugin" output="false" cache="true" cacheTimeout="5">
</cfcomponent>
```

```
<cfcomponent name="userHandler" output="false" cache="true" cacheTimeout="30" cacheLastAccessTimeout="10">
</cfcomponent>
```

## Interceptor Caching

All interceptors declared in the configuration file will be cached automatically by the framework as singletons. In order, for changes to take effect in interceptors, you must re-initialize the application by using the **fwreinit** action url.

## View Caching

In order for you to cache a view, you will have to use the `event.setView()` or the actual `renderView()` methods. You can read an in-depth guide about view caching in the [Layouts & Views Guide](#). This guide will also show you how to purge views from the cache.

So in order to tell the framework to cache the view called `vwHome` for 10 minutes, you would do the following:

```
event.setView(view= "vwHome" ,cache= "true" ,cacheTimeout= "10" );
```

That easy! The framework reads the caching metadata, constructs a cache entry and then saves it for pickup by the renderer plugin. How can you tell if the view was rendered from cache or not? Well, if you are in debug mode, you can check out the debugging panel and you will see that if you are normally rendering a view it says: *Rendering View [vwHome.cfm]*. However, if rendering from cache, it will say: *Rendering Cached View [vwHome.cfm]*

Here is an example of rendering and caching at the same time:

```
<body>
  <---< Render and cache the header --->
  #renderView(name='tags /header' ,cache=true,cacheTimeout='10')#

  <div>
  <---< Main View --->
  #renderView()#
  </div>
</body>
```

## Event Caching

Event caching is extremely useful and easy to use. All you need to do is add several metadata arguments to the `cffunction` tag of your event and the framework will cache the output of the event. In other words, the event executes and produces output that the framework then caches. So the subsequent calls do not do any processing, but just output the content. Almost all of the entire life cycle is skipped, the content is just delivered. Below you can see the life cycle of a cached event:

### Cached Event Life cycle

1. Event is trapped
2. URL action `fwCache` is verified, if it exists, incoming event is purged from cache.
3. `preProcess` Interception is executed
4. `RequestStartHandler` is executed
5. Cached content is retrieved and outputted.

**Important:** Please be aware that an event can have several unique permutations depending on the incoming url/form parameters. Also, caching is application wide, not user wide. So be careful of not caching secure events or session based events.

As from the warning above, when event caching is enabled for an event, the framework will hash the incoming request collection plus the incoming event to determine a cacheable key. So an event can have thousands of different combinations that will create multiple cache entries. So make sure that you have the `Cache Max Objects` setting turned on, so your cache can be more dynamic.

### Enabling Event Caching

To enable event caching, you will need to set a new setting in your configuration file called `EventCaching` which is a Boolean variable.

```
<Setting name= "EventCaching" value= "true" />
```

If you do not enable this setting, the framework will not cache your events. It would be a good idea to have this set to false in development, so your changes can be reflected.

### Setting up an event for caching

The way to set up an event for caching is on its `cffunction` declaration with the following extra arguments:

Attribute	Type	Description
<code>cache</code>	Boolean	A true or false will let the framework know whether to cache this event or not. The default is FALSE.
<code>cachetimeout</code>	numeric	The timeout of the event's output 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 event's output for the entire application timeout controlled by coldfusion, NOT GOOD. Always set a decent timeout for content.
<code>cacheLastAccessTimeout</code>	numeric	The last access timeout of the event's output in minutes. This is an optional attribute and if it is not used, the framework defaults to the default last access object timeout in the cache settings. This tells the framework that if the object has not been accessed in X amount of minutes, then purge it.

**Important:** Please be aware that you should not cache output with 0 timeouts. Always use a timeout.

Here is an example of setting up an event for caching:

```
<cffunction name="dspBlog" access="public" returntype="void" output="false" cache="true" cacheTimeout="30">
```

That's it, whenever the framework detects the dspBlog event, it will cache it.

## Event Cache Suffix

This property is great for adding your own dynamic suffixes when using event caching. All you need to do is create a public property called **EVENT\_CACHE\_SUFFIX** in your event handler and populate it with something you want; static or dynamic. A great use case for this, is multi-lingual applications. You can set the suffix to the user's locale. Then the event caching mechanisms will automatically append the suffix and thus create event caching according to locales now. How cool is that, a simple key and you have multi-lingual caching at your fingertips.

```
<--- Set it to the user's current locale --->
<cfset this.EVENT_CACHE_SUFFIX = getFWLocale() >
```

## Purging Events

If you need to override the event so you can see the real deal, you can use an URL action called: **fwCache**. This URL action tells the framework to not cache the incoming request. If the event is cached, it will first purge it and re-execute and re-cache it. If you want to purge all events, well, just reinit the application using the **fwReinit** URL action or call the **clearAllEvents()** method on the ColdBox cache; this will clear all the events in the cache. However, there are other methods that can clear a family of cached events, or family of handler events and more.

Ways to purge events:

- URL Action **fwCache**
- **getColdboxOCM().clearAllEvents([async=true])**
- **getColdboxOCM().clearEvent(eventSnippet,[queryString=""],[async=true])**

Examples:

```
//Trigger to purge all Events
getColdBoxOCM().clearAllEvents();

//Trigger to purge all events synchronously
getColdboxOCM().clearAllEvents(async= false);

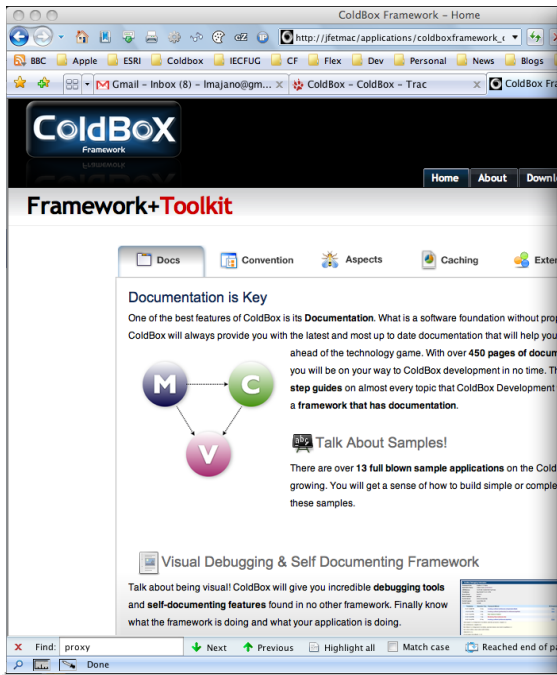
//Purge all events from the blog handler
getColdboxOCM().clearEvent( 'blog' );

//Purge all permutations of the blog.dspBlog event
getColdboxOCM().clearEvent( 'blog.dspBlog' );

//Purge the blog.dspBlog event with entry of 12345
getColdboxOCM().clearEvent( 'blog.dspBlog' , 'id=12345' );
```

As you can see from the **clearEvent()** samples, you can send in snippets of event names and the cache will try to find keys that match and reap them. You can also do this asynchronously if **cfthread** is available in your engine.

## Cache Monitor Alongside Application Screenshot



**ColdBox Framework - Home**

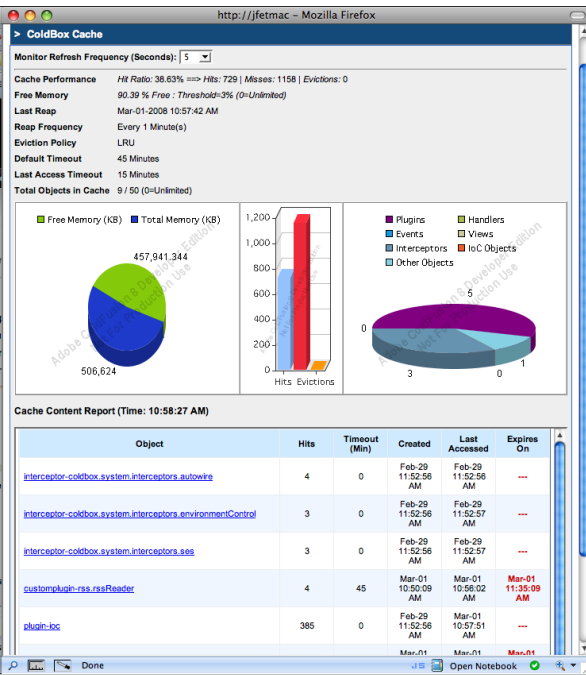
Framework+Toolkit

Documentation is Key

One of the best features of ColdBox is its **Documentation**. What is a software foundation without proper documentation? ColdBox will always provide you with the latest and most up to date documentation that will help you stay ahead of the technology game. With over **450 pages of documentation** you will be on your way to ColdBox development in no time. The **step guides** on almost every topic that ColdBox Development offers are a **framework that has documentation**.

**Visual Debugging & Self Documenting Framework**

Talk about being visual! ColdBox will give you incredible **debugging tools** and **self-documenting features** found in no other framework. Finally know what the framework is doing and what your application is doing.



**ColdBox Cache**

Monitor Refresh Frequency (Seconds): 5

Cache Performance: Hit Ratio: 38.63% ==> Hits: 729 | Misses: 1158 | Evictions: 0

Free Memory: 90.39 % Free : Threshold=3% (0-Unlimited)

Last Reap: Mar-01-2008 10:57:42 AM

Reap Frequency: Every 1 Minute(s)

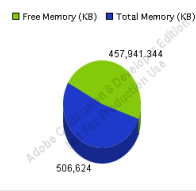
Eviction Policy: LRU

Default Timeout: 45 Minutes

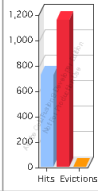
Last Access Timeout: 15 Minutes

Total Objects in Cache: 9 / 50 (0-Unlimited)

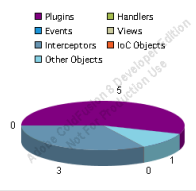
Free Memory (KB) | Total Memory (KB)



457,941,344  
506,624



Hits Evictions



5  
3  
0  
0  
1  
0  
0

**Cache Content Report (Time: 10:58:27 AM)**

Object	Hits	Timeout (Min)	Created	Last Accessed	Expires On
<a href="#">interceptor-coldbox.system.interceptors.autowire</a>	4	0	Feb-29 11:52:56 AM	Feb-29 11:52:56 AM	---
<a href="#">interceptor-coldbox.system.interceptors.environmentControl</a>	3	0	Feb-29 11:52:56 AM	Feb-29 11:52:57 AM	---
<a href="#">interceptor-coldbox.system.interceptors.asp</a>	3	0	Feb-29 11:52:56 AM	Feb-29 11:52:57 AM	---
<a href="#">customplugin-rss.rssReader</a>	4	45	Mar-01 10:50:09 AM	Mar-01 10:56:02 AM	Mar-01 11:35:09 AM
<a href="#">plugin-loc</a>	385	0	Feb-29 11:52:56 AM	Mar-01 10:57:51 AM	---