Spring Data Gemfire Reference Guide

Costin Leau
Oliver Gierke
David Turanski

Spring Data Gemfire Reference Guide

by Costin Leau, Oliver Gierke, and David Turanski

1.2.0.RC1

Note

As of the 1.2.0 release, this project, formerly known as Spring GemFire, has been renamed to Spring Data GemFire to reflect that it is now a component of the Spring Data project.

Table of Contents

Preface

Spring Data GemFire focuses on integrating Spring Framework's powerful, non-invasive programming model and concepts with vFabric GemFire, simplifying configuration, development and providing high-level abstractions. This document assumes the reader already has a basic familiarity with the Spring Framework and vFabric GemFire concepts and APIs.

While every effort has been made to ensure that this documentation is comprehensive and there are no errors, 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 Data GemFire team by raising an issue. Thank you.

Part I. Introduction

1. Introduction

This reference guide for the Spring Data GemFire project explains how to use Spring framework to configure and develop applications with vFabric GemFire. It presents the basic concepts, semantics and provides numerous examples to help you get started.

Note

Spring Data GemFire started as a top level Spring project called Spring GemFire (SGF) and has since moved under the Spring Data umbrella project and has been renamed accordingly.

2. Requirements

Spring Data GemFire requires JDK level 6.0 and above, Spring Framework 3 and vFabric GemFire 6.6 and above.

3. New Features

Note

As of the 1.2.0 release, this project, formerly known as Spring GemFire, has been renamed to Spring Data GemFire to reflect that it is now a component of the Spring Data project.

3.1 New in the 1.2.0 Release

- Full support for GemFire configuration via the *gfe* namespace. Now GemFire components may be configured completely without requiring a native cache.xml file.
- WAN Gateway support for both GemFire 6.6.x. See Section 5.8, “Configuring WAN Gateways”
- Spring Data Repository support with a dedicated namespace, *gfe-data*. See Chapter 9, *GemFire Repositories*
- Namespace support for registering GemFire functions. See Section 5.7, “Configuring GemFire's Function Service”
- A top level `<disk-store>` element has been added to the *gfe* namespace to allow sharing of persist stores among regions, and other components that support persistent backup. See Section 5.6, “Configuring a Disk Store”

  

  Caution

  The `<*-region>` elements no longer allow a nested `<disk-store>`
- GemFire subregions are supported via nested `<*-region>` elements
- A `<local-region>` element has been added to configure a local region

Part II. Reference Guide

4. Document structure

The following chapters explain the core functionality offered by Spring Data GemFire.

Chapter 5, *Bootstrapping GemFire through the Spring Container* describes the configuration support provided for bootstrapping, initializing, configuring, and accessing GemFire caches, cache servers, regions, and related distributed system components

Chapter 6, *Working with the GemFire APIs* explains the integration between the GemFire APIs and the various data access features available in Spring, such as transaction management and exception translation.

Chapter 7, *Working with GemFire Serialization* describes the enhancements for GemFire (de)serialization and management of associated objects.

Chapter 8, *POJO mapping* describes persistence mapping for POJOs stored in GemFire using Spring Data

Chapter 9, *GemFire Repositories* describes how to create and use GemFire Repositories using Spring Data

Chapter 10, *Sample Applications* describes the samples provided with the distribution to illustrate the various features available in Spring GemFire.

5. Bootstrapping GemFire through the Spring Container

Spring Data GemFire provides full configuration and initialization of the GemFire data grid through Spring's IoC container and provides several classes that simplify the configuration of GemFire components including caches, regions, WAN gateways, persistence backup, and other distributed system components to support a variety of scenarios with minimal effort.

Note

This section assumes basic familiarity with GemFire. For more information see the product documentation.

5.1 Advantages of using Spring over GemFire `cache.xml`

As of release 1.2.0, Spring Data Gemfire's XML namespace supports full configuration of the data grid. In fact, the Spring namespace is considered the preferred way to configure GemFire. GemFire will continue to support `cache.xml` for legacy reasons, but you can now do everything in Spring XML and take advantage of the many wonderful things Spring has to offer such as modular XML configuration, property placeholders, SpEL, and environment profiles. Behind the namespace, Spring Data GemFire makes extensive use of Spring's `FactoryBean` pattern to simplify the creation and initialization of GemFire components.

For example, GemFire provides several callback interfaces such as `CacheListener`, `CacheWriter`, `CacheLoader` to allow developers to add custom event handlers. Using the Spring IoC container, these may configured as normal Spring beans and injected into GemFire components. This is a significant improvement over cache.xml which provides relatively limited configuration options and requires callbacks to implement GemFire's `Declarable` interface (see Section 6.6, “Wiring `Declarable` components” to see how you can still use Declarables within Spring's DI container).

In addition, IDEs such as the Spring Tool Suite (STS) provide excellent support for Spring XML namespaces, such as code completion, pop-up annotations, and real time validation, making them easy to use.

5.2 Using the Core Spring Data GemFire Namespace

To simplify configuration, Spring Data Gemfire provides a dedicated XML namespace for configuring core GemFire components. It is also possible to configure the beans directly through Springs standard <bean> definition. However, as of Spring Data GemFire 1.2.0, all bean properties are exposed via the namespace so there is little benefit to using raw bean definitions. For more information about XML Schema-based configuration in Spring, see this appendix in the Spring Framework reference documentation.

Note

Spring Data Repository support uses a separate XML namespace. See Chapter 9, *GemFire Repositories* for more information on how to configure GemFire Repositories.

To use the Spring Data GemFire namespace, simply declare it in your Spring XML configuration:

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

    <bean id ... >

    ❹<gfe:cache ...>

</beans>
```

❶ Spring GemFire namespace prefix. Any name will do but through out the reference documentation, `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 Data 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

It 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="❶http://www.springframework.org/schema/gemfire"
    xmlxsi="http://www.w3.org/2001/XMLSchema-instance"
 xml❷beans="http://www.springframework.org/schema/beans"
 xsi:schemaLocation="
     http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring
     http://www.springframework.org/schema/gemfire http://www.springframework.org/schema/gemfire/sp

    ❸<beabean id ... >

    ❹<cache ...>

</beans>
```

❶ The default namespace declaration for this XML file points to the Spring Data 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).

5.3 Configuring the GemFire Cache

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

Note

This section describes the creation and configuration of a full cache member, appropriate for peer to peer cache topologies and cache servers. A full cache is also commonly used for standalone applications, integration tests and proofs of concept. In a typical production system, most application processes will act as cache clients and will create a ClientCache instance instead. This is described in the sections the section called “Configuring a GemFire Client Cache” and the section called “Client Region”

A cache with default configuration can be created with a very simple declaration:

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

A Spring application context containing this definition will, upon initialization, will register a `CacheFactoryBean` to create a Spring bean named `gemfireCache` referencing a GemFire `Cache` instance. This will be either an existing cache, or if one does not exist, a newly created one. Since no additional properties were specified, a newly created cache will apply the default cache configuration.

All Spring Data GemFire components which depend on the Cache respect this naming convention so that there is no need to explicitly declare the Cache dependency. If you prefer, you can make the dependence explicit via the `cache-ref` attribute provided by various namespace elements. Also you can easily override the Cache's bean name:

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

Starting with Spring Data GemFire 1.2.0, The GemFire Cache may be fully configured using Spring. However, GemFire's native XML configuration file (e.g., cache.xml) is also supported. For scenarios in which the GemFire cache needs to be configured natively, simply provide a reference the GemFire configuration file using the `cache-xml-location` attribute:

```
<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.

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 runtime environment or the prefix specified (if any) in the resource location.

In addition to referencing an external configuration file one can specify GemFire properties using any of Spring's common properties support features. For example, one can use the `properties` element defined in the `util` namespace to define properties directly or load properties from 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-g
        http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xs

     <gfe:cache properties-ref="props"/>

     <util:properties id="props" location="file:/vfabric/gemfire/gemfire.properties"/>
</beans>
```

Note

The cache settings apply only if a new cache needs to be created. If an open cache already exists in the JVM, these settings will be ignored.

Advanced Cache Configuration

For advanced cache configuration, the `cache` element provides a number of configuration options exposed as attributes or child elements

```
❶
<gfe:cache
        copy-on-read="true"
        critical-heap-percentage="70"
        eviction-heap-percentage="60"
        lock-lease="120"
        lock-timeout="60"
        pdx-serializer="myPdxSerializer"
        pdx-disk-store="diskStore"
        pdx-ignore-unread-fields="true"
        pdx-persistent="true"
        pdx-read-serialized="false"
        message-sync-interval="1"
        search-timeout="300"
    >
     ❷<gfe:transaction-listener ref="myTransactionListener"/>

     ❸<gfe:transaction-writer>
        <bean class="org.springframework.data.gemfire.example.TransactionListener"/>
```

```
      </gfe:transaction-writer>

     ❹<gfe:dynamic-region-factory/>
     ❺<gfe:jndi-binding jndi-name="myDataSource" type="ManagedDataSource"/>
</gfe:cache>
```

❶ Various cache options are supported by attributes. For further information regarding anything shown in this example, please consult the GemFire product documentation

❷ An example of a `TransactionListener` callback declaration using a bean reference. The referenced bean must implement TransactionListener

❸ An example of a `TransactionWriter` callback declaration using an inner bean declaration this time. The bean must implement TransactionWriter

❹ Enable GemFire's DynamicRegionFactory

❺ Declares a JNDI binding to enlist an external datasource in a GemFire transaction

Note

The `use-bean-factory-locator` attribute (not shown) deserves a mention. The factory bean responsible for creating the cache uses an internal Spring type called a `BeanFactoryLocator` to enable user classes declared in GemFire's native `cache.xml` to be registered as Spring beans. The `BeanFactoryLocator` implementation also permits only one bean definition for a cache with a given id. In certain situations, such as running JUnit integration tests from within Eclipse, it is necessary to disable the `BeanFactoryLocator` by setting this value to `false` to prevent an exception. This exception may also arise during JUnit tests running from a build script. In this case the test runner should be configured to fork a new JVM for each test (in maven, set `<forkmode>always</forkmode>`) . Generally there is no harm in setting this value to false.

Enabling PDX Serialization

The example above includes a number of attributes related to GemGire's enhanced serialization framework, PDX. While a complete discussion of PDX is beyond the scope of this reference guide, it is important to note that PDX is enabled by registering a PDX serializer which is done via the `pdx-serializer` attribute. GemFire provides an implementation class `com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer`, however it is common for developers to provide their own implementation. The value of the attribute is simply a reference to a Spring bean that implements the required interface. More information on serialization support can be found in Chapter 7, *Working with GemFire Serialization*

Configuring a GemFire Cache Server

In Spring Data GemFire 1.1 dedicated support for configuring a CacheServer was added, allowing complete configuration through the Spring container:

```
<?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:context="http://www.springframework.org/schema/context"
```

```
  xsi:schemaLocation="http://www.springframework.org/schema/gemfire http://www.springframework.org/schema/ge
  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-contex

  <gfe:cache />

  <!-- Advanced example depicting various cache server configuration options -->
  <gfe:cache-server id="advanced-config" auto-startup="true"
       bind-address="localhost" port="${gfe.port.6}" host-name-for-clients="localhost"
       load-poll-interval="2000" max-connections="22" max-threads="16"
       max-message-count="1000" max-time-between-pings="30000"
       groups="test-server">

     <gfe:subscription-config eviction-type="ENTRY" capacity="1000" disk-store="file://${java.io.tmpdir}"/>
   </gfe:cache-server>

   <context:property-placeholder location="classpath:cache-server.properties"/>

</beans>
```

The configuration above illustrates the `cache-server` element and the many options available.

Note

Rather than hard-coding the port, this configuration uses Spring's `context` namespace to declare a `property-placeholder`. The property placeholder reads one or more properties file and then replaces property placeholders with values at runtime. This allows administrators to change such values without having to touch the main application configuration. Spring also provides SpEL and the environment abstraction one to support externalization of environment specific properties from the main code base, easing the deployment across multiple machines.

Note

To avoid initialization problems, the `CacheServer`s started by Spring Data GemFire will start *after* the container has been fully initialized. This allows potential regions, listeners, writers or instantiators defined declaratively to be fully initialized and registered before the server starts accepting connections. Keep this in mind when programmatically configuring these items as the server might start before your components and thus not be seen by the clients connecting right away.

Configuring a GemFire Client Cache

Another configuration addition in Spring Data GemFire 1.1 is the dedicated support for configuring ClientCache. This is similar to a cache in both usage and definition and supported by `org.springframework.data.gemfire.clientClientCacheFactoryBean`.

```
<beans>
    <gfe:client-cache />
</beans>
```

`client-cache` supports much of the same options as the *cache* element. However as opposed to a *full* cache, a client cache connects to a remote cache server through a pool. By default a pool is created

to connect to a server on `localhost` port `40404`. The the default pool is used by all client regions unless the region is configured to use a different pool.

Pools can be defined through the `pool` element; The client side `pool`s can be used to configure connectivity to the server for individual entities or for the entire cache. For example, to customize the default pool used by `client-cache`, one needs to define a pool and wire it to cache definition:

```
<beans>
  <gfe:client-cache id="simple" pool-name="my-pool"/>

  <gfe:pool id="my-pool" subscription-enabled="true">
     <gfe:locator host="${locatorHost}" port="${locatorPort}"/>
 </gfe:pool>
</beans>
```

Client side configuration is covered in more detail in ???.

5.4 Configuring a GemFire Region

A region is required to store and retrieve data from the cache. `Region` is an interface extends `java.util.map` used to perform basic data access using familiar key-value semantics. The `Region` interface is wired into classes that require it so the actual region type is decoupled from the programming model . Typically each region is associated with one domain object, similar to a table in a relational database.

GemFire implements the following types of regions:

- *Replicated* - Data is replicated across all cache members that define the region. This provides very high read performance but writes take longer to perform the replication.
- *Partioned* - Data is partitioned into buckets among cache members that define the region. This provides high read and write performance and is suitable for very large data sets that are too big for a single node.
- *Local* - Data only exists on the local node.
- *Client* - Technically a client region is a local region that acts as a proxy to a replicated or partitioned region hosted on cache servers. It may hold data created or fetched locally, alternately it can be empty. Local updates are synchronized to the cache server. Also, a client region may subscribe to events in order to stay synchronized with changes originating from remote processes that access the same region.

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.

Using an externally configured Region

For referencing Regions already configured through GemFire native configuration, e.g., a `cache.xml` file, use the `lookup-region` element. Simply declare the target region name with 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's `id` will be used. 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. For configuring new GemFire regions, proceed to the appropriate sections below.

Note that in the previous examples, since no cache name was defined, the default naming convention (`gemfireCache`) was used. Alternately, one can reference 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.

Configuring Regions

Spring Data GemFire provides comprehensive support for configuring any type of GemFire Region via the following elements:

- Local Region `<local-region>`
- Replicated Region `<replicated-region>`
- Partitioned Region `<partitioned-region>`
- Client Region `<client-region>`

For a comprehensive description of region types please consult the GemFire product documentation.

Common Region Attributes

The following table(s) list attributes available for various region types:

Table 5.1. Common Region Attributes

Name	Values	Description
cache-ref	*GemFire cache bean name*	The name of the bean defining the GemFire cache (by default 'gemfireCache').
close	*boolean, default:true*	Indicates whether the region should be closed at shutdown

Name	Values	Description
data-policy	*See GemFire's Data Policy*	The region's data policy. Note not all data policies are supported for every region type
destroy	*boolean, default:false*	Indicates whether the region should be destroyed at shutdown
disk-store-ref	*The name of a configured disk store*	A reference to a bean created via the `disk-store` element. Note: This will automatically enable persistence. If persistent is explicitly set to false, an exception will be thrown.
disk-synchronous	*boolean, default:false*	Indicates whether disk store writes are synchronous. Note: This will automatically enable persistence. If persistent is explicitly set to false, an exception will be thrown.
enable-gateway	*boolean, default:false*	Indicates whether the region will synchronize entries over a WAN gateway.
hub-id	*The name of the Gateway Hub*	This will automatically set enable-gateway to true. If enable-gateway is explicitly set to false, an exception will be thrown.
id	*any valid bean name*	Will also be the region name by default
ignore-jta	*boolean, default:false*	Indicates whether the region participates in JTA transactions
index-update-type	*synchronous or asynchronous, default:asynchronous*	Indicates whether indices will be updated synchronously or asynchronously on entry creation
initial-capacity	*integer, default:16*	The initial memory allocation for number of entries
key-constraint	*any valid java class name*	The expected key type
name	*any valid region name*	The name of the region definition. If no specified, it will assume the value of the id attribute (the bean name).
persistent	*boolean, default:false*	Indicates whether the region persists entries to a disk store

Name	Values	Description
statistics	*boolean, default:false*	Indicates whether the region reports statistics
value-constraint	*any valid java class name*	The expected value type

Cache Listeners

Cache Listeners are registered with a region to handle region events such as entries being created, updated, destroyed, etc. A Cache Listener can be any bean that implements the `CacheListener` interface. A region may have multiple listeners, declared using the `cache-listener` element enclosed in a `*-region` element. In the example below, there are two `CacheListener`s declared. The first references a top level named Spring bean; the second is an anonymous inner bean definition.

```
<gfe:replicated-region id="region-with-listeners">
    <gfe:cache-listener>
        <!-- nested cache listener reference -->
        <ref bean="c-listener"/>
        <!-- nested cache listener declaration -->
        <bean class="some.pkg.AnotherSimpleCacheListener"/>
    </gfe:cache-listener>

    <bean id="c-listener" class="some.pkg.SimpleCacheListener"/>
 </gfe:replicated-region>
```

The following example uses an alternate form of the `cache-listener` element with a `ref` attribute. This allows for more concise configuration for a single cache listener. Note that the namespace only allows a single `cache-listener` element so either the style above or below must be used.

Caution

Using `ref` and a nested declaration in a `cache-listener`, or similar element, is illegal. The two options are mutually exclusive and using both on the same element will result in an exception.

```
<beans>
    <gfe:replicated-region id="region-with-one listener">
        <gfe:cache-listener ref="c-listener"/>
    </gfe:replicated-region>

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

Bean Reference Conventions

The `cache-listener` element is an example of a common pattern used in the namespace anywhere GemFire provides a callback interface to be implemented in order to invoke custom code in response to cache or region events. Using Spring's IoC container, the implementation is a standard Spring bean. In order to simplify the configuration, the schema allows a single

occurrence of the `cache-listener` element, but it may contain nested bean references and inner bean definitions in any combination if multiple instances are permitted. The convention is to use the singular form (i.e., `cache-listener` vs `cache-listeners`) reflecting that the most common scenario will in fact be a single instance. We have already seen examples of this pattern in the advanced cache configuration example.

Cache Loaders and Cache Writers

Similar to `cache-listener`, the namespace provides `cache-loader` and `cache-writer` elements to register these respective components for a region. A `CacheLoader` is invoked on a cache miss to allow an entry to be loaded from an external source, a database for example. A `CacheWriter` is invoked afer an entry is created or updated,intended for synchronizing to an external data source. The difference is GemFire only supports at most a single instance of each for each region. However, either declaration style may be used. See `CacheLoader` and `CacheWriter` for more details.

Subregions

In Release 1.2.0, Spring Data GemFire added support for subregions, allowing regions to be arranged in a hierarchical relationship. For example, GemFire allows for a */Customer/Address* region and a different */Employee/Address* region. Additionally, a subregion may have it's own subregions and its own configuration. A subregion does not inherit attributes from the parent region. Regions types may be mixed and matched subject to GemFire constraints. A subregion is naturally declared as a child element of a region. The subregion's name attribute is the simple name. The above example might be configured as:

```
<beans>

    <gfe:replicated-region name="Customer">
        <gfe:replicated-region name="Address"/>
    </gfe:replicated-region>

    <gfe:replicated-region name="Employee">
        <gfe:replicated-region name="Address"/>
    </gfe:replicated-region>

</beans>
```

Note that the `id` attribute is not permitted for a subregion. The subregions will be created with bean names */Customer/Address* and */Employee/Address*, respectively. So they may be injected using the full path name into other beans that use them, such as `GemfireTemplate`. The full path should also be used in OQL query strings.

Data Persistence

Regions can be made persistent. 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 Spring Data GemFire, to enable persistence, simply set the `persistent` attribute to true:

```
<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. The diskstore is referenced using the disk-store-ref attribute. Additionally, the region may perform disk writes synchronously or asynchronously:

```
<gfe:partitioned-region id="persitent-partition" persistent="true" disk-store-ref="myDiskStore" disk-synchr
```

This is discussed further in Section 5.6, “Configuring a Disk Store”

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 to the least recently used entry (also known as LRU). Evicted entries are either destroyed or paged to disk (also known as *overflow*).

Spring Data GemFire 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).

Data Expiration

GemFire allows you to control how long entries exist in the cache. Eviction is driven by elapsed time, as opposed to eviction which is driven by memory usage. Once an entry expires it may no longer be accessed from the cache. GemFire supports the following expiration types:

- *Time to live (TTL)* - The amount of time, in seconds, the object may remain in the cache after the last creation or update. For entries, the counter is set to zero for create and put operations. Region counters are reset when the region is created and when an entry has its counter reset.
- *Idle timeout* - The amount of time, in seconds, the object may remain in the cache after the last access. The idle timeout counter for an object is reset any time its TTL counter is reset. In addition, an entry’s

idle timeout counter is reset any time the entry is accessed through a get operation or a netSearch . The idle timeout counter for a region is reset whenever the idle timeout is reset for one of its entries.

Each of these may be applied to the region itself or entries in the region. Spring Data GemFire provides `<region-ttl>`, `<region-tti>`, `<entry-ttl>` and `<entry-tti>` region child elements to specify timeout values and expiration actions.

Local Region

Spring Data GemFire offers a dedicated `local-region` element for creating local regions. Local regions, as the name implies, are standalone meaning they do not share data with any other distributed system member. Other than that, all common region configuration options are supported. A minimal declaration looks as follows (again, the example relies on the Spring Data GemFire namespace naming conventions to wire the cache):

```
<gfe:local-region id="myLocalRegion" />
```

Here, a local region is created (if one doesn't exist already). The name of the region is the same as the bean id (myLocalRegion) and the bean assumes the existence of a GemFire cache named `gemfireCache`.

Replicated Region

One of the common region types is a *replicated region* or *replica*. In short, when a region is configured to be a replicated region, every member that hosts that region stores a copy of the region's entries 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 rep

Spring Data GemFire offers a `replicated-region` element. A minimal declaration looks as follows. All common configuration options are available for replicated regions.

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

Partitioned Region

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

"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 is created using 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>
```

`partitioned-region` Options

The following table offers a quick overview of configuration options specific to partitioned regions. These are in addition to the common region configuration options described above.

Table 5.2. partitioned-region options

Name	Values	Description
partition-resolver	*bean name*	The name of the partitioned resolver used by this region, for custom partitioning.
partition-listener	*bean name*	The name of the partitioned listener used by this region, for handling partition events.
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 storage.
colocated-with	*valid region name*	The name of the partitioned region with which this newly created partitioned region is colocated.
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.

Name	Values	Description
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.

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 classified in short in: peer-to-peer (p2p), client-server, and wide area cache network (or WAN). In the last two scenarios, it is common to declare *client* regions which connect to a cache server. Spring Data GemFire offers dedicated support for such configuration through the section called "Configuring a GemFire Client Cache", `client-region` and `pool` elements. As the names imply, the former defines a client region while the latter defines connection pools to be used/shared by the various client regions.

Below is a typical client region configuration:

```
<!-- client region using the default client-cache pool -->
<gfe:client-region id="simple">
    <gfe:cache-listener ref="c-listener"/>
</gfe:client-region>

<!-- region using its own dedicated pool -->
<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="true">
    <gfe:locator host="someHost" port="40403"/>
</gfe:pool>
```

As with the other region types, `client-region` supports `CacheListener`s (but not CacheLoaders or CacheWriters). It also requires a connection `pool` for connecting to a server. Each client can have its own pool or they can share the same one.

Note

In the above example, the pool is configured with a `locator`. The locator is a separate process used to discover cache servers in the distributed system and are recommended for production systems. It is also possible to configure the pool to connect directly to one or more cache servers using the `server` element.

For a full list of options to set on the client and especially on the pool, please refer to the Spring Data GemFire schema (Appendix A, *Spring Data GemFire Schema*) and the GemFire documentation.

Client Interests

To minimize network traffic, each client can define its own 'interest', pointing out to GemFire, the data it actually needs. In Spring Data GemFire, 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">
             <constructor-arg value="someKey" />
        </bean>
    </gfe:key-interest>
    <gfe:regex-interest pattern=".*" receive-values="false"/>
</gfe:client-region>
```

A special key `ALL_KEYS` means interest is registered for all keys (identical to a regex interest of `.*`). The `receive-values` attribute indicates whether or not the values are received for create and update events. If true, values are received; if false, only invalidation events are received - refer to the GemFire documentation for more details.

5.5 Creating an Index

GemFire allows creation on indexes (or indices) to improve the performance of (common) queries. Spring Data GemFire allows indecies to be declared through the `index` element:

```
<gfe:index id="myIndex" expression="someField" from="/someRegion"/>
```

Before creating an index, Spring Data GemFire will verify whether one with the same name already exists. If it does, it will compare the properties and if they don't match, will remove the old one to create a new one. If the properties match, Spring Data GemFire will simply return the index (in case it does not exist it will simply create one). To prevent the update of the index, even if the properties do not match, set the property `override` to false.

Note that index declaration are not bound to a region but rather are top-level elements (just like `gfe:cache`). This allows one to declare any number of indecies on any region whether they are just created or already exist - an improvement versus the GemFire `cache.xml`. By default the index relies on the default cache declaration but one can customize it accordingly or use a pool (if need be) - see the namespace schema for the full set of options.

5.6 Configuring a Disk Store

As of Release 1.2.0, Spring Data GemFire supports disk store configuration via a top level `disk-store` element.

Note

Prior to Release 1.2.0, `disk-store` was a child element of `*-region`. If you have regions configured with disk storage using a prior release of Spring Data GemFire and want to upgrade to the latest release, move the disk-store element to the top level, assign an id and use the

region's `disk-store-ref` attribute. Also, `disk-synchronous` is now a region level attribute.

```
<gfe:disk-store id="diskStore1" queue-size="50" auto-compact="true"
        max-oplog-size="10" time-interval="9999">
        <gfe:disk-dir location="/gemfire/store1/" max-size="20"/>
        <gfe:disk-dir location="/gemfire/store2/" max-size="20"/>
</gfe:disk-store>
```

Disk stores are used by regions for file system persistent backup or overflow storage of evicted entries, and persistent backup of WAN gateways. Note that multiple components may share the same disk store. Also multiple directories may be defined for a single disk store. Please refer to the GemFire documentation for an explanation of the configuration options.

5.7 Configuring GemFire's Function Service

As of Release 1.2.0, Spring Data GemFire provides namespace support for registering GemFire `Function`s for remote function execution. Please refer to the GemFire documentation for more information on the function execution framework. Functions are declared as Spring beans and must implement the `com.gemstone.gemfire.cache.execute.Function` interface or extend `com.gemstone.gemfire.cache.execute.FunctionAdapter`. The namespace uses a familiar pattern to declare functions:

```
<gfe:function-service>
  <gfe:function>
      <bean class="com.company.example.Function1"/>
      <ref bean="function2"/>
  </gfe:function>
</gfe:function-service>

<bean id="function2" class="com.company.example.Function2"/>
```

5.8 Configuring WAN Gateways

WAN gateways provide a way to synchronize GemFire distributed systems across geographic distributed areas. As of Release 1.2.0, Spring Data GemFire provides namespace support for configuring WAN gateways as illustrated in the following example:

```
<gfe:cache/>

<gfe:replicated-region id="region-with-gateway" enable-gateway="true" hub-id="gateway-hub"/>

<gfe:gateway-hub id="gateway-hub" manual-start="true">
    <gfe:gateway gateway-id="gateway">
         <gfe:gateway-listener>
             <bean class="com.company.example.MyGatewayListener"/>
         </gfe:gateway-listener>
         <gfe:gateway-queue maximum-queue-memory="5" batch-size="3"
                batch-time-interval="10" />
     </gfe:gateway>

     <gfe:gateway gateway-id="gateway2">
         <gfe:gateway-endpoint port="1234" host="host1" endpoint-id="endpoint1"/>
```

```
        <gfe:gateway-endpoint port="2345" host="host2" endpoint-id="endpoint2"/>
    </gfe:gateway>
</gfe:gateway-hub>
```

A region may synchronize all or part of its contents to a gateway hub used to access one or more remote systems. The region must set `enable-gateway` to `true` and specify the `hub-id`.

Note

If just a hub-id is specified, Spring Data GemFire automatically assumes that the gateway should be enabled.

Please refer to the GemFire product document for a detailed explanation of all the configuration options.

6. 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.

6.1 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, e.g., using a `<gfe:cache/>` declaration) as it acts as an exception translator which is automatically detected by the Spring infrastructure and used accordingly.

6.2 GemfireTemplate

As with many other high-level abstractions provided by the Spring projects, Spring Data GemFire provides a *template* that simplifies GemFire data access. The class provides several *one-line* methods, for common region operations but also the ability to *execute* code against the native GemFire API without having to deal with GemFire checked 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");
  }
});
```

For accessing the full power of the GemFire query language, one can use the `find` and `findUnique` which, as opposed to the `query` method, can execute queries across multiple regions, execute projections, and the like. The `find` method should be used when the query selects multiple items (through `SelectResults`) and the latter, `findUnique`, as the name suggests, when only one object is returned.

6.3 Support for Spring Cache Abstraction

Since 1.1, Spring GemFire provides an implementation for Spring 3.1 cache abstraction. To use GemFire as a backing implementation, simply add `GemfireCacheManager` to your configuration:

```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
  xmlns:cache="http://www.springframework.org/schema/cache"
  xmlns:gfe="http://www.springframework.org/schema/gemfire"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/bea
     http://www.springframework.org/schema/gemfire http://www.springframework.org/schema/gemfire/spring-gem
      http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache.x
  <!-- turn on declarative caching -->
  <cache:annotation-driven />

  <gfe:cache id="gemfire-cache" />
  <!-- declare GemFire Cache Manager -->
  <bean id="cacheManager" class="org.springframework.data.gemfire.support.GemfireCacheManager" p:cache-ref="
</beans>
```

6.4 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, Spring Data GemFire provides a dedicated, per-cache, transaction manager that once declared, allows region operations to be 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 `gemfireCache`. As with the other Spring Data GemFire namespace elements, if the cache name is not configured, the aforementioned naming convention will used. Additionally, the transaction manager name, if not specified is `gemfireTransactionManager`.

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 behavior 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.

6.5 GemFire Continuous Query Container

A powerful functionality offered by GemFire is continuous querying (or CQ). In short, CQ allows one to create a query and automatically be notified when new data that gets added to GemFire matches the query. Spring GemFire provides dedicated support for CQs through the `org.springframework.data.gemfire.listener` package and its *listener container*; very similar in functionality and naming to the JMS integration in Spring Framework; in fact, users familiar with the JMS support in Spring, should feel right at home. Basically Spring Data GemFire allows methods on POJOs to become end-points for CQ - simply define the query and indicate the method that should be notified when there is a match - Spring Data GemFire takes care of the rest. This is similar Java EE's message-driven bean style, but without any requirement for base class or interface implementations, based on GemFire.

Note

Currently, continuous queries are supported by GemFire only in client/server topologies. Additionally the pool used is required to have the `subscription` property enabled. Please refer to the documentation for more information.

Continuous Query Listener Container

Spring Data GemFire simplifies the creation, registration, life-cycle and dispatch of CQs by taking care of the infrastructure around them through `ContinuousQueryListenerContainer` which does all the heavy lifting on behalf of the user - users familiar with EJB and JMS should find the concepts familiar as it is designed as close as possible to the support in Spring Framework and its message-driven POJOs (MDPs)

`ContinuousQueryListenerContainer` acts as an event (or message) listener container; it is used to receive the events from the registered CQs and drive the POJOs that are injected into it. The listener container is responsible for all threading of message reception and dispatches into the listener for processing. It acts as the intermediary between an EDP (Event Driven POJO) and the event provider and takes care of creation and registration of CQs (to receive events), resource acquisition and release, exception conversion and the like. This allows you as an application developer to write the (possibly complex) business logic associated with receiving an event (and reacting to it), and delegates boilerplate GemFire infrastructure concerns to the framework.

The container is fully customizable - one can chose either to use the CQ thread to perform the dispatch (synchronous delivery) or a new thread (from an existing pool for examples) for an asynchronous approach by defining the suitable `java.util.concurrent.Executor` (or Spring's `TaskExecutor`). Depending on the load, the number of listeners or the runtime environment, one should change or tweak the executor to better serve her needs - in particular in managed environments

(such as app servers), it is highly recommended to pick a a proper `TaskExecutor` to take advantage of its runtime.

The `ContinuousQueryListenerAdapter` and `ContinuousQueryListener`

The `ContinuousQueryListenerAdapter` class is the final component in Spring Data GemFire CQ support: in a nutshell, it allows you to expose almost *any* class as a EDP (there are of course some constraints) - it implements `ContinuousQueryListener`, a simpler listener interface similar to GemFire CqListener.

Consider the following interface definition. Notice the various event handling methods and their parameters:

```
public interface EventDelegate {
     void handleEvent(CqEvent event);
     void handleEvent(Operation baseOp);
     void handleEvent(Object key);
     void handleEvent(Object key, Object newValue);
     void handleEvent(Throwable th);
     void handleQuery(CqQuery cq);
     void handleEvent(CqEvent event, Operation baseOp, byte[] deltaValue);
     void handleEvent(CqEvent event, Operation baseOp, Operation queryOp, Object key, Object newValue);
}
```

```
public class DefaultEventDelegate implements EventDelegate {
    // implementation elided for clarity...
}
```

In particular, note how the above implementation of the `EventDelegate` interface (the above `DefaultEventDelegate` class) has *no* GemFire dependencies at all. It truly is a POJO that we will make into an EDP via the following configuration (note that the class doesn't have to implement an interface, one is present only to better show case the decoupling between contract and implementation).

```
<?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"
    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-g


 <gfe:client-cache pool-name="client"/>

 <gfe:pool id="client" subscription-enabled="true">
    <gfe:server host="localhost" port="40404"/>
 </gfe:pool>

 <gfe:cq-listener-container>
    <!-- default handle method -->
    <gfe:listener ref="listener" query="SELECT * from /region"/ >
    <gfe:listener ref="another-listener" query="SELECT * from /another-region" name="my-query" method="hand
 </gfe:cq-listener-container>

 <bean id="listener" class="gemfireexample.DefaultMessageDelegate"/>
 <bean id="another-listener" class="gemfireexample.DefaultMessageDelegate"/>
```

```
  ...
<beans>
```

Note

The example above shows some of the various forms that a listener can have; at its minimum the listener reference and the actual query definition are required. It's possible however to specify a name for the resulting continuous query (useful for monitoring) but also the name of the method (the default is `handleEvent`). The specified method can have various argument types, the `EventDelegate` interface lists the allowed types.

The example above uses the Spring Data GemFire namespace to declare the event listener container and automatically register the listeners. The full blown, *beans* definition is displayed below:

```
<!-- this is the Event Driven POJO (MDP) -->
<bean id="eventListener" class="org.springframework.data.gemfire.listener.adapter.ContinuousQueryListenerAda
    <constructor-arg>
        <bean class="gemfireexample.DefaultEventDelegate"/>
    </constructor-arg>
</bean>

<!-- and this is the event listener container... -->
<bean id="gemfireListenerContainer" class="org.springframework.data.gemfire.listener.ContinuousQueryListener
    <property name="cache" ref="gemfireCache"/>
    <property name="queryListeners">
      <!-- set of listeners -->
      <set>
        <bean class="org.springframework.data.gemfire.listener.ContinuousQueryDefinition" >
               <constructor-arg value="SELECT * from /region" />
               <constructor-arg ref="eventListener" />
        </bean>
      </set>
    </property>
</bean>
```

Each time an event is received, the adapter automatically performs type translation between the GemFire event and the required method argument(s) transparently. Any exception caused by the method invocation is caught and handled by the container (by default, being logged).

6.6 Wiring `Declarable` components

GemFire XML configuration (usually named `cache.xml` allows *user* objects to be declared as part of the configuration. Usually these objects are `CacheLoader`s or other pluggable callback components supported by GemFire. Using native GemFire configuration, each user 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 these 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 objects.

However, if you are starting 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 through Spring as mentioned in Section 5.4, “Configuring a GemFire Region”. That way, one does not have to implement the `Declarable` interface and also benefits from all the features of the Spring IoC container (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, Spring Data GemFire 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. In other words, there is no easy way to manage object creation outside GemFire.

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 the id/name `com.company.app.DBLoader` was 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..).

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.

7. Working with GemFire Serialization

To improve overall performance of the data grid, GemFire supports a dedicated serialization protocol (PDX) 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 Spring Data GemFire simplifies and improves GemFire custom serialization in Java.

7.1 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, Spring Data GemFire 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 6.6, “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.

7.2 Auto-generating custom `Instantiators`

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, Spring Data GemFire 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.InstantiatorFactoryBe
  <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.

8. POJO mapping

8.1 Entity mapping

Spring Data GemFire provides support to map entities to be stored in a GemFire grid. The mapping metadata is define by using annotations at the domain classes just like this:

```
@Region("myRegion")
public class Person {

  @Id Long id;
  String firstname;
  String lastname;

  @PersistenceConstructor
  public Person(String firstname, String lastname) {
    // …
  }

  …
}
```

Example 8.1 Mapping a domain class to GemFire

The first thing you see here is the `@Region` annotation that can be used to customize the region instances of the `Person` class are stored in. The `@Id` annotation can be used to annotate the property that shall be used as cache key. The `@PersistenceConstructor` annotation actually helps disambiguating multiple potentially available constructors taking parameters and explicitly marking the one annotated as the one to be used to create entities. With none or only a single constructor you can omit the annotation.

8.2 Mapping PDX serializer

Spring Data GemFire provides a custom `PDXSerializer` implementation that uses the mapping information to customize entity serialization. Beyond that it allows customizing the entity instantiation by using the Spring Data `EntityInstantiator` abstraction. By default the serializer uses a `ReflectionEntityInstantiator` that will use the persistence constructor of the mapped entity (either the single declared one or explicitly annoted with `@PersistenceConstructor`). To provide values for constructor parameters it will read fields with name of the constructor parameters from the `PDXReader` supplied.

```
public class Person {

  public Person(@Value("#root.foo") String firstname, @Value("bean") String lastname) {
    // …
  }

}
```

Example 8.2 Using @Value on entity constructor parameters

The entity annotated as such will get the field `foo` read from the `PDXReader` and handed as constructor parameter value for `firstname`. The value for `lastname` will be the Spring bean with name `bean`.

9. GemFire Repositories

9.1 Introduction

Spring Data GemFire provides support to use the Spring Data repository abstraction to easily persist entities into GemFire and execute queries. A general introduction into the repository programming model is been provided here .

9.2 Spring configuration

To bootstrap Spring Data repositories you use the `<repositories />` element from the GemFire namespace:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:gf="http://www.springframework.org/schema/gemfire"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       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-gemfire.xsd>

  <gf:repositories base-package="com.acme.repository" />

</beans>
```

Example 9.1 Bootstrap GemFire repositories

This configuration snippet will look for interfaces below the configured base package and create repository instances for those interfaces backed by a `SimpleGemFireRepository`. Note that you have to have your domain classes correctly mapped to configured regions as the bottstrap process will fail otherwise.

9.3 Executing OQL queries

The GemFire repositories allow the definition of query methods to easily execute OQL queries against the Region the managed entity is mapped to.

```
@Region("myRegion")
public class Person { … }
```

```
public interface PersonRepository extends CrudRepository<Person, Long> {

  Person findByEmailAddress(String emailAddress);

  Collection<Person> findByFirstname(String firstname);

  @Query("SELECT * FROM /Person p WHERE p.firstname = $1")
  Collection<Person> findByFirstnameAnnotated(String firstname);

  @Query("SELECT * FROM /Person p WHERE p.firstname IN SET $1")
  Collection<Person> findByFirstnamesAnnotated(Collection<String> firstnames);
}
```

Example 9.2 Sample repository

The first method listed here will cause the following query to be derived: `SELECT x FROM /myRegion x WHERE x.emailAddress = $1`. The second method works the same way except it's returning all entities found whereas the first one expects a single result value. In case the supported keywords are not sufficient to declare your query or the method name gets to verbose you can annotate the query methods with `@Query` as seen for methods 3 and 4.

Table 9.1. Supported keywords for query methods

Keyword	Sample	Logical result
`GreaterThan`	`findByAgeGreaterThan(int age)`	`x.age > $1`
`GreaterThanEqual`	`findByAgeGreaterThanEqual(int age)`	`x.age >= $1`
`LessThan`	`findByAgeLessThan(int age)`	`x.age < $1`
`LessThanEqual`	`findByAgeLessThanEqual(int age)`	`x.age <= $1`
`IsNotNull`, `NotNull`	`findByFirstnameNotNull()`	`x.firstname =! NULL`
`IsNull`, `Null`	`findByFirstnameNull()`	`x.firstname = NULL`
`In`	`findByFirstnameIn(Collection<String> x)`	`x.firstname IN SET $1`
`NotIn`	`findByFirstnameNotIn(Collection<String> x)`	`x.firstname NOT IN SET $1`
(No keyword)	`findByFirstname(String name)`	`x.firstname = $1`
`Not`	`findByFirstnameNot(String name)`	`x.firstname != $1`

Keyword	Sample	Logical result
`IsTrue`, `True`	`findByActiveIsTrue()`	`x.active = true`
`IsFalse`, `False`	`findByActiveIsFalse()`	`x.active = false`

10. Sample Applications

The Spring Data 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.

Additional sample applications may be found in the Spring Data GemFire Examples repository.

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.

10.1 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.

Running under Linux

If you experience networking problems when starting GemFire or the samples, try adding the following system property `java.net.preferIPv4Stack=true` to the command line (insert `-Djava.net.preferIPv4Stack=true`). For an alternative (global) fix especially on Ubuntu see this link

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.

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).

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.

11. Useful Links

- Spring Data GemFire Home Page
- vFabric GemFire Home Page
- vFabric GemFire Documentation
- GemFire Community Home Page
- Spring Data GemFire Forum

Part IV. Appendices

Appendix A. Spring Data GemFire Schema

Spring Data GemFire Core Schema (gfe)

```
<?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"
    xmlns:repository="http://www.springframework.org/schema/data/repository"
    targetNamespace="http://www.springframework.org/schema/gemfire"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="1.2">
    <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:complexType name="cacheBaseType">
        <xsd:sequence>
            <xsd:element name="transaction-listener" type="beanDeclarationType"
                minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Registers a bean as a TransactionListener with the CacheTransactionManager. The bean must implement com.gems
and may be nested or referenced.
   ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="transaction-writer" type="beanDeclarationType"
                minOccurs="0" maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Registers a bean as a TransactionWriter with the CacheTransactionManager. The bean must implement com.gemsto
and may be nested or referenced.
   ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:element>

            <xsd:element name="dynamic-region-factory"
                minOccurs="0" maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Enables Dynamic Regions and specifies their configuration.
            ]]></xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                    <xsd:attribute name="disk-dir" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Specifies the directory path for disk persistence for dynamic regions.
            ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
```

```
                    <xsd:attribute name="persistent" type="xsd:string"
                        default="true">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Enables persistence for dynamic regions.
            ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="register-interest"
                        type="xsd:string" default="true">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Specifies whether dynamic regions register interest in all keys in a corresponding server region.
            ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="jndi-binding" type="jndiBindingType"
                minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Configures a data source to be bound to a JNDI context for use with Gemfire transactions
            ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="copy-on-read" type="xsd:string"
            use="optional" default="false">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Controls whether entry value retrieval methods return direct references to the entry value objects in the ca
or copies of the objects (true).
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="id" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The name of the cache definition (by default "gemfireCache").]]></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"><![CDATA[
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 propertie
consider using a dedicated utility such as the <util:*/> namespace and its 'properties' element.
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="use-bean-factory-locator"
```

```
            type="xsd:string" use="optional" default="true">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates whether a bean factory locator is enabled (default) for this cache definition or not. The locator
the enclosing bean factory reference to allow auto-wiring of Spring beans into GemFire managed classes. Usua
when the same cache is used in multiple application context/bean factories inside the same VM.
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="pdx-serializer" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Sets the PDX serializer for the cache. If this serializer is set, it will be consulted to see if it can seri
domain classes which are added to the cache in portable data exchange format.
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="pdx-disk-store" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Sets the name of the disk store to use for PDX meta data. When serializing objects in the PDX format,
the type definitions are persisted to disk. This setting controls which disk store is used for that persiste
If not set, the metadata will go in the default disk store.
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="pdx-persistent" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Control whether the type metadata for PDX objects is persisted to disk. The default for this setting is true
If you are using a WAN gateway, or persistent regions, you should leave this set to true.
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="pdx-read-serialized" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Sets the object preference to PdxInstance type. When a cached object that was serialized as a PDX is read fr
a PdxInstance will be returned instead of the actual domain class. The PdxInstance is an interface that prov
access to the fields of a PDX without deserializing the entire PDX. The PdxInstance implementation is a ligh
that simply refers to the raw bytes of the PDX that are kept in the cache. Using this method applications ca
access PdxInstance instead of Java object.

Note that a PdxInstance is only returned if a serialized PDX is found in the cache. If the cache contains a
then a domain class instance is returned instead of a PdxInstance.
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="pdx-ignore-unread-fields"
            type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Controls whether pdx ignores fields that were unread during deserialization. The default is to preserve unre
including their data during serialization. But if you configure the cache to ignore unread fields then their
lost during serialization.

You should only set this attribute to true if you know this member will only be reading cache data. In this
do not need to pay the cost of preserving the unread fields since you will never be reserializing pdx data.
```

```
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="critical-heap-percentage">
            <xsd:annotation>
                <xsd:documentation
                    source="com.gemstone.gemfire.cache.control.ResourceManager"><![CDATA[
Set the percentage of heap at or above which the cache is considered in danger of becoming inoperable
due to garbage collection pauses or out of memory exceptions. Changing this value can cause a LowMemoryExcep
be thrown during certain cache operation.  This feature requires additional VM flags to perform properly (s
JavaDocs for com.gemstone.gemfire.cache.control.ResourceManager for more information).
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="eviction-heap-percentage">
            <xsd:annotation>
                <xsd:documentation
                    source="com.gemstone.gemfire.cache.control.ResourceManager"><![CDATA[
Set the percentage of heap at or above which the eviction should begin on Regions configured for HeapLRU evi
This feature requires additional VM flags to perform properly (see the
JavaDocs for com.gemstone.gemfire.cache.control.ResourceManager for more information).
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <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:complexContent>
                <xsd:extension base="cacheBaseType">
                    <xsd:attribute name="lock-timeout"
                        type="xsd:string" use="optional" default="60">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
The timeout, in seconds, for implicit object lock requests. This setting affects automatic locking only,
 and does not apply to manual locking. If a lock request does not return before the specified timeout perio
  it is cancelled and returns with a failure.
                  ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="lock-lease" type="xsd:string"
                        use="optional" default="120">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
The timeout, in seconds, for implicit and explicit object lock leases. This affects both automatic locking a
Once a lock is obtained, it can remain in force for the lock lease time period before being automatically c
                ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="message-sync-interval"
                        type="xsd:string" use="optional" default="1">
```

```
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Used for client subscription queue synchronization when this member acts as a server to clients and server
Sets the frequency (in seconds) at which the primary server sends messages to its secondary servers to remov
that have already been processed by the clients.
                ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="search-timeout"
                        type="xsd:string" use="optional" default="300">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
How many seconds a netSearch operation can wait for data before timing out.
You may want to change this based on your knowledge of the network load or other factors.
                ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:element name="client-cache">
        <xsd:annotation>
            <xsd:documentation
                source="org.springframework.data.gemfire.client.ClientCacheFactoryBean"><![CDATA[
Defines a GemFire Client Cache instance used for creating or retrieving 'regions'.
   ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports
                        type="com.gemstone.gemfire.cache.client.ClientCache" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="cacheBaseType">
                    <xsd:attribute name="pool-name" type="xsd:string"
                        use="optional">
                        <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:element name="transaction-manager">
        <xsd:annotation>
            <xsd:documentation
                source="org.springframework.data.gemfire.GemfireTransactionManager"><![CDATA[
Defines a GemFire Transaction Manager instance for a single GemFire cache.
   ]]></xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:attribute name="id" type="xsd:string"
                use="optional">
                <xsd:annotation>
```

```
                    <xsd:documentation><![CDATA[
The name of the transaction manager definition (by default "gemfireTransactionManager").]]></xsd:documentat
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="cache-ref" type="xsd:string"
                use="optional" default="gemfireCache">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the bean defining the GemFire cache (by default 'gemfireCache').
     ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="copy-on-read" type="xsd:string"
                use="optional" default="true">
                <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" processContents="skip"
                minOccurs="0" maxOccurs="1">
                <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="baseLookupRegionType">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
Defines a lookup Subregion
                    ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="com.gemstone.gemfire.cache.Region" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="basicRegionType">
                <xsd:group ref="subRegionGroup" minOccurs="0"
                    maxOccurs="unbounded" />
            </xsd:extension>
```

```
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="basicSubRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseLookupRegionType">
                <xsd:attribute name="name" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the region definition.]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </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:complexType>
    <!-- -->
    <xsd:complexType name="baseReadOnlyRegionType"
        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.CacheListener"><![CDATA[
A cache listener definition for this region. A cache listener handles region or entry related events (that
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
it is easy to generate a deadlock by interacting with the region. For this reason, it is highly recommended
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.cache.CacheListener" />
                                </tool:annotation>
                            </xsd:appinfo>
                        </xsd:annotation>
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:any namespace="##other"
                                    processContents="skip" minOccurs="0"
                                    maxOccurs="unbounded">
                                    <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="optional">
```

```
                                <xsd:annotation>
                                    <xsd:documentation><![CDATA[
The name of the cache listener bean referred by this declaration. Used as a convenience method. If no refere
use inner bean declarations.
         ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="membership-attributes"
                        minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Configures a Region to require one or more membership roles to be present in the system for reliable access
                        ]]></xsd:documentation>
                        </xsd:annotation>
                        <xsd:complexType>
                            <xsd:attribute name="required-roles"
                                type="xsd:string" use="required">
                                <xsd:annotation>
                                    <xsd:documentation><![CDATA[
A comma delimited list of required role names
                            ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                            <xsd:attribute name="loss-action"
                                type="xsd:string" use="optional">
                                <xsd:annotation>
                                    <xsd:documentation><![CDATA[
Specifies the behavior when one or more required roles are missing:
(full-access, limited-access, no-access, or reconnect)
                            ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                            <xsd:attribute name="resumption-action"
                                type="xsd:string" use="optional">
                                <xsd:annotation>
                                    <xsd:documentation><![CDATA[
Specifies how the region is affected by resumption of reliability
when one or more missing required roles is restored to the distributed membership (none or reinitialize)
                            ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="persistent" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether the defined region is persistent. GemFire ensures that all the data you put into a region
 is configured for persistence will be written to disk in a way that it can be recovered the next time you c
region. This allows data to be recovered after a machine or process failure or after an orderly shutdown and
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:attribute name="disk-synchronous"
                    type="xsd:string" default="false">
```

```
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether the writing to the disk is synchronous or not. Default is false, meaning asynchronous writ

Note this attribute only applies if a disk store is configured for this region.
      ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="disk-store-ref" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates the id of the disk store to use for persistence or overflow.

Note this attribute only applies if a disk store is configured for this region.
      ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="destroy" type="xsd:string"
                    default="false">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether the defined region should be destroyed or not at shutdown. Destroy cascades to all entries
After the destroy, this region object can not be used any more and any attempt to use this region object wi
RegionDestroyedException.

Default is false, meaning that regions are not destroyed.

Note: destroy and close are mutually exclusive. Enabling one will automatically disable the other.
      ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="close" type="xsd:string"
                    default="true">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether the defined region should be closed or not at shutdown. Close performs a local destroy bu
disk files. Additionally it notifies the listeners and callbacks.

Default is true, meaning the regions are closed.

Note: Regions are automatically closed when cache closes.
Note: destroy and close are mutually exclusive. Enabling one will automatically disable the other.
      ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="statistics" type="xsd:string"
                    default="false">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether statistics are enabled or disabled for this region and its entries.
Default is false, meaning statistics are disabled.
      ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="key-constraint" type="xsd:string"
                    use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The fully qualified class name of the expected key type
        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
```

```
                <xsd:attribute name="value-constraint"
                    type="xsd:string" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The fully qualified class name of the expected value type
        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="ignore-jta" type="xsd:string"
                    use="optional" default="false">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
 Indicates whether operations on this region participates in active JTA transactions or ignores them and ope
 This is primarily used in cache loaders, writers, and listeners that need to perform non-transactional oper
 such as caching a result set.
        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="initial-capacity"
                    type="xsd:string" use="optional" default="16">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
 Sets the initial capacity (number of entries) for the region
        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="index-update-type"
                    use="optional" default="synchronous">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
 Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously
        ]]></xsd:documentation>
                    </xsd:annotation>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="asynchronous" />
                            <xsd:enumeration value="synchronous" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="enable-gateway" type="xsd:string"
                    use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
 Specifies if WAN gateway communications are enabled for this region (true or false)
                                ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="hub-id" type="xsd:string"
                    use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
 Specifies if WAN gateway hub id if enable-gateway is true.
                                ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="readOnlyRegionType">
        <xsd:complexContent>
```

```
            <xsd:extension base="baseReadOnlyRegionType">
                <xsd:attributeGroup ref="topLevelRegionAttributes" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="readOnlySubRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseReadOnlyRegionType">
                <xsd:attribute name="name" type="xsd:string"
                    use="required">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the region definition.]]>
</xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="baseRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseReadOnlyRegionType">
                <xsd:sequence minOccurs="0" maxOccurs="1">
                    <xsd:element name="cache-loader" type="beanDeclarationType"
                        minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation
                                source="com.gemstone.gemfire.cache.CacheLoader"><![CDATA[
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.cache.CacheLoader" />
                                </tool:annotation>
                            </xsd:appinfo>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="cache-writer" type="beanDeclarationType"
                        minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation
                                source="com.gemstone.gemfire.cache.CacheWriter"><![CDATA[
The cache writer definition for this region. A cache writer acts as a dedicated synchronous listener that is
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
arbitrarily pick one.
       ]]></xsd:documentation>
                            <xsd:appinfo>
                                <tool:annotation>
                                    <tool:exports
                                        type="com.gemstone.gemfire.cache.CacheWriter" />
                                </tool:annotation>
                            </xsd:appinfo>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="region-ttl" type="expirationType"
                        minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
```

```
                            <xsd:documentation><![CDATA[
Time to live configuration for the region itself. Default: no expiration.
       ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="region-tti" type="expirationType"
                        minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Time to idle (or idle timeout) configuration for the region itself. Default: no expiration.
       ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="entry-ttl" type="expirationType"
                        minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Time to live configuration for the region entries. Default: no expiration.
       ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="entry-tti" type="expirationType"
                        minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Time to idle (or idle timeout) configuration for the region entries. Default: no expiration.
       ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="load-factor" type="xsd:string"
                    default="0.75">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Together with the initial-capacity region attribute, sets the initial parameters on the underlying java.uti
used for storing region entries. This must be a floating point number between 0 and 1, inclusive.
       ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="cloning-enabled"
                    type="xsd:string" default="true">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Determines how fromDelta applies deltas to the local cache for delta propagation. When true, the updates ar
clone of the value and then the clone is saved to the cache. When false, the value is modified in place in
       ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="regionType">
        <xsd:complexContent>
            <xsd:extension base="baseRegionType">
                <xsd:attributeGroup ref="topLevelRegionAttributes" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="subRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseRegionType">
```

```
                <xsd:attribute name="name" type="xsd:string"
                    use="required">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
                The name of the region definition.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="lookupRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseLookupRegionType">
                <xsd:attributeGroup ref="topLevelRegionAttributes" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="lookupSubRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseLookupRegionType">
                <xsd:attribute name="name" type="xsd:string"
                    use="required">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the region definition.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:group name="subRegionGroup">
        <xsd:choice>
            <xsd:element name="lookup-region" type="lookupSubRegionType" />
            <xsd:element name="replicated-region" type="replicatedSubRegionType" />
            <xsd:element name="partitioned-region" type="partitionedSubRegionType" />
            <xsd:element name="local-region" type="localSubRegionType" />
        </xsd:choice>
    </xsd:group>
    <!-- -->
    <xsd:attributeGroup name="topLevelRegionAttributes">
        <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="cache-ref" type="xsd:string"
            use="optional" default="gemfireCache">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The name of the bean defining the GemFire cache (by default 'gemfireCache').
    ]]></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
Required for subregions.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:attributeGroup>
    <!-- -->
    <xsd:attributeGroup name="distributedRegionAttributes">
        <xsd:attribute name="enable-subscription-conflation"
            type="xsd:string" default="false">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates whether the region can conflate its messages to the client.
                                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="enable-async-conflation"
            type="xsd:string" default="false">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
 For TCP/IP distributions between peers, specifies whether to allow aggregation of asynchronous messages ser
 This is a special-purpose boolean attribute that applies only when asynchronous queues are used for slow co
                                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="multicast-enabled" type="xsd:string">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Boolean that indicates whether distributed operations on a region should use multicasting. To enable this, m
distributed system with the mcast-port gemfire.properties setting.
                                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:attributeGroup>
    <!-- -->
    <xsd:element name="lookup-region" type="lookupRegionType" />
    <!-- -->
    <xsd:complexType name="baseReplicatedRegionType">
        <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:complexContent>
            <xsd:extension base="baseRegionType">
                <xsd:sequence minOccurs="1" maxOccurs="1">
                    <xsd:element name="eviction" minOccurs="0"
                        maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Eviction policy for the replicated region.
                                ]]></xsd:documentation>
                        </xsd:annotation>
```

```
                        <xsd:complexType>
                            <xsd:complexContent>
                                <xsd:extension base="evictionType">
                                    <xsd:attribute name="action"
                                        type="evictionActionType" fixed="OVERFLOW_TO_DISK">
                                        <xsd:annotation>
                                            <xsd:documentation><![CDATA[
The action to take when performing eviction.
                                                ]]></xsd:documentation>
                                        </xsd:annotation>
                                    </xsd:attribute>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:group ref="subRegionGroup" minOccurs="0"
                        maxOccurs="unbounded" />
                </xsd:sequence>
                <xsd:attributeGroup ref="distributedRegionAttributes" />
                <xsd:attribute name="concurrency-level">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Provides an estimate of the maximum number of application threads that will concurrently access a region en
This attribute does not apply to partitioned regions. This attribute helps GemFire optimize the use of syst
reduce thread contention. This sets an initial parameter on the underlying java.util.ConcurrentHashMap used
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="scope" type="xsd:string">
                   <xsd:annotation>
                        <xsd:documentation><![CDATA[
Specifies the scope for this region: distributed-ack,distributed-no-ack, global
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="data-policy" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Specifies the data policy for this region
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="is-lock-grantor"
                    type="xsd:string" use="optional" default="false">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether the region is a lock grantor.This attribute is only relevant for regions with global scop
                                ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="replicatedRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseReplicatedRegionType">
                <xsd:attributeGroup ref="topLevelRegionAttributes" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
```

```
    <xsd:complexType name="replicatedSubRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseReplicatedRegionType">
                <xsd:attribute name="name" type="xsd:string"
                    use="required">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the region definition.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:element name="replicated-region" type="replicatedRegionType" />
    <!-- -->
    <xsd:complexType name="baseLocalRegionType">
        <xsd:annotation>
            <xsd:documentation
                source="org.springframework.data.gemfire.ReplicatedRegionFactoryBean"><![CDATA[
Defines a GemFire local region instance. Each local region is scoped only to the local JVM.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="com.gemstone.gemfire.cache.Region" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="baseRegionType">
                <xsd:sequence minOccurs="1" maxOccurs="1">
                    <xsd:element name="eviction" minOccurs="0"
                        maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Eviction policy for the replicated region.
                                ]]></xsd:documentation>
                        </xsd:annotation>
                        <xsd:complexType>
                            <xsd:complexContent>
                                <xsd:extension base="evictionType">
                                    <xsd:attribute name="action"
                                        type="evictionActionType" fixed="OVERFLOW_TO_DISK">
                                        <xsd:annotation>
                                            <xsd:documentation><![CDATA[
The action to take when performing eviction.
                                                ]]></xsd:documentation>
                                        </xsd:annotation>
                                    </xsd:attribute>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:group ref="subRegionGroup" minOccurs="0"
                        maxOccurs="unbounded" />
                </xsd:sequence>
                <xsd:attribute name="data-policy" type="xsd:string"
                    default="NORMAL">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates the DataPolicy to use for this region (NORMAL or PRELOADED)
```

```
      ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="concurrency-level">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Provides an estimate of the maximum number of application threads that will concurrently access a region en
This attribute does not apply to partitioned regions. This attribute helps GemFire optimize the use of syst
reduce thread contention. This sets an initial parameter on the underlying java.util.ConcurrentHashMap used
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="localRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseLocalRegionType">
                <xsd:attributeGroup ref="topLevelRegionAttributes" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="localSubRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseLocalRegionType">
                <xsd:attribute name="name" type="xsd:string"
                    use="required">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the region definition.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:element name="local-region" type="localRegionType" />
    <!-- -->
    <xsd:complexType name="basePartitionedRegionType">
        <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:complexContent>
            <xsd:extension base="baseRegionType">
                <xsd:sequence>
                    <xsd:element name="partition-resolver"
                        type="beanDeclarationType" minOccurs="0"
                        maxOccurs="1">
```

```
                        <xsd:annotation>
                            <xsd:documentation
                                source="com.gemstone.gemfire.cache.PartitionResolver"><![CDATA[
The partition resolver definition for this region, allowing for custom partitioning. GemFire uses the resol
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.gemfire.cache.PartitionResolver" />
                                </tool:annotation>
                            </xsd:appinfo>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="partition-listener"
                        type="beanDeclarationType" minOccurs="0"
                        maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation
                                source="com.gemstone.gemfire.cache.partition.PartitionListener"><![CDATA[
The partition listener definition for this region. Defines a callback for partitioned regions, invoked when
is created or any bucket in a partitioned region becomes primary
                            ]]></xsd:documentation>
                            <xsd:appinfo>
                                <tool:annotation>
                                    <tool:exports
                                        type="com.gemstone.gemfire.cache.partition.PartitionListener" />
                                </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"
                                        type="evictionActionType"
                                        default="LOCAL_DESTROY">
                                        <xsd:annotation>
                                            <xsd:documentation><![CDATA[
The action to take when performing eviction.
            ]]></xsd:documentation>
                                        </xsd:annotation>
                                    </xsd:attribute>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attributeGroup ref="distributedRegionAttributes" />
                <xsd:attribute name="copies" use="optional"
                    default="0">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The number of copies for each partition for high-availability. By default, no copies are created meaning th
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:string" use="optional">
                    <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 available heap is used.
       ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="total-max-memory"
                    type="xsd:string" use="optional">
                    <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:string"
                    use="optional">
                    <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
 move from one VM to another. Buckets may also have copies, depending on redundancy to provide high availabi
  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 V
, 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:string"
                    use="optional" default="-1">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The delay in milliseconds that existing members will wait before satisfying redundancy after another member
-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:string" use="optional" default="-1">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The delay in milliseconds that new members will wait before satisfying redundancy. -1 indicates that adding
will not trigger redundancy recovery. The default is to recover redundancy immediately when a new member is
        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="partitionedRegionType">
        <xsd:complexContent>
            <xsd:extension base="basePartitionedRegionType">
                <xsd:attributeGroup ref="topLevelRegionAttributes" />
            </xsd:extension>
            <!-- subRegions not supported -->
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="partitionedSubRegionType">
        <xsd:complexContent>
            <xsd:extension base="basePartitionedRegionType">
                <xsd:attribute name="name" type="xsd:string"
                    use="required">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the region definition.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
            <!-- subRegions not supported -->
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:element name="partitioned-region" type="partitionedRegionType" />
    <!-- -->
    <xsd:complexType name="expirationType">
        <xsd:attribute name="timeout" type="xsd:string"
            default="0">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The amount of time before the expiration action takes place. Defaults to zero (which means never timeout).
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="action" default="INVALIDATE">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="INVALIDATE">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
When the region or cached object expires, it is invalidated.
       ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:enumeration>
                    <xsd:enumeration value="DESTROY">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
When the region or cached object expires, it is destroyed.
```

```
       ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:enumeration>
                    <xsd:enumeration value="LOCAL_INVALIDATE">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
When the region or cached object expires, it is invalidated locally only. Not supported on partitioned regio
       ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:enumeration>
                    <xsd:enumeration value="LOCAL_DESTROY">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
When the region or cached object expires, it is destroyed locally only. Not supported on partitioned regions
       ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:enumeration>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <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.ObjectSizer" />
                        </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:string"
            use="required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The threshold (or limit) against which the eviction algorithm runs. Once the threashold is reached, evictio
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="baseDiskStoreType">
        <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:appinfo>
                                <tool:annotation>
                                    <tool:exports
                                        type="com.gemstone.gemfire.cache.DiskStore" />
                                </tool:annotation>
                            </xsd:appinfo>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="max-size" type="xsd:string"
                        default="2147483647">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
The maximum size (in megabytes) of data stored in each directory. Default value is 2,147,483,647 which is tw
```

```
       ]]></xsd:documentation>
                       </xsd:annotation>
                   </xsd:attribute>
               </xsd:complexType>
           </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="auto-compact" type="xsd:string"
            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="max-oplog-size" type="xsd:string"
            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 create
amount of file space will be immediately reserved.
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="time-interval" type="xsd:string"
            default="1000">
            <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:string"
            default="0">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The maximum number of operations that can be asynchronously queued. Once this many pending async operations
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: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 wil
The threshold is a percentage in the range 0..100. When the amount of garbage in an oplog exceeds this perc
compaction is done this garbage will be cleaned up freeing up disk space. Garbage is created by entry destr
entry updates, and region destroys.
]]></xsd:documentation>
            </xsd:annotation>
            <xsd:simpleType>
                <xsd:restriction base="xsd:short">
                    <xsd:minInclusive value="0" />
                    <xsd:maxInclusive value="100" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="allow-force-compaction" type="xsd:string"
            default="false">
            <xsd:annotation>
```

```
                <xsd:documentation><![CDATA[
Indicates whether forced compaction is allowed for regions using this disk store
     ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="write-buffer-size" type="xsd:string"
            default="32768">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates the write buffer size in bytes
     ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="diskStoreType">
        <xsd:complexContent>
            <xsd:extension base="baseDiskStoreType">
                <xsd:attribute name="id" type="xsd:string"
                    use="required">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the disk store bean definition. This is also used as the disk store name]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="cache-ref" type="xsd:string"
                    use="optional" default="gemfireCache">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the bean defining the GemFire cache (by default 'gemfireCache').
      ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:element name="disk-store" type="diskStoreType" />
    <!-- -->
    <xsd:element name="client-region">
        <xsd:annotation>
            <xsd:documentation
                source="org.springframework.data.gemfire.client.ClientRegionFactoryBean"><![CDATA[
Defines a GemFire client region instance. A client region is connected to a (long-lived) farm of GemFire se
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
special token 'ALL_KEYS', which will register interest in all keys in the region. In effect, this will cause
```

```
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="interestType">
                                            <xsd:sequence
                                                minOccurs="0" maxOccurs="1">
                                                <xsd:any
                                                    namespace="##other"
                                                    processContents="skip"
                                                    minOccurs="0"
                                                    maxOccurs="unbounded">
                                                    <xsd:annotation>
                                                        <xsd:documentation><![CDATA[
Inner bean definition of the client key interest.
              ]]></xsd:documentation>
                                                    </xsd:annotation>
                                                </xsd:any>
                                            </xsd:sequence>
                                            <xsd:attribute
                                                name="key-ref" type="xsd:string"
                                                use="optional">
                                                <xsd:annotation>
                                                    <xsd:documentation><![CDATA[
The name of the client key interest bean referred by this declaration. Used as a convenience method. If no
use the inner bean declaration.
             ]]></xsd:documentation>
                                                </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 c
         ]]></xsd:documentation>
                                </xsd:annotation>
                                <xsd:complexType>
                                    <xsd:complexContent>
                                        <xsd:extension
                                            base="interestType">
                                            <xsd:attribute
                                                name="pattern" type="xsd:string" />
                                        </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" type="evictionActionType"
                                        default="LOCAL_DESTROY">
                                        <xsd:annotation>
                                            <xsd:documentation><![CDATA[
The action to take when performing eviction.
            ]]></xsd:documentation>
                                        </xsd:annotation>
                                    </xsd:attribute>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="data-policy"
                    type="xsd:string" 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 ov
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
 footprint producers that only want to distribute their data to others and for zero footprint consumers tha
 to see events.
NORMAL - causes data that this region is interested in to be stored in local memory. It allows the contents
cache to differ from other caches.
        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="pool-name" type="xsd:string"
                    use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The name of the pool used by this client. If not set, a default pool (initialized when using client-cache) w
        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="shortcut" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The ClientRegionShortcut for this region. Allows easy initialization of the region based on defaults.
        ]]></xsd:documentation>
                    </xsd:annotation>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="PROXY" />
                            <xsd:enumeration value="CACHING_PROXY" />
                            <xsd:enumeration
                                value="CACHING_PROXY_HEAP_LRU" />
                            <xsd:enumeration
                                value="CACHING_PROXY_OVERFLOW" />
                            <xsd:enumeration value="LOCAL" />
                            <xsd:enumeration value="LOCAL_PERSISTENT" />
                            <xsd:enumeration value="LOCAL_HEAP_LRU" />
                            <xsd:enumeration value="LOCAL_OVERFLOW" />
                            <xsd:enumeration
                                value="LOCAL_PERSISTENT_OVERFLOW" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </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:string" />
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="interestType" abstract="true">
        <xsd:attribute name="durable" type="xsd:string"
            use="optional" default="false">
            <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" use="optional"
            default="KEYS_VALUES">
            <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:attribute name="receive-values" type="xsd:string"
            use="optional" default="true">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates whether values are received with create and update events on keys of interest (true)
or only invalidations are received and the value will be received on the next get instead (false).
Default is true.
    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
```

```
    </xsd:complexType>
    <!-- -->
    <xsd:element name="pool">
        <xsd:annotation>
            <xsd:documentation
                source="org.springframework.data.gemfire.client.PoolFactoryBean"><![CDATA[
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="unbounded" />
                <xsd:element name="server" type="connectionType"
                    minOccurs="1" maxOccurs="unbounded" />
            </xsd:choice>
            <xsd:attribute name="id" type="xsd:string"
                use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the pool definition (by default "gemfirePool").]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="free-connection-timeout"
                type="xsd:string" use="optional" />
            <xsd:attribute name="idle-timeout" type="xsd:string"
                use="optional" />
            <xsd:attribute name="load-conditioning-interval"
                type="xsd:string" use="optional" />
            <xsd:attribute name="max-connections" type="xsd:string"
                use="optional" />
            <xsd:attribute name="min-connections" type="xsd:string"
                use="optional" />
            <xsd:attribute name="multi-user-authentication"
                type="xsd:string" use="optional" />
            <xsd:attribute name="ping-interval" type="xsd:string"
                use="optional" />
            <xsd:attribute name="pr-single-hop-enabled"
                type="xsd:string" use="optional" />
            <xsd:attribute name="read-timeout" type="xsd:string"
                use="optional" />
            <xsd:attribute name="retry-attempts" type="xsd:string"
                use="optional" />
            <xsd:attribute name="server-group" type="xsd:string"
                use="optional" />
            <xsd:attribute name="socket-buffer-size" type="xsd:string"
                use="optional" />
            <xsd:attribute name="statistic-interval" type="xsd:string"
                use="optional" />
            <xsd:attribute name="subscription-ack-interval"
                type="xsd:string" use="optional" />
            <xsd:attribute name="subscription-enabled"
                type="xsd:string" use="optional" />
            <xsd:attribute name="subscription-message-tracking-timeout"
```

```
                type="xsd:string" use="optional" />
            <xsd:attribute name="subscription-redundancy"
                type="xsd:string" use="optional" />
            <xsd:attribute name="thread-local-connections"
                type="xsd:string" use="optional" />
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:element name="cache-server">
        <xsd:annotation>
            <xsd:documentation
                source="org.springframework.data.gemfire.server.CacheServerFactoryBean"><![CDATA[
Defines a Cache Server for feeding data to remote gemfire clients to a server GemFire Cache Servers.
Note: In order to instantiate a cacheserver, a GemFire cache needs to be avaialble in the VM.
   ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports
                        type="com.gemstone.gemfire.cache.server.CacheServer" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="1">
                <xsd:element name="subscription-config"
                    minOccurs="0" maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The client subscription configuration that is used to control a clients use of server resources towards noti
    ]]></xsd:documentation>
                    </xsd:annotation>
                    <xsd:complexType>
                        <xsd:attribute name="eviction-type"
                            use="optional" default="NONE">
                            <xsd:simpleType>
                                <xsd:restriction base="xsd:string">
                                    <xsd:enumeration
                                        value="NONE" />
                                    <xsd:enumeration
                                        value="MEM" />
                                    <xsd:enumeration
                                        value="ENTRY" />
                                </xsd:restriction>
                            </xsd:simpleType>
                        </xsd:attribute>
                        <xsd:attribute name="capacity"
                            type="xsd:string" use="optional" default="1" />
                        <xsd:attribute name="disk-store"
                            type="xsd:string" use="optional" default="." />
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:string"
                use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the cache server definition (by default "gemfireServer").
    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="auto-startup" type="xsd:string"
                use="optional" default="true" />
```

```
            <xsd:attribute name="bind-address" type="xsd:string"
                use="optional" />
            <xsd:attribute name="port" type="xsd:string"
                use="optional" default="40404">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The port number of the server.
     ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="host-name-for-clients"
                type="xsd:string" use="optional" />
            <xsd:attribute name="load-poll-interval" type="xsd:string"
                use="optional" default="5000" />
            <xsd:attribute name="max-connections" type="xsd:string"
                use="optional" default="800" />
            <xsd:attribute name="max-threads" type="xsd:string"
                use="optional" default="0" />
            <xsd:attribute name="max-message-count" type="xsd:string"
                use="optional" default="230000" />
            <xsd:attribute name="max-time-between-pings"
                type="xsd:string" use="optional" default="60000" />
            <xsd:attribute name="message-time-to-live"
                type="xsd:string" use="optional" default="180" />
            <xsd:attribute name="socket-buffer-size" type="xsd:string"
                use="optional" default="32768" />
            <xsd:attribute name="notify-by-subscription"
                type="xsd:string" use="optional" default="true" />
            <xsd:attribute name="groups" type="xsd:string"
                use="optional" default="">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The server groups that this server will be a member of given as a comma separated values list.
    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="load-probe-ref" type="xsd:string"
                use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the bean defining the CacheServer Load Probe.
     ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="cache-ref" type="xsd:string"
                use="optional" default="gemfireCache">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the bean defining the GemFire cache (by default 'gemfireCache').
     ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:element name="cq-listener-container">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
Container for continuous query listeners. All listeners will be hosted by the same container.
       ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
```

```
                    <tool:exports
                        type="org.springframework.data.gemfire.listener.ContinuousQueryListenerContainer" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="listener" type="listenerType"
                    minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:attribute name="cache" type="xsd:string"
                default="gemfireCache">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
A reference (by name) to the GemFire cache bean. Default is "gemfireCache".
           ]]></xsd:documentation>
                    <xsd:appinfo>
                        <tool:annotation kind="ref">
                            <tool:expected-type
                                type="com.gemstone.gemfire.cache.RegionService" />
                        </tool:annotation>
                    </xsd:appinfo>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="task-executor" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
A reference to a Spring TaskExecutor (or standard JDK 1.5 Executor) for executing
GemFire listener invokers. Default is a SimpleAsyncTaskExecutor.
           ]]></xsd:documentation>
                    <xsd:appinfo>
                        <tool:annotation kind="ref">
                            <tool:expected-type
                                type="java.util.concurrent.Executor" />
                        </tool:annotation>
                    </xsd:appinfo>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="phase" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The lifecycle phase within which this container should start and stop. The lower
the value the earlier this container will start and the later it will stop. The
default is Integer.MAX_VALUE meaning the container will start as late as possible
and stop as soon as possible.
           ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="pool-name" type="xsd:string"
                use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the pool used by the container.
       ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:complexType name="listenerType">
        <xsd:attribute name="ref" type="xsd:string" use="required">
            <xsd:annotation>
```

```
                <xsd:documentation><![CDATA[
The bean name of the listener object, implementing the ContinuousQueryListener interface or defining the spe
Required.
         ]]></xsd:documentation>
                <xsd:appinfo>
                    <tool:annotation kind="ref" />
                </xsd:appinfo>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="query" type="xsd:string" use="required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The query for the GemFire continuous query.
         ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="method" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The name of the listener method to invoke. If not specified, the target bean is supposed to implement the Co
interface or provide a method named 'handleEvent'.
         ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="name" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The name of the resulting GemFire continuous query. Useful for monitoring and statistics querying.
         ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="durable" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Whether the resulting GemFire continuous query is durable or not.
         ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:element name="index">
        <xsd:annotation>
            <xsd:documentation
                source="org.springframework.data.gemfire.IndexFactoryBean"><![CDATA[
Defines a GemFire index.
   ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports
                        type="com.gemstone.gemfire.cache.query.Index" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:attribute name="id" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the index bean definition. If property 'name' is not set, it will be used as the index name as w
     ]]></xsd:documentation>
                </xsd:annotation>
```

```
            </xsd:attribute>
            <xsd:attribute name="type" use="optional"
                default="FUNCTIONAL">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="FUNCTIONAL" />
                        <xsd:enumeration value="PRIMARY_KEY" />
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="name" type="xsd:string"
                use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the index.
     ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="expression" type="xsd:string"
                use="required" />
            <xsd:attribute name="from" type="xsd:string"
                use="required" />
            <xsd:attribute name="imports" type="xsd:string"
                use="optional" />
            <xsd:attribute name="override" type="xsd:string"
                use="optional" default="true">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Indicates whether the index is created even if there is an index with the same name (default) or not.
     ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="cache-ref" type="xsd:string"
                use="optional" default="gemfireCache">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the bean defining the GemFire cache (by default 'gemfireCache').
     ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="pool-name" type="xsd:string"
                use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the pool used by the index. Used usually in client scenarios.
     ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:complexType name="jndiBindingType">
        <xsd:sequence>
            <xsd:element name="jndi-prop" type="configPropertyType"
                minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Specifies a vendor-specific property
            ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
```

```
        <xsd:attribute name="jndi-name" type="xsd:string"
            use="required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The JNDI name for this datasource. Will be prefixed with "java:/"
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="type" use="required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the datasource implementation: ManagedDataSource,SimpleDataSource,PooledDataSource,XaPooledDataSou
            ]]></xsd:documentation>
            </xsd:annotation>
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="ManagedDataSource" />
                    <xsd:enumeration value="SimpleDataSource" />
                    <xsd:enumeration value="PooledDataSource" />
                    <xsd:enumeration value="XaPooledDataSource" />
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="blocking-timeout-seconds"
            type="xsd:string" use="optional" />
        <xsd:attribute name="conn-pooled-datasource-class"
            type="xsd:string" use="optional" />
        <xsd:attribute name="connection-url" type="xsd:string"
            use="optional" />
        <xsd:attribute name="idle-timeout-seconds" type="xsd:string"
            use="optional" />
        <xsd:attribute name="init-pool-size" type="xsd:string"
            use="optional" />
        <xsd:attribute name="jdbc-driver-class" type="xsd:string"
            use="optional" />
        <xsd:attribute name="login-timeout-seconds" type="xsd:string"
            use="optional" />
        <xsd:attribute name="managed-connection-factory-class"
            type="xsd:string" use="optional" />
        <xsd:attribute name="max-pool-size" type="xsd:string"
            use="optional" />
        <xsd:attribute name="password" type="xsd:string"
            use="optional" />
        <xsd:attribute name="user-name" type="xsd:string"
            use="optional" />
        <xsd:attribute name="xa-datasource-class" type="xsd:string"
            use="optional" />
        <xsd:attribute name="transaction-type" type="xsd:string"
            use="optional" />
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="configPropertyType" mixed="true">
        <xsd:attribute name="key" type="xsd:string" use="required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies The property key
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="type" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
```

```
Specifies a data type if other than java.lang.String
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <!-- -->
    <xsd:attributeGroup name="commonWANQueueAttributes">
        <xsd:attribute name="batch-size" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the batch size
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="persistent" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies whether persistence is enabled: true or false(default)
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="disk-store-ref" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates the id of disk store to use for persistence
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="maximum-queue-memory" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the maximum memory in MB to allocate for the queue
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:attributeGroup>
    <!-- -->
    <xsd:complexType name="gatewayReceiverType">
        <xsd:annotation>
            <xsd:documentation
                source="com.gemstone.gemfire.cache.wan.GatewayReceiver"><![CDATA[
A gateway receiver definition (requires Gemfire 7.0 or later)
           ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports
                        type="com.gemstone.gemfire.cache.wan.GatewayReceiver" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="transport-filter" type="gatewayTransportFilterType"
                minOccurs="0" maxOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="start-port" type="xsd:string"
            use="optional">
            <xsd:annotation>
```

```
                <xsd:documentation><![CDATA[
Specifies the lower end of a port range to use for the gateway receiver
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="end-port" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the upper end of a port range to use for the gateway receiver
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="bind-address" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the bind address (IP address or host name) for the gateway receiver
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="maximum-time-between-pings"
            type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the maximum time between pings in milliseconds
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="socket-buffer-size" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the socket buffer size in bytes
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="id" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The id of this bean definition
                    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="cache-ref" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The id of the cache - default is gemfireCache
                  ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:element name="function-service">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="function" minOccurs="0"
                    maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation
                            source="com.gemstone.gemfire.cache.execute.Function"><![CDATA[
```

```
Declares one or more remote functions for this cache and register's with them the FunctionService. each bean
must implement com.gemstone.gemfire.cache.execute.Function
                        ]]></xsd:documentation>
                    <xsd:appinfo>
                        <tool:annotation>
                            <tool:exports
                                type="com.gemstone.gemfire.cache.execute.Function" />
                        </tool:annotation>
                    </xsd:appinfo>
                </xsd:annotation>
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:any namespace="##other"
                            processContents="skip" minOccurs="0"
                            maxOccurs="unbounded">
                            <xsd:annotation>
                                <xsd:documentation><![CDATA[
Inner bean definition of the remote function.
                                    ]]></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 remote function bean referred by this declaration. Used as a convenience method. If no refer
use inner bean declarations.
                                ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" use="optional">
                    <xsd:annotation>
                            <xsd:documentation><![CDATA[
The id of the function service (optional)
                                ]]></xsd:documentation>
                        </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
</xsd:element>
<!-- -->
<xsd:simpleType name="scopeType">
    <xsd:annotation>
        <xsd:documentation><![CDATA[
Determines how updates to region entries are distributed to the other caches in the distributed system where
Scope also determines whether to allow remote invocation of some of the region’s event handlers
]]></xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="distributed-ack" />
        <xsd:enumeration value="distributed-no-ack" />
        <xsd:enumeration value="global" />
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="gatewayTransportFilterType">
    <xsd:annotation>
        <xsd:documentation
            source="com.gemstone.gemfire.cache.wan.GatewayTransportFilter"><![CDATA[
```

```
A transport filter for this gateway component
                            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports
                        type="com.gemstone.gemfire.cache.wan.GatewayTransportFilter" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any namespace="##other" processContents="skip"
                minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Inner bean definition of the transport filter.
                                        ]]></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 transport filter bean referred by this declaration. Used as a convenience method. If no ref
use inner bean declarations.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- Gemfire 6 WAN Gateway schema -->
    <xsd:complexType name="gatewayHubType">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
]]></xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="gateway" type="gatewayType"
                minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
]]></xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" use="required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The id of this hub
                    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="port" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The port for this hub (integer value, if not specified, Gemfire will select an open port)
                    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="cache-ref" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The id of the cache - default is gemfireCache
```

```
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="bind-address" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the bind address (IP address or host name) for the gateway hub
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="socket-buffer-size" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the socket buffer size in bytes
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="manual-start" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies if the gateway hub is manually (true) or automatically(false) started
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="startup-policy" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the startup policy (primary,secondary, none) for the gateway hub
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="gatewayEndpointType">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
]]></xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="host" type="xsd:string" use="required">
        </xsd:attribute>
        <xsd:attribute name="port" type="xsd:string" use="required">
        </xsd:attribute>
        <xsd:attribute name="endpoint-id" type="xsd:string"
            use="required">
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="gatewayQueueType">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
]]></xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="enable-batch-conflation"
            type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies whether batch conflation is enabled (true or false)
            ]]></xsd:documentation>
```

```
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="batch-time-interval" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The maximum time interval that can elapse before a partial batch is sent from a GatewaySender to its corresp
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="alert-threshold" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the alert threshold in miliseconds, indicating the maximum time elapsed from when the gateway sent
to when the acknowldgement was received from the gateway receiver.
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="batch-size" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the batch size
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="persistent" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies whether persistence is enabled: true or false(default)
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="disk-store-ref" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates the id of disk store to use for persistence
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="maximum-queue-memory" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the maximum memory in MB to allocate for the queue
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="gatewayType">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
]]></xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="gateway-endpoint"
                    minOccurs="1" maxOccurs="unbounded" type="gatewayEndpointType" />
```

```
                <xsd:element name="gateway-listener"
                    minOccurs="1" maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation
                            source="com.gemstone.gemfire.cache.util.GatewayEventListener"><![CDATA[
An gateway event listener definition for the gateway
                        ]]></xsd:documentation>
                        <xsd:appinfo>
                            <tool:annotation>
                                <tool:exports
                                    type="com.gemstone.gemfire.cache.util.GatewayEventListener" />
                            </tool:annotation>
                        </xsd:appinfo>
                    </xsd:annotation>
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:any namespace="##other"
                                processContents="skip" minOccurs="0"
                                maxOccurs="unbounded">
                                <xsd:annotation>
                                    <xsd:documentation><![CDATA[
Inner bean definition of the gateway event listener
                                        ]]></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 gateway event listener bean referred by this declaration. Used as a convenience method. If n
use inner bean declarations.
                                    ]]></xsd:documentation>
                            </xsd:annotation>
                        </xsd:attribute>
                    </xsd:complexType>
                </xsd:element>
            </xsd:choice>
            <xsd:element name="gateway-queue" minOccurs="0"
                maxOccurs="1" type="gatewayQueueType" />
        </xsd:sequence>
        <xsd:attribute name="gateway-id" type="xsd:string"
            use="required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the id for this gateway
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="socket-buffer-size" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the socket buffer size in bytes
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="socket-read-timeout" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the socket read timeout in milliseconds
```

```
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="order-policy" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the order policy - This only applies if parallel is enabled:
KEY: Indicates that events will be parallelized based on the event's key,
PARTITION:Indicates that events will be parallelized based on the event's: partition (using the PartitionRes
THREAD:Indicates that events will be parallelized based on the event's originating member and thread
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="concurrency-level" type="xsd:string"
            use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the number of parallel threads
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:element name="gateway-hub" type="gatewayHubType">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
]]></xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <!-- End Gemfire 6 WAN Gateway schema -->
</xsd:schema>
```

Spring Data GemFire Data Access Schema (gfe-data)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.springframework.org/schema/data/gemfire" xmlns:xsd="http://www.w3.org/2001/XM
xmlns:tool="http://www.springframework.org/schema/tool"
xmlns:repository="http://www.springframework.org/schema/data/repository"
targetNamespace="http://www.springframework.org/schema/data/gemfire" elementFormDefault="qualified" attribu
 <xsd:import namespace="http://www.springframework.org/schema/beans"/>
 <xsd:import namespace="http://www.springframework.org/schema/tool"/>
 <xsd:import namespace="http://www.springframework.org/schema/data/repository" schemaLocation="http://www.sp
    <xsd:import namespace="http://www.springframework.org/schema/gemfire" schemaLocation="http://www.springf
 <!-- -->
 <xsd:annotation>
  <xsd:documentation><![CDATA[
            Namespace support for the Spring Data GemFire Repositories.
        ]]></xsd:documentation>
 </xsd:annotation>
 <!-- -->
 <!-- Repositories -->
 <xsd:element name="repositories">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:extension base="repository:repositories">
     <xsd:attributeGroup ref="gemfire-repository-attributes"/>
     <xsd:attributeGroup ref="repository:repository-attributes"/>
    </xsd:extension>
   </xsd:complexContent>
```

```
  </xsd:complexType>
 </xsd:element>
 <!-- -->
 <xsd:attributeGroup name="gemfire-repository-attributes">
  <xsd:attribute name="mapping-context-ref" type="mappingContextRef">
   <xsd:annotation>
    <xsd:documentation>
                    The reference to a MappingContext. If not set a default one will be created.
                </xsd:documentation>
   </xsd:annotation>
  </xsd:attribute>
 </xsd:attributeGroup>
 <!-- -->
 <xsd:simpleType name="mappingContextRef">
  <xsd:annotation>
   <xsd:appinfo>
    <tool:annotation kind="ref">
     <tool:assignable-to type="org.springframework.data.gemfire.GemfireMappingContext"/>
    </tool:annotation>
   </xsd:appinfo>
  </xsd:annotation>
  <xsd:union memberTypes="xsd:string"/>
 </xsd:simpleType>
 <!-- -->
</xsd:schema>
```