



Coherence Spring Reference Documentation

Gunnar Hillert

Version 3.0.0-SNAPSHOT

Table of Contents

Legal	1
1. Coherence Spring Documentation	2
1.1. About the Documentation	2
1.2. Getting Help	2
1.3. What is new?	2
1.4. First Steps	3
2. Quickstart	4
2.1. How to Run the Demo	4
2.2. Interacting with the Cache	5
2.3. Behind the Scenes	7
3. Coherence Spring Core	9
3.1. Getting Started	9
4. Coherence Spring Cache	10
4.1. Introduction	10
4.2. Configuring Coherence Cache for Spring	10
5. Coherence Spring Session	12
5.1. Getting Started	12
6. Coherence Spring Data	13
6.1. Getting Started	13
7. Coherence Spring Boot	14
7.1. Getting Started	14
7.2. Using Coherence with Spring Boot	14
7.3. Customize Coherence	16
7.4. Using Coherence as Spring Caching Provider	16
8. Coherence Spring Cloud Config	17
8.1. Overview	17
8.2. Demo	17
8.3. Coherence Backend	18
9. Appendices	19

Legal

Oracle licenses the Oracle Coherence Spring project under the [The Universal Permissive License \(UPL\), Version 1.0](#).

The Universal Permissive License (UPL), Version 1.0

Subject to the condition set forth below, permission is hereby granted to any person obtaining a copy of this software, associated documentation and/or data (collectively the "Software"), free of charge and under any and all copyright rights in the Software, and any and all patent rights owned or freely licensable by each licensor hereunder covering either (i) the unmodified Software as contributed to or provided by such licensor, or (ii) the Larger Works (as defined below), to deal in both

(a) the Software, and (b) any piece of software and/or hardware listed in the `lrgwrks.txt` file if one is included with the Software (each a "Larger Work" to which the Software is contributed by such licensors),

without restriction, including without limitation the rights to copy, create derivative works of, display, perform, and distribute the Software and make, use, sell, offer for sale, import, export, have made, and have sold the Software and the Larger Work(s), and to sublicense the foregoing rights on either these or other terms.

This license is subject to the following condition: The above copyright notice and either this complete permission notice or at a minimum a reference to the UPL must be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Chapter 1. Coherence Spring Documentation

Welcome to the reference documentation of [Coherence Spring](#), a collection of several libraries that will help you to integrate [Coherence](#) with the wider [Spring](#) ecosystem.

This section provides a brief overview of the Coherence Spring reference documentation.

1.1. About the Documentation

The Coherence Spring reference guide is available as:

- [Multi-page HTML](#)
- [Single page HTML](#)
- [PDF](#)

1.2. Getting Help

If you run into issues with Spring Coherence, we are here to help.

- *Try the [Quickstart](#).* The Quickstart will give you an overview of Coherence Spring's capabilities and provides a sample application to get you started.
- *Learn the Coherence basics.* Please have at least some basic understanding of Oracle Coherence since all Spring Coherence modules depend on it. Check out the [Coherence CE](#) web-site for general Coherence targeted reference documentation.
- *Learn the Spring basics.* The reference guide assumes that you have a basic understanding of [Spring Framework](#) and [Spring Boot](#). Coherence Spring utilizes several other Spring projects. Check the [spring.io](#) web-site for general reference documentation. If you are starting out with Spring, try one of the [guides](#) or generate a starter project using [start.spring.io/](#).
- *Ask a question.* Chat with us directly on [Slack](#). We also monitor [stackoverflow.com](#) for questions tagged with [oracle-coherence](#).
- *Contribute.* Report bugs with Spring Coherence via [GitHub Issues](#). Both, Coherence CE and Coherence Spring are Open Source Software (OSS) under the liberal [Universal Permissive License \(UPL\)](#). Contributing back is a great way to attain a deeper understanding of our projects.



All of *Coherence Spring* is open source, including the documentation. If you find problems with the docs or if you want to improve them, please [get involved](#).

1.3. What is new?

In order to see what changes were made from earlier versions of Coherence Spring, see the [Change History](#) as well as the [GitHub Releases](#) page.

1.4. First Steps

If you are getting started with Coherence Spring, start with the [Quickstart](#). It is a great way to see a working solution quickly. Particularly if you are relatively new to Spring, continue with the [Coherence Spring Boot](#) chapter next. The reference documentation makes a distinction between [Spring Framework](#) and [Spring Boot](#). At its very core, Spring Framework provides Dependency Injection (DI) or Inversion Of Control (IOC) to Java applications. Furthermore, Spring Framework gives developers comprehensive infrastructure support for developing Java applications.

Spring Boot on the other hand, is an opinionated extension to the Spring Framework by:

- Eliminating boilerplate configurations
- Providing Auto-Configuration for other Spring modules and third-party integrations
- Metrics + health checks

The vast majority of new Spring projects will utilize Spring Boot. Nonetheless, please also study the Spring Framework targeted chapters as Spring Framework is the foundation for everything related to Spring Boot.

Chapter 2. Quickstart

In this getting started chapter we will look a demo to illustrate basic usage of Oracle Coherence when using it with Spring. This demo provides an example of using Coherence Spring's [Cache Abstraction](#).

The demo application is basically a super-simple event manager. We can create **Events** and assign **People** to them using an exposed REST API. The data is saved in an embedded HSQL database. The caching is implemented at the service layer:

When an **Event** is created, it is not only persisted to the database but also *put* to the Coherence Cache. Therefore, whenever an **Event** is retrieved, it will be returned from the Coherence Cache. You can also delete **Events**, in which case the **Event** will be *evicted* from the cache. You can perform the same [CRUD](#) operations for people as well.

2.1. How to Run the Demo

In order to get started, please checkout the code from the [Coherence Spring Demo](#) GitHub repository.

Clone GitHub Repository

```
$ git clone https://github.com/ghillert/coherence-spring-demo.git
$ cd coherence-spring-demo
```

This repository contains a Maven project consisting of 3 modules. The first two Maven modules are essentially variations of the same app. The third module contains shared code.

coherence-spring-demo-classic	Provides a demo using Spring Framework without Spring Boot
coherence-spring-demo-boot	Provides a demo using Spring Boot
coherence-spring-demo-core	Contains common code shared between the two apps

In this chapter we will focus on the **Spring Boot** version. Since we checked out the project, let's build it using Maven:

Build the project

```
$ ./mvnw clean package
```

Now we are ready to run the application:

Run the Spring Boot application

```
$ java -jar coherence-spring-demo-boot/target/coherence-spring-demo-boot-1.0.0-SNAPSHOT.jar
```

2.2. Interacting with the Cache

Once the application is started, the embedded database is empty. Let's create an event with 2 people added to them using [curl](#):

Create the first event

```
$ curl --request POST  
'http://localhost:8080/api/events?title=First%20Event&date=2020-11-30'
```

This call will create and persist an **Event** to the database. However, there is more going on. The created **Event** is also added to the Coherence Cache. The magic is happening in the Service layer, specifically in `DefaultEventService#createAndStoreEvent()`, which is annotated with `@CachePut(cacheNames="events", key="#result.id")`.

The `cacheNames` attribute of the `@CachePut` annotation indicates the name of the underlying cache to use. As caches are basically just a Map, we also need a key. In this case we use the expression `#result.id` to retrieve the primary key of the **Event** as it was persisted. Thus, the saved **Event** is added to the cache named `events` and ultimately also returned and printed to the console:

Return result of the created event

```
{  
  "id" : 1,  
  "title" : "First Event",  
  "date" : "2020-11-30T00:00:00.000+00:00"  
}
```

We see that an Event with the id **1** was successfully created. Let's verify that the *cache put* worked by looking at the cache's statistics:

Retrieving Cache Statistics

```
$ curl --request GET 'http://localhost:8080/api/statistics/events'
```

In the console you should see some basic statistics being printed including `totalPuts : 1`:

Cache Statistic Results

```
{
  "averageMissMillis" : 0.0,
  "cachePrunesMillis" : 0,
  "averagePruneMillis" : 0.0,
  "totalGetsMillis" : 0,
  "averageGetMillis" : 0.0,
  "totalPutsMillis" : 11,
  "averagePutMillis" : 11.0,
  "cacheHitsMillis" : 0,
  "averageHitMillis" : 0.0,
  "cacheMissesMillis" : 0,
  "cacheHits" : 0,
  "cacheMisses" : 0,
  "hitProbability" : 0.0,
  "totalPuts" : 1,
  "totalGets" : 0,
  "cachePrunes" : 0
}
```

Next, lets retrieve the Event using id 1:

Retrieve Event

```
curl --request GET 'http://localhost:8080/api/events/1'
```

The Event is returned. Did you notice? No SQL queries were executed as the value was directly retrieved from the Cache. Let's check the statistics again by executing:

Retrieve Cache Statistics

```
curl --request GET 'http://localhost:8080/api/statistics/events'
```

We will see now how values are being returned from the cache by seeing increasing cacheHits, e.g. **"cacheHits" : 1**,. Let's evict our Event with id 1 from the cache named events:

Evict Event

```
curl --request DELETE 'http://localhost:8080/api/events/1'
```

If you now retrieve the event again using:

Retrieve Event

```
curl --request GET 'http://localhost:8080/api/events/1'
```

you will see an SQL query executed in the console, re-populating the cache. Feel free to play along

with the Rest API. We can for example add people:

Add people

```
curl --request POST
'http://localhost:8080/api/people?firstName=Conrad&lastName=Zuse&age=85'
curl --request POST
'http://localhost:8080/api/people?firstName=Alan&lastName=Turing&age=41'
```

Or assign people to events:

Assign People to Events

```
curl --request POST 'http://localhost:8080/api/people/2/add-to-event/1'
curl --request POST 'http://localhost:8080/api/people/3/add-to-event/1'
```

2.3. Behind the Scenes

What is involved to make this all work? Using Spring Boot, the setup is incredibly simple. We take advantage of Spring Boot's [AutoConfiguration](#) capabilities, and the sensible defaults provided by *Coherence Spring*.

In order to activate AutoConfiguration for Coherence Spring you need to add the `coherence-spring-boot-starter` dependency as well as the desired dependency for Coherence.

POM configuration

```
<dependency>
  <groupId>com.oracle.coherence.spring</groupId>
  <artifactId>coherence-spring-boot-starter</artifactId> ①
  <version>3.0.0-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>com.oracle.coherence.ce</groupId>
  <artifactId>coherence</artifactId> ②
  <version>20.12</version>
</dependency>
```

① Activate Autoconfiguration by adding the `coherence-spring-boot-starter` dependency

② Add the desired version of Coherence (CE or Commercial)

In this quickstart example we are using Spring's Caching abstraction and therefore, we use the `spring-boot-starter-cache` dependency as well:

POM configuration for Spring Cache Abstraction

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

For caching you also must activate caching using the `@EnableCaching` annotation.

Spring Boot App configuration

```
@SpringBootApplication
@EnableCaching
public class CoherenceSpringBootDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(CoherenceSpringBootDemoApplication.class, args);
    }

}
```

① Activate the Spring Cache Abstraction

Please see the relevant chapter on [Caching](#) in the Spring Boot reference guide.

With `@EnableCaching` in place, Coherence's autoconfiguration will also provide a `CoherenceCacheManager` bean to the application context.

Chapter 3. Coherence Spring Core

This section dives into the Coherence Spring Core module. Coherence Spring Core provides the basic support for the [Spring Framework](#).

3.1. Getting Started

The main building block for setting up Coherence for Spring is the `@EnableCoherence` annotation. This annotation will import the `CoherenceSpringConfiguration` under the covers. Therefore, you can alternatively also declare `@Import(CoherenceSpringConfiguration.class)` instead.

Without providing any further configuration various defaults are applied.

Chapter 4. Coherence Spring Cache

This section dives into the Coherence Spring Cache module. It explains how to use Coherence's support for the Spring Framework's [Cache Abstraction](#).

4.1. Introduction

Spring provides its own cache abstraction, allowing you to add caching to Java methods. Coherence Spring provides an implementation of this abstraction for Oracle Coherence.



Spring's Cache abstraction also supports [JSR-107](#) which is also supported by Oracle Coherence. As such you have another alternative for setting up caching.



If you are using JPA/Hibernate you may also consider using the Coherence support for Hibernate's second-level cache SPI, which is provided by the [Coherence Hibernate project](#).

4.2. Configuring Coherence Cache for Spring

As a start, please familiarize yourself with Spring's Cache Abstraction by reading the [relevant section](#) of Spring's reference documentation.

Properties

Yaml

```
example:
  property:
    alpha: a
```

Properties

Yaml

```
spring:
  devtools:
    restart:
      exclude: "static/**,public/**"
```

Example 1. Creating a CoherenceInstance

Java

```
@Configuration
@EnableCaching
public class CacheConfiguration {

    @Bean
    public CoherenceInstance coherenceInstance() {
        return new CoherenceInstance();
    }

    @Bean
    public CacheManager cacheManager(CoherenceInstance coherenceInstance) {
        return new CoherenceCacheManager(coherenceInstance);
    }
}
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cache="http://www.springframework.org/schema/cache"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/cache
        https://www.springframework.org/schema/cache/spring-cache.xsd">

    <cache:annotation-driven/>

    <bean id="coherenceInstance"
class="com.oracle.coherence.spring.CoherenceInstance"/>

    <bean id="cacheManager"
class="com.oracle.coherence.spring.cache.CoherenceCacheManager">
        <constructor-arg ref="coherenceInstance"/>
    </bean>
</beans>
```

Chapter 5. Coherence Spring Session

This section dives into the Coherence Spring Session module. It explains how to use Coherence's support for [Spring Session](#).

5.1. Getting Started

TBD

Chapter 6. Coherence Spring Data

This section dives into the Coherence Spring Data module. It explains how to use Coherence's support for [Spring Data](#) repositories.

6.1. Getting Started

TBD

Chapter 7. Coherence Spring Boot

This section dives into the Coherence Spring Boot module. It explains how to use Coherence's dedicated support for [Spring Boot](#), e.g. Autoconfiguration.

7.1. Getting Started

In order to start using Coherence with Spring Boot you have to add the `coherence-spring-boot-starter` dependency as well as the desired version of Coherence.

Maven

```
<dependencies>
  <dependency>
    <groupId>com.oracle.coherence.spring</groupId>
    <artifactId>coherence-spring-boot-starter</artifactId>
    <version>3.0.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>com.oracle.coherence.ce</groupId>
    <artifactId>coherence</artifactId>
    <version>20.12</version>
  </dependency>
</dependencies>
```

Gradle

```
dependencies {
    compile("com.oracle.coherence.spring:coherence-spring-boot-starter:3.0.0-SNAPSHOT")
    compile("com.oracle.coherence.ce:coherence:20.12")
}
```



As Coherence Spring takes advantage of the new Coherence Bootstrap API, it requires Oracle Coherence CE version **20.12** or higher.

7.2. Using Coherence with Spring Boot

By adding the `coherence-spring-boot-starter` dependency, AutoConfiguration will be activated via the `CoherenceAutoConfiguration` class. This will also bind the `CoherenceProperties` for further configuration. The configuration for Spring Boot's Coherence support may look like the following:

Example YAML configuration (Properties)

Example YAML configuration (Yaml)

```
coherence:
  logging:
    destination: slf4j
    logger-name: MyCoherence
  sessions:
    - name: default
      config: "coherence-cache-config.xml"
      priority: 1
    - name: test
      config: "test-coherence-config.xml"
      priority: 2
  properties:
    coherence.log.limit: 400
    coherence.log.level: 1
```

The following configuration properties are available.

Table 1. Coherence Configuration Properties

Key	Default Value	Description
coherence.logging.destination		The type of the logging destination. Default to slf4j if not set.
coherence.logging.severity-level		Specifies which logged messages are emitted to the log destination. The legal values are -1 to 9 . No messages are emitted if -1 is specified. More log messages are emitted as the log level is increased.
coherence.logging.logger-name		
coherence.logging.message-format		
coherence.logging.character-limit		
coherence.properties.*		Any native Coherence properties
coherence.sessions[0].name		
coherence.sessions[0].type		Represents the various session type that can be configured: CLIENT, SERVER, GRPC

Key	Default Value	Description
coherence.sessions[0].config		The Coherence cache configuration URI for the session
coherence.sessions[0].priority		The priority order to be used when starting the session. Sessions will be started with the lowest priority first.
coherence.sessions[0].scope-name		The scope name for the session.



All but the session property are translated into native Coherence properties. If both Spring Boot property AND a native property `coherence.properties.*` are configured, the Spring Boot property is used.

For a list of available native properties, please consult the reference guide chapter on [System Property Overrides](#).

7.3. Customize Coherence

7.4. Using Coherence as Spring Caching Provider

If caching is enabled via `@EnableCaching`, Coherence Autoconfiguration will it automatically provide a `CacheManager` to the `ApplicationContext`, however only if no `CacheManager` was configured explicitly beforehand.

Chapter 8. Coherence Spring Cloud Config

This section explains how to configure Coherence using [Spring Cloud Config](#). Furthermore, this chapter also shows how to use [Coherence](#) as a Spring Cloud Config storage backend, allowing you to set up Spring applications with configuration data stored in Coherence.

8.1. Overview

Spring Cloud Config provides support for externalized configuration in distributed systems. It integrates seamlessly with Spring Boot applications and allows you to externalize / centralize critical application properties. Spring Cloud Config provides numerous storage backends for your configuration data.



Please familiarize yourself with the [Spring Cloud Config reference documentation](#).

In this chapter we will cover two aspects of Coherence-specific support for Spring Cloud Config:

- Configure Coherence and its Spring support using Spring Cloud Config
- Use Oracle Coherence as a configuration backend for Spring Cloud Config and thus store your Configuration data in a Coherence cluster

Let's get started with an example to show the general functioning of Spring Cloud Config.

8.2. Demo

This demo is essentially the same as used in the [Quickstart](#) chapter. However, we externalize some Coherence configuration using Spring Cloud Config. To get started, please clone the demo projects:

Clone the Spring Cloud Config demo project

```
git clone https://github.com/ghillert/coherence-spring-cloud-config-demo
cd coherence-spring-cloud-config-demo
```

The project consists of 2 Maven modules:

-

The Config Server is essentially using 3 dependencies:

Maven

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Gradle

```
dependencies {
    compile("com.oracle.coherence.spring:coherence-spring-boot-starter:3.0.0-SNAPSHOT")
    compile("com.oracle.coherence.ce:coherence:20.12")
}
```

The demo client will use the following dependencies:

Maven

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

curl coherence:c0h3r3nc3r0ck5@localhost:8888/config-client/development/master

8.3. Coherence Backend

TBD

Chapter 9. Appendices