# Spring Gemfire Integration Reference Guide

1.0.0.RELEASE

Costin Leau (SpringSource, a division of VMware)

# Preface

Spring GemFire Integration focuses on integrating Spring Framework's powerful, non-invasive programming model and concepts with Gemstone's GemFire Enterprise Fabric, providing easier configuration, use and high-level abstractions. This document assumes the reader already has a basic familiarity with the Spring Framework and GemFire concepts and APIs.

While every effort has been made to ensure that this documentation is comprehensive and there are no errors, nevertheless some topics might require more explanation and some typos might have crept in. If you do spot any mistakes or even more serious errors and you can spare a few cycles during lunch, please do bring the error to the attention of the Spring GemFire Integration team by raising an issue. Thank you.

# Part I. Introduction

This document is the reference guide for Spring GemFire project (SGF). It explains the relationship between Spring framework and GemFire Enterprise Fabric (GEF) 6.0.x, defines the basic concepts and semantics of the integration and how these can be used effectively.

# Part II. Reference Documentation

# Document structure

This part of the reference documentation explains the core functionality offered by Spring GemFire integration.

Chapter 1, *Bootstrapping GemFire through the Spring container* describes the configuration support provided for bootstrapping, initializing and accessing a GemFire cache or region.

Chapter 2, *Working with the GemFire APIs* explains the integration between GemFire API and the various "data" features available in Spring, such as transaction management and exception translation.

Chapter 3, *Working with GemFire Serialization* describes the enhancements for GemFire (de)serialization process and management of associated objects.

Chapter 4, *Sample Applications* describes the samples provided with the distribution for showcasing the various features available in Spring GemFire.

# Chapter 1. Bootstrapping GemFire through the Spring container

One of the first tasks when using GemFire and Spring is to configure the data grid through the IoC container. While this is possible out of the box, the configuration tends to be verbose and only address basic cases. To address this problem, the Spring GemFire project provides several classes that enable the configuration of distributed caches or regions to support a variety of scenarios with minimal effort.

## 1.1. Using the Spring GemFire Namespace

To simplify configuration, SGF provides a dedicated namespace for most of its components. However, one can opt to configure the beans directly through the usual <bean> definition. For more information about XML Schema-based configuration in Spring, see this appendix in the Spring Framework reference documentation.

To use the SGF namespace, one just needs to import it inside the configuration:

```
<?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:gfe❶="http://www.springframework.org/schema/gemfire❷"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/gemfire http://www.springframework.org/schema/gemfire/spring-gemfi
       <bean id ... >

    <gfe❹:cache ...>

</beans>
```

❶ Spring GemFire namespace prefix. Any name can do but through out the reference documentation, the `gfe` will be used.
❷ The namespace URI.
❸ The namespace URI location. Note that even though the location points to an external address (which exists and is valid), Spring will resolve the schema locally as it is included in the Spring GemFire library.
❹ Declaration example for the GemFire namespace. Notice the prefix usage.

Once declared, the namespace elements can be declared simply by appending the aforementioned prefix. Note that is possible to change the default namespace, for example from `<beans>` to `<gfe>`. This is useful for configuration composed mainly of GemFire components as it avoids declaring the prefix. To achieve this, simply swap the namespace prefix declaration above:

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

    <beans:bean id ... >                                       ❸

    <cache ...>                                                ❹

</beans>
```

❶ The default namespace declaration for this XML file points to the Spring GemFire namespace.
❷ The beans namespace prefix declaration.

❸ Bean declaration using the `<beans>` namespace. Notice the prefix.
❹ Bean declaration using the `<gfe>` namespace. Notice the lack of prefix (as the default namespace is used).

For the remainder of this doc, to improve readability, the XML examples will simply refer to the `<gfe>` namespace without the namespace declaration, where possible.

## 1.2. Configuring the GemFire `cache`

In order to use the GemFire Fabric, one needs to either create a new `Cache` or connect to an existing one. As in the current version of GemFire, there can be only one opened cache per VM (or classloader to be technically correct). In most cases the cache is created once and then all other consumers connect to it.

In its simplest form, a cache can be defined in one line:

```
<gfe:cache />
```

The declaration above declares a bean(`CacheFactoryBean`) for the GemFire Cache, named `gemfire-cache`. All the other SGF components use this naming convention if no name is specified, allowing for very concise configurations. The definition above will try to connect to an existing cache and, in case one does not exist, create it. Since no additional properties were specified the created cache uses the default cache configuration.Especially in environments with opened caches, this basic configuration can go a long way.

For scenarios where the cache needs to be configured, the user can pass in a reference the GemFire configuration file:

```
<gfe:cache id="cache-with-xml" cache-xml-location="classpath:cache.xml"/>
```

In this example, if the cache needs to be created, it will use the file named `cache.xml` located in the classpath root. Only if the cache is created will the configuration file be used.

### Note

Note that the configuration makes use of Spring's `Resource` abstraction to locate the file. This allows various search patterns to be used, depending on the running environment or the prefix specified (if any) by the value.

In addition to referencing an external configuration file one can specify GemFire settings directly through Java `Properties`. This can be quite handy when just a few settings need to be changed.

To setup properties one can either use the `properties` element inside the `util` namespace to declare or load properties files (the latter is recommended for externalizing environment specific settings outside the application configuration):

```
<?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:gfe="http://www.springframework.org/schema/gemfire"
    xmlns:util="http://www.springframework.org/schema/util"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/be
        http://www.springframework.org/schema/gemfire http://www.springframework.org/schema/gemfire/spring-gemfi
        http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd">

     <gfe:cache id="cache-with-xml" cache-xml-location="classpath:cache.xml" properties-ref="props"/>

     <util:properties id="props" location="classpath:/deployment/env.properties"/>
</beans>
```

Or can use fallback to a *raw* `<beans>` declaration:

```
<bean id="cache-with-props" class="org.springframework.data.gemfire.CacheFactoryBean">
    <property name="properties">
        <props>
           <prop key="bind-address">127.0.0.1</prop>
        </props>
    </property>
</bean>
```

In this last example, the SGF classes are declared and configured directly without relying on the namespace. As one can tell, this approach is a generic one, exposing more of the backing infrastructure.

It is worth pointing out again, that the cache settings apply only if the cache needs to be created, there is no opened cache in existence otherwise the existing cache will be used and the configuration will simply be discarded.

## 1.3. Configuring a GemFire `Region`

Once the `Cache` is configured, one needs to configure one or more `Regions` to interact with the data fabric. SGF allows various region types to be configured and created directly from Spring or in case they are created directly in GemFire, retrieved as such.

For more information about the various region types and their capabilities as well as configuration options, please refer to the GemFire Developer's Guide and community site.

### 1.3.1. Using an externaly configured `Region`

For consuming but not creating `Regions` (for example in case, the regions are already configured through GemFire native configuration, the `cache.xml`), one can use the `lookup-region` element. Simply declare the target region name the `name` attribute; for example to declare a bean definition, named `region-bean` for an existing region named `orders` one can use the following definition:

```
<gfe:lookup-region id="region-bean" name="orders"/>
```

If the `name` is not specified, the bean name will be used automatically. The example above becomes:

```
<!-- lookup for a region called 'orders' -->
<gfe:lookup-region id="orders"/>
```

**Note**

If the region does not exist, an initialization exception will be thrown. See the section below on how to configure GemFire regions.

Note that in the previous examples, since no cache name was defined, the default SGF naming convention (`gemfire-cache`) was used. If that is not an option, one can point to the cache bean through the `cache-ref` attribute:

```
<gfe:cache id="cache"/>

<gfe:lookup-region id="region-bean" name="orders" cache-ref="cache"/>
```

The `lookup-region` provides a simple way of retrieving existing, pre-configured regions without exposing the region semantics or setup infrastructure.

### 1.3.2. Replicated Region

One of the common region types supported by GemFire is *replicated region* or *replica*. In short:

**What is a replica?**

When a region is configured to be a replicated region, every member that hosts that region stores a copy of the contents of the region locally. Any update to a replicated region is distributed to all copies of the region. [...] When a replica is created, it goes through an initialization stage in which it discovers other replicas and automatically copies all the entries. While one replica is initializing you can still continue to use the other replicas.

SGF offers a dedicated element for creating replicas in the form of `replicated-region` element. A minimal declaration looks as follows (again, the example will not setup the cache wiring, relying on the SGF namespace naming conventions):

```
<gfe:replicated-region id="simple-replica" />
```

Here, a replicated region is created (if one doesn't exist already). The name of the region is the same as the bean name (`simple-replica`) and the bean assumes the existence of a GemFire cache named `gemfire-cache`.

When setting a region, it's fairly common to associate various `CacheLoaders`, `CacheListeners` and `CacheWriters` with it. These components can be either referrenced or declared inlined by the region declaration.

**Note**

Following the GemFire recommandations, the namespace allows for each region created multiple listeners but only one cache writer and cache loader. This restriction can be relaxed, for advanced usages by using the `beans` declaration (see the next section).

Below is an example, showing both styles:

```
<gfe:replicated-region id="mixed">
    <gfe:cache-listener>
            <!-- nested cache listener reference -->
        <ref bean="c-listener"/>
        <!-- nested cache listener declaration -->
        <bean class="some.pkg.SimpleCacheListener"/>
    </gfe:cache-listener>
    <!-- loader reference -->
    <gfe:cache-loader ref="c-loader"/>
    <!-- writer reference -->
    <gfe:cache-writer ref="c-writer"/>
</gfe:replicated-region>
```

**Warning**

Using `ref` and a nested declaration on `cache-listener`, `cache-loader` or `cache-writer` is illegal. The two options are mutually exclusive and using them at the same time, on the same element will throw an exception.

#### 1.3.2.1. `replicated-region` Options

The following table offers a quick overview of the most important configuration options names, possible values and short descriptions for each of settings supported by the `replicated-region` element. Please see the storage and eviction section for the relevant configuration.

**Table 1.1. replicated-region options**

| Name | Values | Description |
|---|---|---|
| id | *any valid bean name* | The id of the region bean definition. |
| name | *any valid region name* | The name of the region definition. If no specified, it will have the value of the id attribute (that is, the bean name). |
| cache-ref | *GemFire cache bean name* | The name of the bean defining the GemFire cache (by default 'gemfire-cache'). |
| cache-listener | *valid bean name or definition* | The name or nested bean declaration of a GemFire `CacheListener`. |
| cache-loader | *valid bean name or definition* | The name or nested bean declaration of a GemFire `CacheLoader`. |
| cache-writer | *valid bean name or definition* | The name or nested bean declaration of a GemFire `CacheWriter`. |

### 1.3.3. Partition(ed) Region

Another region type supported out of the box by the SGF namespace, is the partitioned region. To quote again the GemFire docs:

#### What is a partition?

A partitioned region is a region where data is divided between peer servers hosting the region so that each peer stores a subset of the data. When using a partitioned region, applications are presented with a logical view of the region that looks like a single map containing all of the data in the region. Reads or writes to this map are transparently routed to the peer that hosts the entry that is the target of the operation. [...] GemFire divides the domain of hashcodes into buckets. Each bucket is assigned to a specific peer, but may be relocated at any time to another peer in order to improve the utilization of resources across the cluster.

A partition can be created by SGF through the `partitioned-region` element. Its configuration options are similar to that of the `replicated-region` plus the partion specific features such as the number of redundant copies, total maximum memory, number of buckets, partition resolver and so on. Below is a quick example on

setting up a partition region with 2 redundant copies:

```
<!-- bean definition named 'distributed-partition' backed by a region named 'redundant' with 2 copies
and a nested resolver declaration  -->
<gfe:partitioned-region id="distributed-partition" copies="2" total-buckets="4" name="redundant">
    <gfe:partition-resolver>
        <bean class="some.pkg.SimplePartitionResolver"/>
    </gfe:partition-resolver>
</gfe:partitioned-region>
```

#### 1.3.3.1. `partitioned-region` Options

The following table offers a quick overview of the most important configuration options names, possible values and short descriptions for each of settings supported by the partition element. Please see the storage and eviction section for the relevant configuration.

**Table 1.2. partitioned-region options**

| Name | Values | Description |
|---|---|---|
| id | *any valid bean name* | The id of the region bean definition. |
| name | *any valid region name* | The name of the region definition. If no specified, it will have the value of the id attribute (that is, the bean name). |
| cache-ref | *GemFire cache bean name* | The name of the bean defining the GemFire cache (by default 'gemfire-cache'). |
| cache-listener | *valid bean name or definition* | The name or nested bean declaration of a GemFire `CacheListener`. |
| cache-loader | *valid bean name or definition* | The name or nested bean declaration of a GemFire `CacheLoader`. |
| cache-writer | *valid bean name or definition* | The name or nested bean declaration of a GemFire `CacheWriter`. |
| partition-resolver | *bean name* | The name of the partitioned resolver used by this region, for custom partitioning. |
| copies | 0..4 | The number of copies for each partition for high-availability. By default, no copies are created meaning there is no redundancy. Each copy provides extra backup at the expense of extra storages. |
| colocated-with | *valid region name* | The name of the partitioned region with which this newly created partitioned region is colocated. |

| Name | Values | Description |
|---|---|---|
| local-max-memory | *positive integer* | The maximum amount of memory, in megabytes, to be used by the region in *this* process. |
| total-max-memory | *any integer value* | The maximum amount of memory, in megabytes, to be used by the region in *all* processes. |
| recovery-delay | *any long value* | The delay in milliseconds that existing members will wait before satisfying redundancy after another member crashes. -1 (the default) indicates that redundancy will not be recovered after a failure. |
| startup-recovery-delay | *any long value* | The delay in milliseconds that new members will wait before satisfying redundancy. -1 indicates that adding new members will not trigger redundancy recovery. The default is to recover redundancy immediately when a new member is added. |

### 1.3.4. Client Region

GemFire supports various deployment topologies for managing and distributing data. The topic is outside the scope of this documentation however to quickly recap, they can be categoried in short in: peer-to-peer (p2p), client-server (or super-peer cache network) and wide area cache network (or WAN). In the last two scenarios, it is common to declare *client* regions which connect to a backing cache server (or super peer). SGF offers dedicated support for such configuration through the `client-region` and `pool` elements. As the name imply, the former defines a client region while the latter connection pools to be used/shared by the various client regions.

Below is a usual configuration for a client region:

```
<!-- client region declaration -->
<gfe:client-region id="complex" pool-name="gemfire-pool">
    <gfe:cache-listener ref="c-listener"/>
</gfe:client-region>

<bean id="c-listener" class="some.pkg.SimpleCacheListener"/>

<!-- pool declaration -->
<gfe:pool id="gemfire-pool" subscription-enabled="false">
    <gfe:locator host="localhost" port="40403"/>
</gfe:pool>
```

Just as the other region types, `client-region` allows defining `CacheListeners`. It also relies on the same naming conventions in case the region name or the cache are not set explicitely. However, it also requires a connection `pool` to be specified for connecting to the server. Each client can have its own pool or they can share the same one.

For a full list of options to set on the client and especially on the pool, please refer to the SGF schema

Appendix A, *Spring GemFire Integration Schema*) and the GemFire documentation.

#### 1.3.4.1. Client Interests

To minimize network traffic, each client can define its own 'interest', pointing out to GemFire, the data it actually needs. In SGF, interests can be defined for each client, both key-based and regular-expression-based types being supported; for example:

```
<gfe:client-region id="complex" pool-name="gemfire-pool">
    <gfe:key-interest durable="true" result-policy="KEYS">
        <bean id="key" class="java.lang.String"/>
    </gfe:key-interest>
    <gfe:regex-interest pattern=".*"/>
</gfe:client-region>
```

### 1.3.5. Configurating Disk Storage

GemFire can use disk as a secondary storage for persisting regions or/and overflow (known as data pagination or eviction to disk). SGF allows such options to be configured directly from Spring through `disk-store` element available on both `replicated-region` and `partitioned-region` as well as `client-region`. A disk store defines how that particular region can use the disk and how much space it has available. Multiple directories can be defined in a disk store such as in our example below:

```
<gfe:partitioned-region id="partition-data">
    <gfe:disk-store queue-size="50" auto-compact="true" max-oplog-size="10" synchronous-write="false" time-inter
        <gfe:disk-dir location="/mainbackup/partition" max-size="999"/>
        <gfe:disk-dir location="/backup2/partition" max-size="999"/>
    </gfe:disk-store>
</gfe:replicated-region>
```

In general, for maximum efficiency, it is recommended that each region that accesses the disk uses a disk store configuration.

For the full set of options and their meaning please refer to the Appendix A, *Spring GemFire Integration Schema* and GemFire documentation.

### 1.3.6. Data Persistence

Both partitioned and replicated regions can be made persistent. That is:

#### What is region persistence?

GemFire ensures that all the data you put into a region that is configured for persistence will be written to disk in a way that it can be recovered the next time you create the region. This allows data to be recovered after a machine or process failure or after an orderly shutdown and restart of GemFire.

With SGF, to enable persistence, simply set to true the `persistent` attribute on `replicated-region`, `partitioned-region` or `client-region`:

```
<gfe:partitioned-region id="persitent-partition" persistent="true"/>
```

#### Important

Persistence for partitioned regions is supported from GemFire 6.5 onwards - configuring this option on a previous release will trigger an initialization exception.

When persisting regions, it is recommended to configure the storage through the `disk-store` element for maximum efficiency.

### 1.3.7. Data Eviction and Overflowing

Based on various constraints, each region can have an eviction policy in place for `evicting` data from memory. Currently, in GemFire eviction applies on the least recently used entry (also known as LRU). Evicted entries are either destroyed or paged to disk (also known as *overflow*).

SGF supports all eviction policies (entry count, memory and heap usage) for both `partitioned-region` and `replicated-region` as well as `client-region`, through the nested `eviction` element. For example, to configure a partition to overflow to disk if its size is more then 512 MB, one could use the following configuration:

```
<gfe:partitioned-region id="overflow-partition">
     <gfe:eviction type="MEMORY_SIZE" threshold="512" action="OVERFLOW_TO_DISK"/>
</gfe:partitioned-region>
```

**Important**

Replicas cannot use a `local destroy` eviction since that would invalidate them. See the GemFire docs for more information.

When configuring regions for oveflow, it is recommended to configure the storage through the `disk-store` element for maximum efficiency.

For a detailed description of eviction policies, see the GemFire documentation (such as this page).

### 1.3.8. Advanced Region Configuration

SGF namespaces allow short and easy configuration of the major GemFire regions and associated entities. However, there might be corner cases where the namespaces are not enough, where a certain combination or set of attributes needs to be used. For such situations, using directly the SGF `FactoryBeans` is a possible alternative as it gives access to the full set of options at the expense of conciseness.

As a warm up, below are some common configurations, declared through raw `beans` definitions.

A basic configuration looks as follows:

```
<bean id="basic" class="org.springframework.data.gemfire.RegionFactoryBean">
  <property name="cache">
      <bean class="org.springframework.data.gemfire.CacheFactoryBean"/>
  </property>
  <property name="name" value="basic"/>
</bean>
```

Notice how the GemFire cache definition has been nested into the declaring region definition. Let's add more regions and make the cache a top level bean.

Since the region bean definition name is usually the same with that of the cache, the `name` property can be omitted (the bean name will be used automatically). Additionally by using the name the `p` namespace, the configuration can be simplified even more:

```
<?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:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/sp

    <!-- shared cache across regions -->
    <bean id="cache" class="org.springframework.data.gemfire.CacheFactoryBean"/>

    <!-- region named 'basic' -->
    <bean id="basic" class="org.springframework.data.gemfire.RegionFactoryBean" p:cache-ref="cache"/>

    <!-- region with a name different then the bean definition -->
    <bean id="root-region" class="org.springframework.data.gemfire.RegionFactoryBean" p:cache-ref="cache" p:name
</beans>
```

It is worth pointing out, that for the vast majority of cases configuring the cache loader, listener and writer through the Spring container is preferred since the same instances can be reused across multiple regions and additionally, the instances themselves can benefit from the container's rich feature set:

```
<bean id="cacheLogger" class="org.some.pkg.CacheLogger"/>
<bean id="customized-region" class="org.springframework.data.gemfire.RegionFactoryBean" p:cache-ref="cache">
  <property name="cacheListeners">
    <array>
      <ref name="cacheLogger"/>
    <bean class="org.some.other.pkg.SysoutLogger"/>
    </array>
  </property>
  <property name="cacheLoader"><bean class="org.some.pkg.CacheLoad"/></property>
  <property name="cacheWriter"><bean class="org.some.pkg.CacheWrite"/></property>
</bean>

<bean id="local-region" class="org.springframework.data.gemfire.RegionFactoryBean" p:cache-ref="cache">
  <property name="cacheListeners" ref="cacheLogger"/>
</bean>
```

For scenarios where a *CacheServer* is used and *clients* need to be configured and the namespace is not an option, SGF offers a dedicated configuration class named: `ClientRegionFactoryBean`. This allows client *interests* to be registered in both key and regex form through `Interest` and `RegexInterest` classes in the `org.springframework.data.gemfire.client` package:

```
<bean id="interested-client" class="org.springframework.data.gemfire.client.ClientRegionFactoryBean" p:cache-ref
  <property name="interests">
    <array>
      <!-- key-based interest -->
      <bean class="org.springframework.data.gemfire.client.Interest" p:key="Vlaicu" p:policy="NONE"/>
      <!-- regex-based interest -->
      <bean class="org.springframework.data.gemfire.client.RegexInterest" p:key=".*" p:policy="KEYS" p:durable="
    </array>
  </property>
</bean>
```

Users that need fine control over a region, can configure it in Spring by using the `attributes` property. To ease declarative configuration in Spring, SGF provides two `FactoryBeans` for creating `RegionAttributes` and `PartitionAttributes`, namely `RegionAttributesFactory` and `PartitionAttributesFactory`. See below an example of configuring a partitioned region through Spring XML:

```
<bean id="partitioned-region" class="org.springframework.data.gemfire.RegionFactoryBean" p:cache-ref="cache">
  <property name="attributes">
    <bean class="org.springframework.data.gemfire.RegionAttributesFactory" p:initial-capacity="1024">
      <property name="partitionAttributes">
        <bean class="org.springframework.data.gemfire.PartitionAttributesFactory" p:redundant-copies="2" p:local
```

```
      </property>
    </bean>
  </property>
</bean>
```

By using the attribute factories above, one can reduce the size of the `cache.xml` or even eliminate it all together.

## 1.4. Advantages of using Spring over GemFire `cache.xml`

With SGF, GemFire regions, pools and cache can be configured either through Spring or directly inside GemFire, native, `cache.xml` file. While both are valid approaches, it's worth pointing out that Spring's powerful DI container and AOP functionality makes it very easy to wire GemFire into an application. For example configuring a region cache loader, listener and writer through the Spring container is preferred since the same instances can be reused across multiple regions and additionally are either to configure due to the presence of the DI and eliminates the need of implementing GemFire's `Declarable` interface (see Section 2.4, “Wiring `Declarable` components” on chapter on how you can still use them yet benefit from Spring's DI container).

Whatever route one chooses to go, SGF supports both approaches allowing for easy migrate between them without forcing an upfront decision.

# Chapter 2. Working with the GemFire APIs

Once the GemFire cache and regions have been configured they can injected and used inside application objects. This chapter describes the integration with Spring's transaction management functionality and `DaoException` hierarchy. It also covers support for dependency injection of GemFire managed objects.

## 2.1. Exception translation

Using a new data access technology requires not just accommodating to a new API but also handling exceptions specific to that technology. To accommodate this case, Spring Framework provides a technology agnostic, consistent exception hierarchy that abstracts one from proprietary (and usually checked) exceptions to a set of focused runtime exceptions. As mentioned in the Spring Framework documentation, exception translation can be applied transparently to your data access objects through the use of the `@Repository` annotation and AOP by defining a PersistenceExceptionTranslationPostProcessor bean. The same exception translation functionality is enabled when using Gemfire as long as at least a `CacheFactoryBean` is declared. The `Cache` factory acts as an exception translator which is automatically detected by the Spring infrastructure and used accordingly.

## 2.2. `GemfireTemplate`

As with many other high-level abstractions provided by the Spring Framework and related projects, Spring GemFire provides a *template* that plays a central role when working with the GemFire API. The class provides several *one-liner* methods, for popular operations but also the ability to *execute* code against the native GemFire API without having to deal with exceptions for example through the `GemfireCallback`.

The template class requires a GemFire `Region` instance and once configured is thread-safe and should be reused across multiple classes:

```
<bean id="gemfireTemplate" class="org.springframework.data.gemfire.GemfireTemplate" p:region-ref="someRegion"/>
```

Once the template is configured, one can use it alongside `GemfireCallback` to work directly with the GemFire `Region`, without having to deal with checked exceptions, threading or resource management concerns:

```
template.execute(new GemfireCallback<Iterable<String>>() {
  public Iterable<String> doInGemfire(Region reg) throws GemFireCheckedException, GemFireException {
    // working against a Region of String
    Region<String, String> region = reg;

    region.put("1", "one");
    region.put("3", "three");

    return region.query("length < 5");
  }
});
```

## 2.3. Transaction Management

One of the most popular features of Spring Framework is transaction management. If you are not familiar with it, we strongly recommend looking into it as it offers a consistent programming model that works transparently across multiple APIs that can be configured either programmatically or declaratively (the most popular choice).

For GemFire, SGF provides a dedicated, per-cache, transaction manager that once declared, allows actions on the `Regions` to be grouped and executed atomically through Spring:

```
<gfe:transaction-manager id="tx-manager" cache-ref="cache"/>
```

**Note**

The example above can be simplified even more by eliminating the `cache-ref` attribute if the GemFire cache is defined under the default name `gemfire-cache`. As with the other SGF namespace elements, if the cache name is not configured, the aforementioned naming convention will used. Additionally, the transaction manager name, if not specified is `gemfire-transaction-manager`.

or if you prefer `bean` declarations:

```
<bean id="tx-manager" class="org.springframework.data.gemfire.GemfireTransactionManager" p:cache-ref="cache"/>
```

Note that currently GemFire supports optimistic transactions with *read committed* isolation. Furthermore, to guarantee this isolation, developers should avoid making *in-place* changes, that is manually modifying the values present in the cache. To prevent this from happening, the transaction manager configured the cache to use *copy on read* semantics, meaning a clone of the actual value is created, each time a read is performed. This behaviour can be disabled if needed through the `copyOnRead` property. For more information on the semantics of the underlying GemFire transaction manager, see the GemFire documentation.

## 2.4. Wiring `Declarable` components

GemFire XML configuration (usually named `cache.xml` allows *user* objects to be declared as part of the fabric configuration. Usually these objects are `CacheLoaders` or other pluggable components into GemFire. Out of the box in GemFire, each such type declared through XML must implement the `Declarable` interface which allows arbitrary parameters to be passed to the declared class through a `Properties` instance.

In this section we describe how you can configure the pluggable components defined in `cache.xml` using Spring while keeping your Cache/Region configuration defined in `cache.xml` This allows your pluggable components to focus on the application logic and not the location or creation of DataSources or other collaboration object.

However, if you are starting on a green-field project, it is recommended that you configure Cache, Region, and other pluggable components directly in Spring. This avoids inheriting from the `Declarable` interface or the base class presented in this section. See the following sidebar for more information on this approach.

**Eliminate `Declarable` components**

One can configure custom types entirely inside through Spring as mentioned in Section 1.3, “Configuring a GemFire `Region`”. That way, one does not have to implement the `Declarable` interface and gets access to all the features of the Spring IoC container (including not just dependency injection but also life-cycle and instance management).

As an example of configuring a `Declarable` component using Spring, consider the following declaration (taken from the `Declarable` javadoc):

```
<cache-loader>
```

```
   <class-name>com.company.app.DBLoader</class-name>
   <parameter name="URL">
     <string>jdbc://12.34.56.78/mydb</string>
   </parameter>
</cache-loader>
```

To simplify the task of parsing, converting the parameters and initializing the object, SGF offers a base class (`WiringDeclarableSupport`) that allows GemFire user objects to be wired through a *template* bean definition or, in case that is missing perform autowiring through the Spring container. To take advantage of this feature, the user objects need to extend `WiringDeclarableSupport` which automatically locates the declaring `BeanFactory` and performs wiring as part of the initialization process.

**Why is a base class needed?**

In the current GemFire release there is no concept of an *object factory* and the types declared are instantiated and used as is - that is there are no other ways in which third parties can take care of the object creation outside GemFire. Support for this feature is planned for the up-coming GemFire release (6.5)

### 2.4.1. Configuration using *template* definitions

When used `WiringDeclarableSupport` tries to first locate an existing bean definition and use that as wiring template. Unless specified, the component class name will be used as an implicit bean definition name. Let's see how our `DBLoader` declaration would look in that case:

```
public class DBLoader extends WiringDeclarableSupport implements CacheLoader {
  private DataSource dataSource;

  public void setDataSource(DataSource ds){
    this.dataSource = ds;
  }

  public Object load(LoaderHelper helper) { ... }
}
```

```
<cache-loader>
   <class-name>com.company.app.DBLoader</class-name>
   <!-- no parameter is passed (use the bean implicit name
   that is the class name) -->
</cache-loader>
```

```
<?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:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="dataSource" ... />

     <!-- template bean definition -->
     <bean id="com.company.app.DBLoader" abstract="true" p:dataSource-ref="dataSource"/>
</beans>
```

In the scenario above, as no parameter was specified, a bean with id/name `com.company.app.DBLoader` was searched for. The found bean definition is used as a template for wiring the instance created by GemFire. For cases where the bean name uses a different convention, one can pass in the `bean-name` parameter in the GemFire configuration:

```
<cache-loader>
   <class-name>com.company.app.DBLoader</class-name>
   <!-- pass the bean definition template name
        as parameter -->
   <parameter name="bean-name">
     <string>template-bean</string>
   </parameter>
</cache-loader>
```

```
<?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:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="dataSource" ... />

     <!-- template bean definition -->
     <bean id="template-bean" abstract="true" p:dataSource-ref="dataSource"/>

</beans>
```

### Note

The *template* bean definitions do not have to be declared in XML - any format is allowed (Groovy, annotations, etc..).

## 2.4.2. Configuration using auto-wiring and annotations

If no bean definition is found, by default, `WiringDeclarableSupport` will autowire the declaring instance. This means that unless any dependency injection *metadata* is offered by the instance, the container will find the object setters and try to automatically satisfy these dependencies. However, one can also use JDK 5 annotations to provide additional information to the auto-wiring process. We strongly recommend reading the dedicated chapter in the Spring documentation for more information on the supported annotations and enabling factors.

For example, the hypothetical `DBLoader` declaration above can be injected with a Spring-configured DataSource in the following way:

```
public class DBLoader extends WiringDeclarableSupport implements CacheLoader {
  // use annotations to 'mark' the needed dependencies
  @javax.inject.Inject
  private DataSource dataSource;

  public Object load(LoaderHelper helper) { ... }
}
```

```
<cache-loader>
   <class-name>com.company.app.DBLoader</class-name>
   <!-- no need to declare any parameters anymore
        since the class is auto-wired -->
</cache-loader>
```

```
<?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:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd">

     <!-- enable annotation processing -->
```

```
    <context:annotation-config/>

</beans>
```

By using the JSR-330 annotations, the cache loader code has been simplified since the location and creation of the DataSource has been externalized and the user code is concerned only with the loading process. The `DataSource` might be transactional, created lazily, shared between multiple objects or retrieved from JNDI - these aspects can be easily configured and changed through the Spring container without touching the `DBLoader` code.

# Chapter 3. Working with GemFire Serialization

To improve overall performance of the data fabric, GemFire supports a dedicated serialization protocol that is both faster and offers more compact results over the standard Java serialization and works transparently across various language platforms (such as Java, .NET and C++). This chapter discusses the various ways in which SGF simplifies and improves GemFire custom serialization in Java.

## 3.1. Wiring deserialized instances

It is fairly common for serialized objects to have transient data. Transient data is often dependent on the node or environment where it lives at a certain point in time, for example a DataSource. Serializing such information is useless (and potentially even dangerous) since it is local to a certain VM/machine. For such cases, SGF offers a special `Instantiator` that performs wiring for each new instance created by GemFire during deserialization.

Through such a mechanism, one can rely on the Spring container to inject (and manage) certain dependencies making it easy to split transient from persistent data and have *rich domain objects* in a transparent manner (Spring users might find this approach similar to that of `@Configurable`). The `WiringInstantiator` works just like `WiringDeclarableSupport`, trying to first locate a bean definition as a wiring template and following to autowiring otherwise. Please refer to the previous section (Section 2.4, “Wiring `Declarable` components”) for more details on wiring functionality.

To use this `Instantiator`, simply declare it as a usual bean:

```
<bean id="instantiator" class="org.springframework.data.gemfire.serialization.WiringInstantiator">
  <!-- DataSerializable type -->
  <constructor-arg>org.pkg.SomeDataSerializableClass</constructor-arg>
  <!-- type id -->
  <constructor-arg>95</constructor-arg>
</bean>
```

During the container startup, once it is being initialized, the `instantiator` will, by default, register itself with the GemFire system and perform wiring on all instances of `SomeDataSerializableClass` created by GemFire during deserialization.

## 3.2. Auto-generating custom `Instantiator`s

For data intensive applications, a large number of instances might be created on each machine as data flows in. Out of the box, GemFire uses reflection to create new types but for some scenarios, this might prove to be expensive. As always, it is good to perform profiling to quantify whether this is the case or not. For such cases, SGF allows the automatic generation of `Instatiator` classes which instantiate a new type (using the default constructor) without the use of reflection:

```
<bean id="instantiator-factory" class="org.springframework.data.gemfire.serialization.InstantiatorFactoryBean">
  <property name="customTypes">
    <map>
      <entry key="org.pkg.CustomTypeA" value="1025"/>
      <entry key="org.pkg.CustomTypeB" value="1026"/>
    </map>
  </property>
</bean>
```

The definition above, automatically generated two `Instantiators` for two classes, namely `CustomTypeA` and `CustomTypeB` and registers them with GemFire, under user id `1025` and `1026`. The two instantiators avoid the

use of reflection and create the instances directly through Java code.

# Chapter 4. Sample Applications

The Spring GemFire project includes one sample application. Named "Hello World", the sample demonstrates how to configure and use GemFire inside a Spring application. At runtime, the sample offers a *shell* to the user allowing him to run various commands against the grid. It provides an excellent starting point for users unfamiliar with the essential components or the Spring and GemFire concepts.

The sample is bundled with the distribution and is Maven-based. One can easily import them into any Maven-aware IDE (such as SpringSource Tool Suite) or run them from the command-line.

## 4.1. Hello World

The Hello World sample demonstrates the core functionality of the Spring GemFire project. It bootstraps GemFire, configures it, executes arbitrary commands against it and shuts it down when the application exits. Multiple instances can be started at the same time as they will work with each other sharing data without any user intervention.

### 4.1.1. Starting and stopping the sample

Hello World is designed as a stand-alone java application. It features a `Main` class which can be started either from your IDE of choice (in Eclipse/STS through `Run As/Java Application`) or from the command line through Maven using `mvn exec:java`. One can also use `java` directly on the resulting artifact if the classpath is properly set.

To stop the sample, simply type `exit` at the command line or press `Ctrl+C` to stop the VM and shutdown the Spring container.

### 4.1.2. Using the sample

Once started, the sample will create a shared data grid and allow the user to issue commands against it. The output will likely look as follows:

```
INFO: Created GemFire Cache [Spring GemFire World] v. X.Y.Z
INFO: Created new cache region [myWorld]
INFO: Member xxxxxx:50694/51611 connecting to region [myWorld]
Hello World!
Want to interact with the world ? ...
Supported commands are:

get <key> - retrieves an entry (by key) from the grid
put <key> <value> - puts a new entry into the grid
remove <key> - removes an entry (by key) from the grid
...
```

For example to add new items to the grid one can use:

```
-> put 1 unu
INFO: Added [1=unu] to the cache
null
-> put 1 one
INFO: Updated [1] from [unu] to [one]
unu
-> size
1
-> put 2 two
INFO: Added [2=two] to the cache
null
```

```
-> size
2
```

Multiple instances can be created at the same time. Once started, the new VMs automatically see the existing region and its information:

```
INFO: Connected to Distributed System ['Spring GemFire World'=xxxx:56218/49320@yyyyy]
Hello World!
...

-> size
2
-> map
[2=two] [1=one]
-> query length = 3
[one, two]
```

Experiment with the example, start (and stop) as many instances as you want, run various commands in one instance and see how the others react. To preserve data, at least one instance needs to be alive all times - if all instances are shutdown, the grid data is completely destroyed (in this example - to preserve data between runs, see the GemFire documentations).

### 4.1.3. Hello World Sample Explained

Hello World uses both Spring XML and annotations for its configuration. The initial boostrapping configuration is `app-context.xml` which includes the cache configuration, defined under `cache-context.xml` file and performs classpath scanning for Spring components. The cache configuration defines the GemFire cache, region and for illustrative purposes a simple cache listener that acts as a logger.

The main *beans* are `HelloWorld` and `CommandProcessor` which rely on the `GemfireTemplate` to interact with the distributed fabric. Both classes use annotations to define their dependency and life-cycle callbacks.

# Part III. Other Resources

In addition to this reference documentation, there are a number of other resources that may help you learn how to use GemFire and Spring framework. These additional, third-party resources are enumerated in this section.

# Chapter 5. Useful Links

- *Spring GemFire Integration Home Page* - here
- *SpringSource blog* - here
- *GemFire Community* - here

# Part IV. Appendices

# Appendix A. Spring GemFire Integration Schema

Spring GemFire Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.springframework.org/schema/gemfire"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:beans="http://www.springframework.org/schema/beans"
            xmlns:tool="http://www.springframework.org/schema/tool"
            targetNamespace="http://www.springframework.org/schema/gemfire"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            version="1.0">

    <xsd:import namespace="http://www.springframework.org/schema/beans"/>
    <xsd:import namespace="http://www.springframework.org/schema/tool"/>

    <xsd:annotation>
        <xsd:documentation><![CDATA[
            Namespace support for the Spring GemFire project.
        ]]></xsd:documentation>
    </xsd:annotation>

    <xsd:element name="cache">
                <xsd:annotation>
                        <xsd:documentation source="org.springframework.data.gemfire.CacheFactoryBean"><![CDATA[
Defines a GemFire Cache instance used for creating or retrieving 'regions'.
                        ]]></xsd:documentation>
                        <xsd:appinfo>
                                <tool:annotation>
                                        <tool:exports type="com.gemstone.gemfire.cache.Cache" />
                                </tool:annotation>
                        </xsd:appinfo>
                </xsd:annotation>
                <xsd:complexType>
                        <xsd:attribute name="id" type="xsd:ID" use="optional">
                                <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The name of the cache definition (by default "gemfire-cache").]]></xsd:documentation>
                                </xsd:annotation>
                        </xsd:attribute>
                        <xsd:attribute name="cache-xml-location" type="xsd:string" use="optional">
                                <xsd:annotation>
                                        <xsd:documentation source="org.springframework.core.io.Resource"><![CDAT
The location of the GemFire cache xml file, as a Spring resource location: a URL, a "classpath:" pseudo URL,
or a relative file path.
                                                ]]></xsd:documentation>
                                </xsd:annotation>
                        </xsd:attribute>
                        <xsd:attribute name="properties-ref" type="xsd:string" use="optional">
                                <xsd:annotation>
                                        <xsd:documentation source="java.util.Properties"><![CDATA[
The bean name of a Java Properties object that will be used for property substitution. For loading properties
consider using a dedicated utility such as the <util:*/> namespace and its 'properties' element.
                                                ]]></xsd:documentation>
                                </xsd:annotation>
                        </xsd:attribute>
                </xsd:complexType>
        </xsd:element>

        <xsd:element name="transaction-manager">
                <xsd:annotation>
                        <xsd:documentation source="org.springframework.data.gemfire.TransactionManager"><![CDATA
Defines a GemFire Transaction Manager instance for a single GemFire cache.
                        ]]></xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                        <xsd:attribute name="id" type="xsd:ID" use="optional">
                                <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The name of the transaction manager definition (by default "gemfire-transaction-manager").]]></xsd:documentation
```

```
                                </xsd:annotation>
                        </xsd:attribute>
                        <xsd:attribute name="cache-ref" type="xsd:string" default="gemfire-cache" use="optional"
                                <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The name of the bean defining the GemFire cache (by default 'gemfire-cache').
                                        ]]></xsd:documentation>
                                </xsd:annotation>
                        </xsd:attribute>
                        <xsd:attribute name="copy-on-read" type="xsd:boolean" default="true" use="optional">
                                <xsd:annotation>
                                        <xsd:documentation><![CDATA[
Indicates whether the cache returns direct references or copies of the objects (default) it manages.
While copies imply additional work for every fetch operation, direct references can cause dirty reads
across concurrent threads in the same VM, whether or not transactions are used.
                                        ]]></xsd:documentation>
                                </xsd:annotation>
                        </xsd:attribute>

                </xsd:complexType>
        </xsd:element>

        <!-- nested bean definition -->
        <xsd:complexType name="beanDeclarationType">
                <xsd:sequence>
                        <xsd:any namespace="##other" minOccurs="0" maxOccurs="1" processContents="skip">
                            <xsd:annotation>
                                 <xsd:documentation><![CDATA[
Inner bean definition. The nested declaration serves as an alternative to bean references (using
both in the same definition) is illegal.
                                        ]]></xsd:documentation>
                                </xsd:annotation>
                 </xsd:any>
                </xsd:sequence>
                <xsd:attribute name="ref" type="xsd:string" use="optional">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
The name of the bean referred by this declaration. If no reference exists, use an inner bean declaration.
                                ]]></xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
        </xsd:complexType>

        <xsd:complexType name="basicRegionType">
                <xsd:annotation>
                        <xsd:appinfo>
                                <tool:annotation>
                                        <tool:exports type="com.gemstone.gemfire.cache.Region" />
                                </tool:annotation>
                        </xsd:appinfo>
                </xsd:annotation>
                <xsd:attribute name="id" type="xsd:string" use="required">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
The id of the region bean definition.
                                ]]></xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="name" type="xsd:string" use="optional">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
The name of the region definition. If no specified, it will have the value of the id attribute (that is, the bea
                                ]]></xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="cache-ref" type="xsd:string" default="gemfire-cache" use="optional">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
The name of the bean defining the GemFire cache (by default 'gemfire-cache').
                                ]]></xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
        </xsd:complexType>


        <xsd:complexType name="readOnlyRegionType" abstract="true">
                <xsd:complexContent>
                        <xsd:extension base="basicRegionType">
```

```
                                <xsd:sequence>
                                        <xsd:element name="cache-listener" minOccurs="0" maxOccurs="1">
                                                <xsd:annotation>
                                                        <xsd:documentation source="com.gemstone.gemfire.cache.Ca
A cache listener definition for this region. A cache listener handles region or entry related events (that occur
various operations on the region). Multiple listeners can be declared in a nested manner.

Note: Avoid the risk of deadlock. Since the listener is invoked while holding a lock on the entry generating the
it is easy to generate a deadlock by interacting with the region. For this reason, it is highly recommended to u
other thread for accessing the region and not waiting for it to complete its task.
                                                        ]]></xsd:documentation>
                                                        <xsd:appinfo>
                                                                <tool:annotation>
                                                                        <tool:exports type="com.gemstone.gemfire
                                                                </tool:annotation>
                                                        </xsd:appinfo>
                                                </xsd:annotation>
                                                <xsd:complexType>
                                                        <xsd:sequence>
                                                                <xsd:any namespace="##other" minOccurs="0" maxOc
                                                                    <xsd:annotation>
                                                                         <xsd:documentation><![CDATA[
Inner bean definition of the cache listener.
                                                                                ]]></xsd:documentation>
                                                                    </xsd:annotation>
                                                                 </xsd:any>
                                                        </xsd:sequence>
                                                        <xsd:attribute name="ref" type="xsd:string" use="optiona
                                                                <xsd:annotation>
                                                                        <xsd:documentation><![CDATA[
The name of the cache listener bean referred by this declaration. Used as a convenience method. If no reference
use inner bean declarations.
                                                                        ]]></xsd:documentation>
                                                                </xsd:annotation>
                                                        </xsd:attribute>
                                                </xsd:complexType>
                                        </xsd:element>
                                        <xsd:element name="disk-store" type="diskStoreType" minOccurs="0" maxOcc
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
Disk storage configuration for the defined region.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:element>
                                </xsd:sequence>
                                <xsd:attribute name="persistent" type="xsd:boolean" default="false">
                                        <xsd:annotation>
                                                <xsd:documentation><![CDATA[
Indicates whether the defined region is persistent or not. GemFire ensures that all the data you put into a regi
 is configured for persistence will be written to disk in a way that it can be recovered the next time you creat
region. This allows data to be recovered after a machine or process failure or after an orderly shutdown and res
of GemFire.

Default is false, meaning the regions are not persisted.

Note: Persistence for partitioned regions is supported only from GemFire 6.5 onwards.
                                                ]]></xsd:documentation>
                                        </xsd:annotation>
                                </xsd:attribute>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>

        <xsd:complexType name="regionType">
                <xsd:complexContent>
                        <xsd:extension base="readOnlyRegionType">
                                <xsd:sequence minOccurs="0" maxOccurs="1">
                                        <xsd:element name="cache-loader" minOccurs="0"  maxOccurs="1" type="bean
                                                <xsd:annotation>
                                                        <xsd:documentation source="com.gemstone.gemfire.cache.Ca
The cache loader definition for this region. A cache loader allows data to be placed into a region.
                                                        ]]></xsd:documentation>
                                                        <xsd:appinfo>
                                                                <tool:annotation>
                                                                        <tool:exports type="com.gemstone.gemfire
                                                                </tool:annotation>
                                                        </xsd:appinfo>
                                                </xsd:annotation>
```

```
                                        </xsd:element>
                                        <xsd:element name="cache-writer" minOccurs="0" maxOccurs="1" type="beanD
                                                <xsd:annotation>
                                                        <xsd:documentation source="com.gemstone.gemfire.cache.Ca
The cache writer definition for this region. A cache writer acts as a dedicated synchronous listener that is not
before a region or an entry is modified. A typical example would be a writer that updates the database.

Note: Only one CacheWriter is invoked. GemFire will always prefer the local one (if it exists) otherwise it will
arbitrarily pick one.
                                                        ]]></xsd:documentation>
                                                        <xsd:appinfo>
                                                                <tool:annotation>
                                                                        <tool:exports type="com.gemstone.gemfire
                                                                </tool:annotation>
                                                        </xsd:appinfo>
                                                </xsd:annotation>
                                        </xsd:element>
                                </xsd:sequence>
                        </xsd:extension>
                </xsd:complexContent>
        </xsd:complexType>

        <xsd:element name="lookup-region" type="basicRegionType">
                <xsd:annotation>
                        <xsd:documentation><![CDATA[
Looks up an existing, working, GemFire region. Typically regions are defined through GemFire own configuration,
cache.xml. If the region does not exist, an exception will be thrown.

For defining regions, consider the region elements.
                        ]]></xsd:documentation>
                </xsd:annotation>
        </xsd:element>

        <xsd:element name="replicated-region">
                <xsd:annotation>
                        <xsd:documentation source="org.springframework.data.gemfire.RegionFactoryBean"><![CDATA[
Defines a GemFire replicated region instance. Each replicated region contains a complete copy of the data.
As well as high availability, replication provides excellent performance as each region contains a complete,
up to date copy of the data.
                        ]]></xsd:documentation>
                        <xsd:appinfo>
                                <tool:annotation>
                                        <tool:exports type="com.gemstone.gemfire.cache.Region" />
                                </tool:annotation>
                        </xsd:appinfo>
                </xsd:annotation>
                <xsd:complexType>
                        <xsd:complexContent>
                                <xsd:extension base="regionType">
                                        <xsd:sequence minOccurs="1" maxOccurs="1">
                                                <xsd:element name="eviction" minOccurs="0" maxOccurs="1">
                                                        <xsd:annotation>
                                                                <xsd:documentation><![CDATA[
Eviction policy for the partitioned region.
                                                                ]]></xsd:documentation>
                                                        </xsd:annotation>
                                                        <xsd:complexType>
                                                                <xsd:complexContent>
                                                                        <xsd:extension base="evictionType">
                                                                                <xsd:attribute name="action" typ
                                                                                        <xsd:annotation>
                                                                                                <xsd:documentati
The action to take when performing eviction.
                                                                                                ]]></xsd:documen
                                                                                        </xsd:annotation>
                                                                                </xsd:attribute>
                                                                        </xsd:extension>
                                                                </xsd:complexContent>
                                                        </xsd:complexType>
                                                </xsd:element>
                                        </xsd:sequence>
                                </xsd:extension>
                        </xsd:complexContent>
                </xsd:complexType>
        </xsd:element>

        <xsd:element name="partitioned-region">
                <xsd:annotation>
```

```
                        <xsd:documentation source="org.springframework.data.gemfire.RegionFactoryBean"><![CDATA[
Defines a GemFire partitioned region instance. Through partitioning, the data is split across regions.
Partitioning is useful when the amount of data to store is too large for one member to hold and work
with as if it were a single entity. One can configure the partitioned region to store redundant copies
in different members, for high availability in case of an application failure.
                        ]]></xsd:documentation>
                        <xsd:appinfo>
                                <tool:annotation>
                                        <tool:exports type="com.gemstone.gemfire.cache.Region" />
                                </tool:annotation>
                        </xsd:appinfo>
                </xsd:annotation>
                <xsd:complexType>
                        <xsd:complexContent>
                                <xsd:extension base="regionType">
                                        <xsd:sequence>
                                                <xsd:element name="partition-resolver" minOccurs="0" maxOccurs="
                                                        <xsd:annotation>
                                                                <xsd:documentation source="com.gemstone.gemfire.
The partition resolver definition for this region, allowing for custom partitioning. GemFire uses the resolver t
colocate data based on custom criterias (such as colocating trades by month and year).
                                                                ]]></xsd:documentation>
                                                                <xsd:appinfo>
                                                                        <tool:annotation>
                                                                                <tool:exports type="com.gemstone
                                                                        </tool:annotation>
                                                                </xsd:appinfo>
                                                        </xsd:annotation>
                                                </xsd:element>
                                                <xsd:element name="eviction" minOccurs="0" maxOccurs="1">
                                                        <xsd:annotation>
                                                                <xsd:documentation><![CDATA[
Eviction policy for the partitioned region.
                                                                ]]></xsd:documentation>
                                                        </xsd:annotation>
                                                        <xsd:complexType>
                                                                <xsd:complexContent>
                                                                        <xsd:extension base="evictionType">
                                                                                <xsd:attribute name="action" typ
                                                                                        <xsd:annotation>
                                                                                                <xsd:documentati
The action to take when performing eviction.
                                                                                                ]]></xsd:documer
                                                                                        </xsd:annotation>
                                                                                </xsd:attribute>
                                                                        </xsd:extension>
                                                                </xsd:complexContent>
                                                        </xsd:complexType>
                                                </xsd:element>
                                        </xsd:sequence>
                                        <xsd:attribute name="copies" default="0" use="optional">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The number of copies for each partition for high-availability. By default, no copies are created meaning there i
redundancy. Each copy provides extra backup at the expense of extra storages.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                                <xsd:simpleType>
                                                        <xsd:restriction base="xsd:byte">
                                                                <xsd:minInclusive value="0"/>
                                                                <xsd:maxInclusive value="3"/>
                                                        </xsd:restriction>
                                                </xsd:simpleType>
                                        </xsd:attribute>
                                        <xsd:attribute name="colocated-with" type="xsd:string" use="optional">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The name of the partitioned region with which this newly created partitioned region is colocated.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:attribute>
                                        <xsd:attribute name="local-max-memory" type="xsd:positiveInteger" use="o
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The maximum amount of memory, in megabytes, to be used by the region in this process. If not set, a default of 9
of available heap is used.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
```

```
                                        </xsd:attribute>
                                        <xsd:attribute name="total-max-memory" type="xsd:positiveInteger" use="o
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The maximum amount of memory, in megabytes, to be used by the region in all process.

Note: This setting must be the same in all processes using the region.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:attribute>
                                        <xsd:attribute name="total-buckets" type="xsd:positiveInteger" use="opti
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The total number of hash buckets to be used by the region in all processes.

A bucket is the smallest unit of data management in a partitioned region. Entries are stored in buckets and buck
 move from one VM to another. Buckets may also have copies, depending on redundancy to provide high availability
  face of VM failure.
The number of buckets should be prime and as a rough guide at the least four times the number of partition VMs.
, there is significant overhead to managing a bucket, particularly for higher values of redundancy.

Note: This setting must be the same in all processes using the region.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:attribute>
                                        <xsd:attribute name="recovery-delay" type="xsd:long" default="-1" use="o
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The delay in milliseconds that existing members will wait before satisfying redundancy after another member cras
-1 (the default) indicates that redundancy will not be recovered after a failure.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:attribute>
                                        <xsd:attribute name="startup-recovery-delay" type="xsd:long" default="-1
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The delay in milliseconds that new members will wait before satisfying redundancy. -1 indicates that adding new
 will not trigger redundancy recovery. The default is to recover redundancy immediately when a new member is add
                                                         ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:attribute>
                                </xsd:extension>
                        </xsd:complexContent>
                </xsd:complexType>
        </xsd:element>

        <xsd:complexType name="evictionType">
                <xsd:sequence minOccurs="0" maxOccurs="1">
                        <xsd:element name="object-sizer" type="beanDeclarationType">
                                <xsd:annotation>
                                        <xsd:documentation><![CDATA[
Entity computing sizes for objects stored into the grid.
                                        ]]></xsd:documentation>
                                        <xsd:appinfo>
                                                <tool:annotation>
                                                        <tool:exports type="com.gemstone.gemfire.cache.util.Obje
                                                </tool:annotation>
                                        </xsd:appinfo>
                                </xsd:annotation>
                        </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="type" default="ENTRY_COUNT">
                        <xsd:simpleType>
                                <xsd:restriction base="xsd:string">
                                        <xsd:enumeration value="ENTRY_COUNT">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
Considers the number of entries in the region before performing an eviction.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:enumeration>
                                        <xsd:enumeration value="MEMORY_SIZE">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
Considers the amount of memory consumed by the region before performing an eviction.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:enumeration>
```

```
                                        <xsd:enumeration value="HEAP_PERCENTAGE">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
Considers the amount of heap used (through the GemFire resource manager) before performing an eviction.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:enumeration>
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="threshold" type="xsd:long" use="required">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
The threshold (or limit) against which the eviction algorithm runs. Once the threashold is reached, eviction is
performed.
                                ]]></xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
        </xsd:complexType>

        <xsd:simpleType name="evictionActionType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="LOCAL_DESTROY">
                                <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The LRU (least-recently-used) region entries is locally destroyed.

Note: this option is not compatible with replicated regions (as it render the replica region incomplete).
                                        ]]></xsd:documentation>
                                </xsd:annotation>
                        </xsd:enumeration>
                        <xsd:enumeration value="OVERFLOW_TO_DISK">
                                <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The LRU (least-recently-used) region entry values are written to disk and nulled-out in the member to
reclaim memory.
                                        ]]></xsd:documentation>
                                </xsd:annotation>
                        </xsd:enumeration>
                </xsd:restriction>
        </xsd:simpleType>

        <xsd:complexType name="diskStoreType">
                <xsd:sequence>
                        <xsd:element name="disk-dir" minOccurs="0" maxOccurs="unbounded">
                                <xsd:complexType>
                                        <xsd:attribute name="location" type="xsd:string" use="required">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
Directory on the file system for storing data.

Note: the directory must already exist.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:attribute>
                                        <xsd:attribute name="max-size" type="xsd:int" default="10240">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The maximum size (in megabytes) of data stored in each directory. Default is 10240 MB (10 gigabytes).
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:attribute>
                                </xsd:complexType>
                        </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="synchronous-write" type="xsd:boolean" default="false">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
Indicates whether the writing to the disk si synchronous or not. Default is false, meaning asynchronous writing.
                                ]]>
                                </xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="auto-compact" type="xsd:boolean" default="true">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
Indicates whether or not the operation logs are automatically compacted or not. Default is true.
                                ]]>
```

```
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <!--
                <xsd:attribute name="compaction-threshold" default="50">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Sets the threshold at which an oplog will become compactable. Until it reaches this threshold the oplog will not
compacted. The threshold is a percentage in the range 0..100. When the amount of garbage in an oplog exceeds thi
percentage then when a compaction is done this garbage will be cleaned up freeing up disk space. Garbage is crea
by entry destroys, entry updates, and region destroys.
                        ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:short">
                            <xsd:minExclusive value="0"/>
                            <xsd:maxExclusive value="100"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
                 -->
                <xsd:attribute name="max-oplog-size" type="xsd:long" default="1024">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Sets the maximum size in megabytes a single oplog (operation log) is allowed to be. When an oplog is created thi
amount of file space will be immediately reserved.
                        ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="time-interval" type="xsd:long" default="1">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Sets the number of milliseconds that can elapse before unwritten data is written to disk.
It is considered only for asynchronous writing.
                        ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="queue-size" type="xsd:long" default="0">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The maximum number of operations that can be asynchronously queued. Once this many pending async operations have
queued async ops will begin blocking until some of the queued ops have been flushed to disk.
Considered only for asynchronous writing.
                        ]]>
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
        </xsd:complexType>

        <xsd:element name="client-region">
                <xsd:annotation>
                    <xsd:documentation source="org.springframework.data.gemfire.client.ClientRegionFactoryBe
Defines a GemFire client region instance. A client region is connected to a (long-lived) farm of GemFire servers
which it receives its data. The client can hold some data locally or forward all requests to the server.
                    ]]></xsd:documentation>
                    <xsd:appinfo>
                        <tool:annotation>
                            <tool:exports type="com.gemstone.gemfire.cache.Region" />
                        </tool:annotation>
                    </xsd:appinfo>
                </xsd:annotation>
                <xsd:complexType>
                    <xsd:complexContent>
                        <xsd:extension base="readOnlyRegionType">
                            <xsd:sequence>
                                <xsd:choice minOccurs="0" maxOccurs="unbounded">
                                    <xsd:element name="key-interest">
                                        <xsd:annotation>
                                            <xsd:documentation><![CDATA[
Key based interest. If the key is a List, then all the keys in the List will be registered. The key can also be
special token 'ALL_KEYS', which will register interest in all keys in the region. In effect, this will cause an
to any key in this region in the CacheServer to be pushed to the client.
                                            ]]></xsd:documentation>
                                        </xsd:annotation>
                                        <xsd:complexType>
```

```
                                                                <xsd:complexContent>
                                                                        <xsd:extension base="interestTyp
                                                                                <xsd:sequence minOccurs=
                                                                                        <xsd:any namespa
                                                                                             <xsd:annotat
                                                                                                  <xsd:do
Inner bean definition of the client key interest.

                                                                                                 </xsd:an
                                                                                 </xsd:any>
                                                                                </xsd:sequence>
                                                                                <xsd:attribute name="key
                                                                                        <xsd:annotation>
                                                                                            <xsd:doc
The name of the client key interest bean referred by this declaration. Used as a convenience method. If no refer
use the inner bean declaration.
                                                                                                ]]></xsd
                                                                                        </xsd:annotation
                                                                                </xsd:attribute>
                                                                        </xsd:extension>
                                                                </xsd:complexContent>
                                                        </xsd:complexType>
                                                </xsd:element>
                                                <xsd:element name="regex-interest">
                                                        <xsd:annotation>
                                                                <xsd:documentation><![CDATA[
Regular expression based interest. If the pattern is '.*' then all keys of any type will be pushed to the client
                                                                ]]></xsd:documentation>
                                                        </xsd:annotation>
                                                        <xsd:complexType>
                                                                <xsd:complexContent>
                                                                        <xsd:extension base="interestTyp
                                                                                <xsd:attribute name="pat
                                                                        </xsd:extension>
                                                                </xsd:complexContent>
                                                        </xsd:complexType>
                                                </xsd:element>
                                        </xsd:choice>
                                        <xsd:element name="eviction" minOccurs="0" maxOccurs="1">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
Eviction policy for the partitioned region.
                                                        ]]></xsd:documentation>
                                                </xsd:annotation>
                                                <xsd:complexType>
                                                        <xsd:complexContent>
                                                                <xsd:extension base="evictionType">
                                                                        <xsd:attribute name="action" typ
                                                                                <xsd:annotation>
                                                                                        <xsd:documentati
The action to take when performing eviction.
                                                                                        ]]></xsd:documen
                                                                                </xsd:annotation>
                                                                        </xsd:attribute>
                                                                </xsd:extension>
                                                        </xsd:complexContent>
                                                </xsd:complexType>
                                        </xsd:element>
                                </xsd:sequence>
                                <xsd:attribute name="data-policy" use="optional" default="NORMAL">
                                        <xsd:annotation>
                                                <xsd:documentation><![CDATA[
The data policy for this client. Can be either 'EMPTY' or 'NORMAL' (the default). In case persistence or overflo
configured for this region, this parameter will be ignored.

EMPTY - causes data to never be stored in local memory. The region will always appear empty. It can be used to f
 footprint producers that only want to distribute their data to others and for zero footprint consumers that onl
 to see events.
NORMAL - causes data that this region is interested in to be stored in local memory. It allows the contents in t
cache to differ from other caches.
                                                 ]]></xsd:documentation>
                                        </xsd:annotation>
                                        <xsd:simpleType>
                                                <xsd:restriction base="xsd:string">
                                                        <xsd:enumeration value="EMPTY"/>
                                                        <xsd:enumeration value="NORMAL"/>
                                                </xsd:restriction>
                                        </xsd:simpleType>
```

```
                                        </xsd:attribute>
                                        <xsd:attribute name="pool-name" use="required" type="xsd:string">
                                                <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
The name of the pool used by this client.
                                                         ]]></xsd:documentation>
                                                </xsd:annotation>
                                        </xsd:attribute>
                                </xsd:extension>
                        </xsd:complexContent>
                </xsd:complexType>
        </xsd:element>

        <xsd:complexType name="connectionType">
                <xsd:attribute name="host" type="xsd:string">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
The host name or ip address of the connection.
                                ]]></xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="port">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
The port number of the connection (between 1 and 65535 inclusive).
                                ]]></xsd:documentation>
                        </xsd:annotation>
                        <xsd:simpleType>
                                <xsd:restriction base="xsd:positiveInteger">
                                        <xsd:minInclusive value="1"/>
                                        <xsd:maxInclusive value="65535"/>
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:attribute>
        </xsd:complexType>

        <xsd:complexType name="interestType" abstract="true">
                <xsd:attribute name="durable"  type="xsd:boolean" default="false" use="optional">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
Indicates whether or not the registered interest is durable or not. Default is false.
                                ]]></xsd:documentation>
                        </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="result-policy" default="KEYS_VALUES" use="optional">
                        <xsd:annotation>
                                <xsd:documentation><![CDATA[
The result policy for this interest. Can be one of 'KEYS' or 'KEYS_VALUES' (the default) or 'NONE'.

KEYS - Initializes the local cache with the keys satisfying the request.
KEYS-VALUES - initializes the local cache with the keys and current values satisfying the request.
NONE -  Does not initialize the local cache.
                                        ]]></xsd:documentation>
                        </xsd:annotation>
                        <xsd:simpleType>
                                <xsd:restriction base="xsd:string">
                                        <xsd:enumeration value="KEYS"/>
                                        <xsd:enumeration value="KEYS_VALUES"/>
                                        <xsd:enumeration value="NONE"/>
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:attribute>
        </xsd:complexType>

        <xsd:element name="pool">
                <xsd:annotation>
                        <xsd:documentation source="org.springframework.data.gemfire.client.PoolFactoryBean"><![C
Defines a pool for connections from a client to a set of GemFire Cache Servers.

Note that in order to instantiate a pool, a GemFire cache needs to be already started.
                        ]]></xsd:documentation>
                        <xsd:appinfo>
                                <tool:annotation>
                                        <tool:exports type="com.gemstone.gemfire.cache.client.Pool" />
                                </tool:annotation>
                        </xsd:appinfo>
                </xsd:annotation>
                <xsd:complexType>
```

```
                        <xsd:choice minOccurs="1" maxOccurs="1">
                                <xsd:element name="locator" type="connectionType" minOccurs="1" maxOccurs="unbou
                                <xsd:element name="server" type="connectionType" minOccurs="1" maxOccurs="unbour
                        </xsd:choice>
                        <xsd:attribute name="id" type="xsd:ID" use="optional">
                                <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The name of the pool definition (by default "gemfire-pool").]]></xsd:documentation>
                                </xsd:annotation>
                        </xsd:attribute>
                        <xsd:attribute name="free-connection-timeout" use="optional" type="xsd:int"/>
                        <xsd:attribute name="idle-timeout" use="optional" type="xsd:long"/>
                        <xsd:attribute name="load-conditioning-interval" use="optional" type="xsd:int"/>
                        <xsd:attribute name="max-connections" use="optional" type="xsd:int"/>
                        <xsd:attribute name="min-connections" use="optional" type="xsd:int"/>
                        <xsd:attribute name="ping-interval" use="optional" type="xsd:long"/>
                        <xsd:attribute name="read-timeout" use="optional" type="xsd:int"/>
                        <xsd:attribute name="retry-attempts" use="optional" type="xsd:int"/>
                        <xsd:attribute name="server-group" use="optional" type="xsd:string"/>
                        <xsd:attribute name="socket-buffer-size" use="optional" type="xsd:int"/>
                        <xsd:attribute name="statistic-interval" use="optional" type="xsd:int"/>
                        <xsd:attribute name="subscription-ack-interval" use="optional" type="xsd:int"/>
                        <xsd:attribute name="subscription-enabled" use="optional" type="xsd:boolean"/>
                        <xsd:attribute name="subscription-message-tracking-timeout" use="optional" type="xsd:int
                        <xsd:attribute name="subscription-redundancy" use="optional" type="xsd:int"/>
                        <xsd:attribute name="thread-local-connections" use="optional" type="xsd:boolean"/>
                </xsd:complexType>
        </xsd:element>
</xsd:schema>
```