



# Cúram 8.2

## Cúram Cache Guide



## Note

---

Before using this information and the product it supports, read the information in [Notices on page 19](#)



# Edition

---

This edition applies to Cúram 8.2.

© Merative US L.P. 2012, 2025

Merative and the Merative Logo are trademarks of Merative US L.P. in the United States and other countries.



# Contents

---

**Note**.....

**Edition**.....

<b>1 Developing with Cúram Cache</b> .....	<b>9</b>
1.1 Overview.....	9
1.2 What is Cúram Cache.....	9
Configuration.....	9
Statistics.....	11
Shutting Down Cúram Cache.....	11
1.3 Global Caches.....	11
Global Cache Provider.....	12
Default Global Cache Group.....	12
Global Caches.....	12
Usage Recommendations.....	12
Configuration.....	13
Using Global Caches in a Transactional Context.....	14
1.4 Code Samples.....	14
Cache Loader.....	14
Cache Client.....	15
Cache Invalidation.....	15
1.5 Thread Local Caches.....	15
Configuration.....	16
Code Samples.....	16
1.6 Transaction Local Caches.....	17
Configuration.....	17
Code Samples.....	17

**Notices**.....

Privacy policy.....	20
Trademarks.....	20





# Chapter 1 Developing with Cúram Cache

---

Use this information to configure the Cúram cache. Cúram cache is a generic caching service for short and long lived caches. The service is available in both the client and server containers of an application server environment and in a standard stand-alone Java™ process.

## 1.1 Overview

---

The aim of this document is to introduce Cúram Cache, a generic caching service that is designed to satisfy the requirement for short and long lived caches in the application.

This guide is intended for architects and developers who are interested in using Cúram Cache to satisfy their caching requirements.

## 1.2 What is Cúram Cache

---

Cúram Cache is a generic caching service that is designed to satisfy the requirement for short and long lived caches in the application. The service is available in both the client and server containers of an application server environment (online application) and in a standard stand-alone Java™ process (batch programs).

Cúram Cache allows the creation of three types of caches:

- Global - these are global (at JVM level) caches.
- Thread local - these are caches that live as long as the thread that owns them.
- Transaction local - these are caches that exist during the current transaction.

The last two types of caches are collectively referred to in this document as multi-instance caches because at any moment there might be more than one instance of a cache with a given name (one for each active transaction or thread).

## Configuration

---

The configuration of all types of caches in Cúram Cache is entirely declarative and it is based on the configuration mechanism that is provided by the application. Cache configuration parameters must be added to the APP\_CACHE property section.

In the current implementation, global caches support both size and time-based eviction policies while the multi-instance caches have support only for time-based eviction policy.

The following cache configuration parameters can be adjusted:

- Size – the maximum number of elements in memory. The default value is 200. The type is INT32.

- Eviction policy – the policy that is used for evicting items from memory when the maximum number of elements in memory is reached. The default value is `LRU`. The type is `STRING`. Valid values are:
  - `LRU` – least recently used
  - `LFU` – least frequently used
  - `FIFO` – first in first out
- Time to live - is the maximum number of seconds an item can remain in the cache regardless of use. Because the default implementation of Curam Cache is backed by Ehcache, the item expires at this limit and will no longer be returned from the cache (but may not be evicted yet).
- Time to idle - is the maximum number of seconds an item in the cache is allowed to be unused. Because the default implementation of Curam Cache is backed by Ehcache, the item expires at this limit and will no longer be returned from the cache (but may not be evicted yet).

All cache configuration properties must conform to this notation:

```
curam.cache.<cache_group_name>.<cache_name>.<parameter>
```

where:

- `<cache_group_name>` - is the name of the cache group the cache belongs to.
- `<cache_name>` - is the name of the cache. This might also be "\*" and then, the configuration parameter is applied to all caches in this cache group.
- `<parameter>` - can be `size`, `evictionPolicy`, `timeToIdle` or `timeToLive`.

In the example below, the global cache `curam.myproject.mycache` in the default global cache group `curam-group` is configured with a size of 1000 items and an eviction policy of Least Recently Used.

```
curam.cache.curam-group.curam.myproject.mycache.size=1000
curam.cache.curam-group.curam.myproject.mycache.evictionPolicy=LRU
```

*Figure 1: Configuring a cache*

In this second example, the transaction local cache `curam.myproject.mycache` in the transaction local cache group `transaction-group` is configured with a time to idle of 10 seconds while all other transaction local caches are configured with a value of 5 seconds.

```
curam.cache.transaction-group.curam.myproject.mycache.timeToIdle=10
curam.cache.transaction-group.*.timeToIdle=5
```

*Figure 2: Configuring all caches in a group*

Cache configuration data that is stored in the application configuration repository can be overridden by passing the relevant values as JVM system properties. This might be of interest for batch processes where the application profile might be different than the online application.

The example here shows how to disable the global cache `curam.myproject.mycache` in the default global cache group for a batch process.

```
ant -f app_batchlauncher.xml
    -Dcuram.cache.curam-group.curam.myproject.mycache.size=0
```

```
-Dbatch.userna...
```

Figure 3: Disabling a cache for a batch process

## Statistics

All caches in Cúram Cache are instrumented for statistics and these are integrated with the Cúram JMX infrastructure.

The following minimum set of statistics are exposed by each type of cache through the `CuramCacheStats` MBean:

- Cache group - the name of the cache group
- Cache – the name of the cache
- Layer – the name of the cache layer (memory, disk,...)
- Size – the number of items in the cache
- Hits - the number of requests to the cache that returned an item that is already loaded in the cache
- Misses - the number of requests to the cache that returned an item, which had to be loaded in the cache
- Evictions - the number of times items that are evicted from the cache
- Average get time(ns) - the average elapsed time, in nanoseconds, that takes for an item to be read from the cache. Note that some cache providers might support only millisecond resolution.

Multi-instance caches offer snapshot and aggregated statistics. Snapshot statistics are for all instances alive at the moment of the query and aggregated statistics are calculated from all instances that are created.

## Shutting Down Cúram Cache

Cúram Cache requires orderly shutdown on JVM exit. Cúram Cache installs automatically a JVM shut-down hook to clear the cache as the last resort solution but it is recommended, where possible, the use of the explicit shutdown by starting `CacheManager.shutdown()` when the application is shut down.

## 1.3 Global Caches

Global caches are caches that exist in the scope of the JVM process or beyond. In the current version of Cúram Cache, global caches exist only in the scope of the JVM process. An entry that is stored in a global cache lives across transaction boundaries until it is removed explicitly, by the developer, or implicitly, as a result of the eviction policy associated with the cache.

It is important to note that because global caches are long lived, their data is prone to short periods of inconsistency when cached objects are updated. When an update is made in the application that affects a cached object, the associated cache entry is invalidated asynchronously. The caching infrastructure guarantees that the cache entry is, eventually, invalidated but it

cannot guarantee a certain maximum time frame. Understanding this behavior is important when deciding if certain application data can be cached in a global cache.

## ***Global Cache Provider***

---

Cúram Cache implements large parts of the global caching infrastructure that uses third party caching solutions, which are referred to in this document as caching providers. The default provider is Ehcache, an open source, high performance, distributed caching infrastructure.

## ***Default Global Cache Group***

---

Global caches are grouped together based on common configuration requirements such as replication and disk storage. All caches in the application are created in the default cache group. The name of the default cache group is `curam-group`.

In the current implementation, the default cache group is not self-replicating and does not support disk overflow and disk persistence. Because self-replication is disabled, the cache operations are only visible to the JVM where the global cache is located. However, to keep all caches in the default cache group consistent throughout the application server cluster, an explicit cache invalidation mechanism is provided. The cache invalidation can be triggered only from the server code but it invalidates caches in both the server and the client containers across all JVMs in the application server cluster.

## ***Global Caches***

---

Global caches are created with a call to the `get()` method of the cache group. If a cache does not exist already, a cache is created and configuration data, if this data exists, is applied to the cache before being returned. Global caches are usually populated by using cache loaders that are registered by cache clients. This approach isolates the cache client from the management of concurrent access to the cache while the cache is loading.

However, Cúram Cache does not enforce the use of serializable objects in its API, certain features offered by the caching infrastructure are only available if the key or the cached object are serializable. For this reason, it is recommended that, whenever possible, serializable keys and values are used in Cúram Cache.

**Usage of non-serializable keys** Cache entries that have non-serializable keys are only invalidated on the local JVM and not throughout the application server cluster.

## ***Usage Recommendations***

---

The following list is recommendations on how a global cache is used:

- Only cache immutable objects.

- Use serializable keys and values whenever possible. At the very least the keys should be serializable.
- Use a cache loader to populate the cache. This allows the cache to take advantage of the fine grain concurrency optimizations that are built into the cache provider and it does not require the user to be concerned with managing concurrent access to the cache.
- The loading of a cache without a loader (by using `get ( )` and `put ( )` calls) must be avoided for two main reasons:
  - Concurrency management - In this case the user is responsible for managing concurrent access to the cache while the cache is loading. The user has two choices:
    - Control concurrent access to the `get ( )` and `put ( )` block of code – this approach is not recommended in a performance sensitive part of the application but it offers the guarantee that an object is only loaded once.
    - Allow concurrent access to the `get ( )` and `put ( )` block of code – this approach supports higher concurrency but an object might be loaded more than once by different threads.
  - Efficient data management - without a cache loader, the cache must be pre-populated with all data. With a cache loader, only required data is pulled into the cache.
- Use cache names that are prefixed with a package name unique to your project. For instance `curam.cpm.myCache` would be a suitable name for a cache in the Curam Provider Management project.

## Configuration

---

All global caches inherit default values for the configuration parameters.

Global caches inherit the following default configuration parameter values:

- `size` - 200
- `evictionPolicy` - LRU
- `timeToLive` - 0 (not active)
- `timeToIdle` - 0 (not active)

The `timeToLive` parameter is the maximum number of seconds that an item can remain in the cache regardless of use. Because the default implementation of Curam Cache is backed by Ehcache, the item expires at this limit and will no longer be returned from the cache. However, it cannot be evicted yet.

The `timeToIdle` parameter is the maximum number of seconds that an item in the cache is allowed to be unused. Because the default implementation of Curam Cache is backed by Ehcache, the item expires at this limit and will no longer be returned from the cache. However, it cannot be evicted yet.

The default values can be overridden for any global cache. In the following examples, `curam-group` is the name of the default cache group and `curam.myproject.mycache` is the name of the cache.

```
curam.cache.curam-group.curam.myproject.mycache.size=1000
curam.cache.curam-group.curam.myproject.mycache.evictionPolicy=LRU
curam.cache.curam-group.curam.myproject.mycache.timeToLive=3600
curam.cache.curam-group.curam.myproject.mycache.timeToIdle=300
```

Figure 4: Configuring a global cache

## Using Global Caches in a Transactional Context

When a global cache is used in a transactional context, care must be taken to ensure that the cache maintains its consistency in case the current transaction is rolled back.

**Cache invalidation in a transactional context** When modifying data that affects the content of a cache do not remove or update the cached element directly; instead, invoke the `CacheManagerEjb.postInvalidationMessage()` method to post an invalidation message that triggers the cache invalidation.

## 1.4 Code Samples

The section contains code samples that show how to write a cache loader, how to use the cache and how to invalidate a cache entry.

### Cache Loader

The example shows how the `CacheLoaderAdapter` class is used to help in the implementation of `MyCacheLoader`.

```
...
public class MyCacheLoader extends
    CacheLoaderAdapter<Integer, ReadWorkQueueDetails> {
    /* (non-Javadoc)
     * @see curam.util.cache.CacheLoader#load(java.lang.Object)
     */
    public ReadWorkQueueDetails load(Integer workQueueID)
        throws AppException, InformationalException {
        WorkAllocation wa = (WorkAllocation)WorkAllocationFactory
            .newInstance();
        ReadWorkQueueKey key = new ReadWorkQueueKey();
        key.key = new ReadWorkQueueKey();
        key.key.key = new WorkQueueKey();
        key.key.key.workQueueID = workQueueID;
        ReadWorkQueueDetails item = wa.readWorkQueue(key);
        if(item != null) {
            return item.dtls;
        }
    }
}
```

```

        return null;
    }
}
...

```

Figure 5: Using `CacheLoaderAdapter` to implement a cache loader

## Cache Client

---

The example shows the usual way of registering a cache loader and using the cache.

```

...
public class MyCacheClient {
    // keep a static reference to mycache
    private static Cache<Integer,
        ReadWorkQueueDetails> myCache;

    static {
        // retrieve a reference to mycache and register
        // the cache loader
        myCache = CacheManager.getDefaultCacheGroup()
            .getCache("mycache");
        myCache.registerCacheLoader(new MyCacheLoader());
    }

    public WorkAllocation() {
        ...
    }
    ...
    // use the cache
    ReadWorkQueueDetails wq = myCache.get(1);
    ...
}

```

Figure 6: Registering a cache loader and using the cache

## Cache Invalidation

---

The example shows how to invalidate a cache entry in code running in a transactional context (server code).

As explained in [Default Global Cache Group on page 12](#), cache invalidation for global caches in the default cache group can be triggered only by server code.

```

...
CacheManagerEjb.postInvalidationMessage(
    new CacheInvalidationMessage<Key>("mycache", 1));
...

```

Figure 7: Invalidating a cache entry in server code

## 1.5 Thread Local Caches

---

These caches are closely tied to the thread used to create them. No other thread can access data in these caches and caches are only deleted when the thread that created them is stopped. Thread

local caches are specialized. They must be used only for small caches where the overhead of multi-threaded access control that exists for global cache cannot be tolerated.

## Configuration

---

Thread local caches support only a time-based eviction policy.

The only two configuration parameters that can be used are

- `timeToLive` - this is the maximum number of seconds an item can remain in the cache regardless of use
- `timeToIdle` - this is the maximum number of seconds an item in the cache is allowed to be unused.

The default properties for the parameters are:

- `timeToLive` - 0 (not active)
- `timeToIdle` - 0 (not active)

The name of the group for thread local caches is `thread-group`. This name must be used to configure thread local cache as shown in the example.

```
curam.cache.thread-group.curam.myproject.mycache.timeToLive=60
curam.cache.thread-group.curam.myproject.mycache.timeToIdle=10
```

*Figure 8: Configuring a thread local cache with name `curam.myproject.mycache`*

## Code Samples

---

Thread local caches are by accessed only where the correct context (thread) exists.

For instance, it is not recommended to set up a thread local cache in static block of code. Because that thread might not be the same as the thread using the cache later.

```
public void myMethod() {
    ...
    Cache<String, String> threadCache = CacheManager.
        getThreadLocalCacheGroup().getCache("mycache");
    String value = threadCache.get("key");
    if(value == null) {
        // perform expensive operation to calculate value - this
        // processing only happens once for each thread
        ...
        // and store the result
        threadCache.put("key", "value");
    }
    ...
}
```

*Figure 9: Setting up and using a thread local cache*



## 1.6 Transaction Local Caches

A transaction local cache is a cache that lives only during the current transaction. This type of cache is only available in the server application.

### Configuration

Transaction local caches support only a time-based eviction policy.

Only the following two configuration parameters can be used:

- `timeToLive` - this is the maximum number of seconds an item can remain in the cache regardless of use
- `timeToIdle` - this is the maximum number of seconds an item in the cache is allowed to be unused.

The default properties for the parameters are:

- `timeToLive` - 0 (not active)
- `timeToIdle` - 5

The name of the group for transaction local caches is `transaction-group`. This name must be used to configure transaction local caches as shown in the following example:

```
curam.cache.transaction-group.curam.myproject.mycache.timeToLive=60
curam.cache.transaction-group.curam.myproject.mycache.timeToIdle=10
```

Figure 10: Configuring a transaction local cache with name `curam.myproject.mycache`

### Code Samples

Like thread local caches, transaction local caches are accessed only where the correct context (transaction) exists.

```
public void myMethod() {
    ...
    Cache<String, String> txnCache = CacheManagerEjb.
        getTransactionLocalCacheGroup().getCache("mycache");
    String value = txnCache.get("key");
    if(value == null) {
        // perform expensive operation to calculate value - this
        // processing only happens once per transaction
        ...
        // and store the result
        txnCache.put("key", "value");
    }
    ...
}
```

Figure 11: Setting up and using a transaction local cache



# Notices

---

Permissions for the use of these publications are granted subject to the following terms and conditions.

## **Applicability**

These terms and conditions are in addition to any terms of use for the Merative website.

## **Personal use**

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

## **Commercial use**

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

## **Rights**

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

Information concerning non-Merative products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Merative has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Merative products. Questions on the capabilities of non-Merative products should be addressed to the suppliers of those products.

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those

websites are not part of the materials for this Merative product and use of those websites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

#### **COPYRIGHT LICENSE:**

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to Merative, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. Merative, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. Merative shall not be liable for any damages arising out of your use of the sample programs.

## ***Privacy policy***

---

The Merative privacy policy is available at <https://www.merative.com/privacy>.

## ***Trademarks***

---

Merative™ and the Merative™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Adobe™, the Adobe™ logo, PostScript™, and the PostScript™ logo are either registered trademarks or trademarks of Adobe™ Systems Incorporated in the United States, and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft™, Windows™, and the Windows™ logo are trademarks of Microsoft™ Corporation in the United States, other countries, or both.

UNIX™ is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.