SC\_CC

Service Connector Cache Coherency

SC Cache Coherency Model

SC\_CC-V1.0\_E (Version V1.0)

This document describes the SC Cache Coherency Model.

**Copyright © 2010 STABILIT Informatik AG, Switzerland**

**Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at**

**http://www.apache.org/licenses/LICENSE-2.0**

**Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.**

**See the License for the specific language governing permissions and limitations under the License.**

All other logos, product names are trademarks of their respective owners.

This Document has been created with Microsoft Word 2003 (11) with template file and printed at 14 May 2012 14:40.

# Identification

|  |  |
| --- | --- |
| Project: | SC\_CC |
| Title: | Service Connector Cache Coherency |
| Subtitle: | SC Cache Coherency Model |
| Version: | V1.0 |
| Reference: | SC\_CC-V1.0\_E |
| Classification: | Public |
| Keywords: | Architecture, Cache Concept, Coherency Model |
| Comment: | This document describes the SC Cache Coherency Model. |
| Author(s): | STABILIT Informatik AG  Joël Traber |
| Approval (Reviewed by): | Signature Jan Trnka |
| Audience: | Project team, Review team |
| Distribution: | Public |
| Filename |  |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Author | Description |
| 10.05.2012 | D1.0 | Joël Traber | Initial draft |
| 12.06.2012 | V1.0 | Joël Traber | Change cache structure, appendix messages (SIX requirements). One cache per SC! |

# Table of Contents

1 Preface 3

1.1 Purpose & Scope of this Document 3

1.2 Definitions & Abbreviations 3

1.3 External References 3

1.4 Typographical Conventions 3

1.5 Outstanding Issues 3

2 Introduction 4

2.1 Cache coherence problem 4

3 Cache Coherence Model 5

3.1 Cache Coherence 5

3.1.1 Fundamental caching concept 5

3.1.2 Cache Manager 6

3.1.3 Large SCMP Messages 7

3.1.4 Deletion of a cached message 7

3.1.5 Interacting with the cache 7

3.1.6 Clear managed data in caches 7

4 Cache Coherence - SC Client API 8

4.1 SC Client API – Cache Coherence 8

4.2 SC Client API – Loading of large messages 9

5 Monitoring and Troubleshooting 10

5.1 Cache Coherence Logging 10

5.2 Cache Coherence Monitor 10

6 Configuration of Cache Manager 11

6.1 Cache Subscription Service configuration 11

7 Best Practice 12

7.1 Proper separation of static and managed data 12

7.2 Use of the subscription mask 12

8 Glossary 13

Appendix 14

Index 15

# Tables

Table 1 Abbreviations & Definitions 3

Table 2 External references 3

Table 3 Typographical conventions 3

# Figures

Figure 1 Topology & Caching in Service Connector 4

Figure 2 Cache Coherence Model 5

Figure 3 Fundamental Caching Concept 6

Figure 4 SCMessage Hierarchy 8

# Preface

## Purpose & Scope of this Document

This document describes the SC-CC **S**ervice **C**onnector **C**ache **C**oherency Model.

The final and approved version of this document serves as base for publication as Open Source.

This document is particularly important to all project team members and serves as communication medium between them.

## Definitions & Abbreviations

|  |  |
| --- | --- |
| Item / Term | Definition / Description |
| HTTP | Hypertext Transport Protocol |
| HTTPS | HTTP over SSL, encrypted and authenticated transport protocol |

Table Abbreviations & Definitions

## External References

|  |  |
| --- | --- |
| References | Item / Reference to other Document |
| [1] | SC\_0\_Specification\_E – Requirement and Specifications for Service Connector |
| [2] | SC\_0\_SCMP\_E – SC Message Protocol V1.2 |
| [3] | SC\_4\_Operation\_E – Configuration and Operation Guide |
| [4] |  |

Table External references

## Typographical Conventions

|  |  |
| --- | --- |
| Convention | Meaning |
| *text in italics* | features not implemented in the actual release |
| text in Courier font | code example |
| [ phrase ] | In syntax diagrams, indicates that the enclosed values are optional |
| { phrase1 | phrase2 } | In syntax diagrams, indicates that multiple possibilities exists. |
| … | In syntax diagrams, indicates a repetition of the previous expression |

Table Typographical conventions

The terminology used in this document may be somewhat different from other sources. The chapter Glossary includes a list of often used terms with the explanation of their meaning in this document.

## Outstanding Issues

Following issues are outstanding at the time of the document release:

* none

# Introduction

The SC supports message exchange between requesting application (client) and another application providing a service (server). Caching of messages can be activated on every SC.

Cache coherency refers to the consistency of data stored in local caches of every SC node. The coherency protocol described in following sections addresses the problem of maintaining the consistency between all the caches in a system of distributed shared memory. Therefore a coherence model has been specified.



Figure Topology & Caching in Service Connector

## Cache coherence problem

Caching or not caching of a message (response) is an agreement of client and server. The duration until expire date of a cached message describes its **most common lifecycle** which is the timeframe it is valid for clients.

Lots of messages don’t really have a known lifecycle or it is dependent on other client actions. Caching of this type of messages is only possible if we are having a model to keep data consistent. This simply means at the time data changes on server node, caches immediately need to get the new message content and stop satisfying request by handing out obsolete data. Main target of the concept is to keep time between “data change” and “cache update” as small as possible. The client’s actions shouldn’t base on obsolete data.

It is impossible to avoid the coherence problem completely. For a short time period it will always exist.

# Cache Coherence Model

To assure cache consistency Cache Managers are supported by the SC. The main strategy of communication works like a publish service. Any changed data on a server node and known as cached content needs to be published to a Cache Manager. By using the fan out mechanism data update gets populated to the Clients. Updating the cache content is done inside the SC and works according to section 3.1.5Interacting with the cache.



Figure Cache Coherence Model

## Cache Coherence

### Fundamental caching concept

As mentioned earlier, caching or not caching of data is an agreement of client and server. A client indication (cache id) for caching of a message is needed. Afterwards the server confirms caching by returning the same cache id. Indication of the client allows blocking other clients with the same request. If the server disclaims returning the cache id **nothing** gets cached.

The granularity of cached messages exactly correlates to the SCMP messages sent over the wire. Neither restructuring of messages nor modification of bodies will be done!



Figure Fundamental Caching Concept

Caching is completely based on the cache id. Structuring the cached data it’s up to client and server. The concept of used identifiers must be agreed.

### Cache Manager

**Cache Structure**

The cache is divided into static and managed data.

Static data stays in cache until “expiration time” times out or a remove is received. Neither publishing appendixes nor replacements are allowed for static data.

For managed data the SC is responsible for handling the coherency problem. Appendixes or replacements may be applied.

**SC Client API**

The SC Client API subscribes to a Cache Manager. Any changed data published to this Cache Manager by the publish server will be populated up to the client. As long as an API user keeps the cache manager active updates will automatically be received over a callback. At the time the Cache Manager is inactive requested data is still consistent but client is not informed about updates anymore.

**SC**

Cache Managers are defined in the configuration file (sc.properties) of the SCs. The cache is managed by Cache Managers. On a cascaded SC more than one Cache Manage might be active. Messages get loaded by a session request of a client. The first Cache Manager applying an update to a cached message is responsible to keep data consistency. Different messages may be assigned to different Cache Managers.

A Cache Manager stops in following cases:

* No client online (nobody interested in updates)
* Connection loss between client and SC (potential lack of updates)
* Connection loss between SC and SC (potential lack of updates)
* Connection loss between SC and Publish Server (potential lack of updates)

Stopping of a Cache Manager triggers a cleaning procedure in the cache module. Any message the inactive Cache Manager has treated will be removed to avoid the coherence problem. Next client requesting a deleted message causes a new load process**. In clever topology the request will never end up on server level!**

**Publish Server**

A publish server sends updates to a Cache Manager. Messages are populated up to the clients.

The same update might be published of more than one Publish Server to different Cache Managers. Cached messages are only updated of one Cache Manager. Other updates are ignored.

As long as no large messages are publish it’s possible to have more than one server sending updates to the same Cache Manager. Apparently sending the same update twice over each server is nonsense and invalids cached message!

### Large SCMP Messages

For updating a large message in the cache the header attribute “bodyHash” must be set. The publishing server is responsible to set a hash code of complete body in part messages.

To be defined: what algo to use?

### Interacting with the cache

Final decision of making a message cacheable is taken by the session server. SCMP Version 1.3 introduced a new header attribute cachingMethod, which allows the server interacting with the cache. Above actions are done by the Cache Manager depending on the values of cacheMethod.

cachingMethod = “static” or unset

* **Only possible for responds by a session server.**
* Data is cached as static data (no updates possible) until expire time.
* Remove possible by publish server.

cachingMethod = “initial”

* Session server declares data in cache as **managed** data (updates possible).
* Append, initial (replace existing) or remove possible by publish server.
* An initial message published replaces correlating existing initial message and possible appendixes in cache.

cachingMethod = “append”, appendix

* **Only possible by a publish server.**
* Message is appended to cached initial message.
* Appendix ignored if no existing message found in cache.

cachingMethod = “remove”

* **Only possible by a publish server.**
* Removes correlating initial message and possible appendixes in cache.
* Remove will be populated to the client.

### Clear managed data in caches

Managed data in caches are bound to a publish Server. Shutting down or abortion of a publish server clears any managed data in the whole topology bound to this publish server. As well as proper unsubscribe of the publish server causes abortion of subscriptions to this Cache Manager.

# Cache Coherence - SC Client API

The SC Client API supports usage of the cache coherence model by the interface described in following section.

## SC Client API – Cache Coherence

**Starting / Stopping Cache Manager**

Method calls are synchronous and can be done after attaching SCClient successfully. Only one Cache Manager can be started for a SCClient instance.

*startCacheManager(String cacheManagerName, int operationTimeoutSeconds, SCSubscribeMessage scSubscribeMessage, SCMessageCallback scMessageCallback, int receivePublicationTimeoutSeconds) throws SCServiceException, SCMPValidatorException*

* *cacheManagerName*: Identifies the CacheManager
* *operationTimeoutSeconds*: Time until starting of Cache Manager aborts. (Operation timed out)
* *scSubscribeMessage*: Subscribe message (see publish service for more details)
* *scMessageCallback*: Callback to receive cache updates.
* *receivePublicationTimeoutSeconds*: Time to wait for completion of a receive publication request.

Client needs to have an active Cache Manager in order to receive updates over the callback. If no client is interested in updates (no Cache Manager active) no managed data will be cached.

Following method stops the cache subscription service in a shutdown scenario.

*stopCacheManager()*

**Receiving updates or removes**

The callback to receive updates looks like other callbacks in the SC Client API. Only returned object type of the message in receive method may differ.

*public abstract void receive(SCMessage reply);*

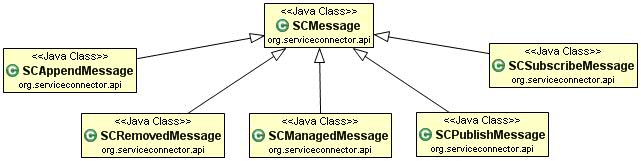
**

Figure SCMessage Hierarchy

**SCAppendMessage**

Indicates that message is an appendix to an initial message. Header attribute appendixNr contains the number of appendixes already applied to the initial message.

**SCRemoveMessage**

Indicates that message has been removed from cache. The cache id as a header attribute identifies deleted message. It does not make sense to deliver content in this message.

**SCManagedMessage**

Type of message for managed cache message read by the SC Client API. Beside the initial message in the body it contains an ordered list of Appendixes as type of SCAppendMessage.

## SC Client API – Loading of large messages

Having an active Cache Manager slightly changes loading of large messages. During the time a client loads a cached large message from the cache, a CACHE\_LOADING exception can be thrown.

During the loading process the cache receives an update for exact this large message. Message state will be changed to loading. Cache stops satisfying requests on this cache id. Client gets a CACHE\_LOADING exception, needs to discard already loaded parts and to retry.

# Monitoring and Troubleshooting

Base concepts of SC monitoring and troubleshooting are described in <SC_4_Operation_E.pdf>. Subsequent section covers monitoring and troubleshooting for the context of cache coherence.

## Cache Coherence Logging

The **cacheLogger** is responsible for logging the events in context of caching.

Following events will be logged:

* New subscription on Cache Manager.
* New managed message received.
* New Appendix message received.
* New Initial message received, existing data replaced.
* New Remove messaged received, existing data completely removed.
* Appendix error because no corresponding “cacheId” found.
* Broken Cache Manager-> Managed data removed.
* No more subscriptions on Cache Manager -> Managed data removed.

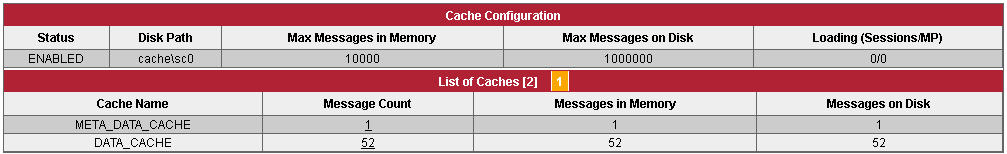
## Cache Coherence Monitor

Below mentioned information in the context of cache coherence model are visible in the Web UI.

* Static data with expiration time.
* Managed data
  + Creation time of managed message (initial message received)
  + Available appendixes, total number of
  + Time last appended message arrived
  + Cache Manager assigned to this message
* State of Cache Manager (active or inactive)
  + Active or inactive
  + Nr of subscriptions to the Cache Manager

Cache

The cache form shows the state of the cache.



# Configuration of Cache Manager

The SC is configured with a single configuration file sc.properties. More details are described in <SC_4_Operation_E.pdf>. Configuration of Cache Manager is also done in sc.properties.

## Cache Manager Configuration

Names of the Cache Managers have to be listed as shown below. A remote node defines which cascaded SC (only one) is publishing messages to a specific Cache Manager.

…...

cacheManagers=sc1-cacheMgr, sc2-cacheMgr

…...

sc1-cacheMgr.type=CacheManager

sc1-cacheMgr.enabled=true

sc1-cacheMgr.remoteNode=sc1

sc1-cacheMgr.noDataIntervalSeconds=10

sc2-cacheMgr2.type=CacheManager

sc2-cacheMgr2.enabled=true

sc2-cacheMgr2.remoteNode=sc2

sc2-cacheMgr2.noDataIntervalSeconds=10

…...

# Best Practice

## Proper separation of static and managed data

Chapter 3.1.2 and 3.1.6 explain the handling of managed data when a lack of updates occurs. Basically the deletion of managed data by a Cache Manager it’s a normal procedure to avoid cache coherency problem. For this reason the separation of static and managed data is very important. Data without the need to be updated for a longer time period should be declared static. Apparently the granularity of the messages has impact to the concept as well.

## Use of the subscription mask

Like a publish service does a cache Manager support the usage of the subscription mask. Published updates will be broadcasted according to the subscription mask. It’s up to server and clients agreement to wisely use it.

If server and client agree on a ordered list of all cache ids and use the particular positions to identify the mask bit, client may precisely subscribe for the updates he needs. This also reduces network traffic between proxies.

# Glossary

|  |  |
| --- | --- |
| Term | Explanation |
| Appendix | Append message referring to an initial message. |
| Cache Coherency / Coherence | Cache coherence (also cache coherency) refers to the consistency of data stored in multiple [caches](http://en.wikipedia.org/wiki/Cache_%28computing%29) of a shared resource |
| cache id | Message identifier in the cache. Must be unique in the cache. Controlled by client and server. |
| Cache Manager | SC Module to guarantee data consistency in a cache. Responsible for the treatment of specific messages in a cache. |
| Data Consistency | Data consistency summarizes the validity, accuracy, usability and integrity of related data. |
| Initial message | Base message appendixes may be applied to. |
| Managed data | Data in cache the SC takes care of consistency. Initial and append messages are managed data. |
| Static data | Data valid until expiration time. No appendixes able to apply to. |

1. Appendix

C

Cache • 10