Spring Data GemFire Reference Guide

Costin Leau , David Turanski , John Blum , Oliver Gierke

Version 2.0.0.M3, 2017-05-09

Table of Contents

Preface

Spring Data GemFire focuses on integrating the *Spring Framework's* powerful, non-invasive programming model and concepts with Pivotal GemFire to simplify configuration and development of Java applications using GemFire.

This document assumes the reader already has a basic familiarity with the *Spring Framework* and Pivotal GemFire concepts and APIs.

While every effort has been made to ensure this documentation is comprehensive and complete, with no errors, some topics are beyond the scope of this document and may require more explanation (e.g. data distribution management with partitioning for HA while still preserving consistency). Additionally, some typos might have crept in. If you do spot mistakes or even more serious errors and you can spare a few cycles, please do bring these issues to the attention of the *Spring Data GemFire* team by raising an appropriate issue.

Thank you.

Chapter 1. Introduction

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

Chapter 2. Requirements

Spring Data GemFire requires JDK 8.0, Spring Framework 5 and Pivotal GemFire 9.0.x.

Chapter 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 module of the Spring Data project.

3.1. New in the 1.2 Release

- Full support for GemFire configuration via the SDG **gfe** namespace. Now GemFire components may be configured completely without requiring a native **cache.xml** file.
- WAN Gateway support for GemFire 6.6.x. See Configuring WAN Gateways.
- Spring Data Repository support using a dedicated SDG namespace, **gfe-data**. See Spring Data GemFire Repositories
- Namespace support for registering GemFire Functions. See Configuring the Function Service
- A top-level `<disk-store>` element has been added to the SDG **gfe** namespace to allow sharing of persist stores among Regions, and other components that support persistent backup or overflow. See [bootstrap-diskstore]

WARNING	The `<*-region>` elements no longer allow a nested `<disk-store>` element.

- GemFire Sub-Regions are supported via nested `<*-region>` elements.
- A `<local-region>` element has been added to configure a Local Region.
- Support for the re-designed WAN Gateway in GemFire 7.0.

3.2. New in the 1.3 Release

- Annotation support for GemFire Functions. It is now possible to declare and register Functions written as POJOs using annotations. In addition, Function executions are defined as annotated interfaces, similar to the way Spring Data Repositories work. See Annotation Support for Function Execution.
- Added a `<datasource>` element to the SDG **gfe-data** namespace to simplify establishing a basic client connection to a GemFire data grid.
- Added a `<json-region-autoproxy>` element to the SDG **gfe-data** namespace to support JSON features introduced in GemFire 7.0, enabling Spring AOP to perform the necessary conversions automatically on Region operations.
- Upgraded to GemFire 7.0.1 and added namespace support for new AsyncEventQueue attributes.
- Added support for setting subscription interest policy on Regions.
- Support for void returns on Function executions. See Annotation Support for Function Execution for complete details.
- Support for persisting Local Regions. See Local Region and [bootstrap:region:common:attributes].

- Support for entry time-to-live and entry idle-time on a GemFire Client Cache. See Configuring a GemFire ClientCache.
- Support for multiple Spring Data GemFire web-based applications using a single GemFire cluster, operating concurrently inside tc Server.
- Support for concurrency-checks-enabled on all GemFire Cache Region definitions using the SDG **gfe** namespace. See [bootstrap:region:common:attributes].
- Support for Cache Loaders and Cache Writers on Client, Local Regions. See [bootstrap:region:common:loaders-writers].
- Support for registering CacheListeners, AsyncEventQueues and Gateway Senders on GemFire Cache Sub-Regions.
- Support for PDX persistent keys in GemFire Regions.
- Support for correct Partition Region bean creation in a Spring context when collocation is specified with the **colocated-with** attribute.
- Full support for GemFire Cache Sub-Regions using proper, nested `<*-region>` element syntax in the SDG **gfe** namespace.
- Upgraded Spring Data GemFire to Spring Framework 3.2.8.
- Upgraded Spring Data GemFire to Spring Data Commons 1.7.1.

3.3. New in the 1.4 Release

- Upgrades to Pivotal GemFire 7.0.2.
- Upgrades to *Spring Data Commons* 1.8.x.RELEASE.
- Upgrades to *Spring Framework* 3.2.x.RELEASE.
- Integrates *Spring Data GemFire* with *Spring Boot*, which includes both a **spring-boot-starter-data-gemfire** POM along with a *Spring Boot* sample application demonstrating GemFire Cache Transactions configured with SDG and bootstrapped with *Spring Boot*.
- Support for bootstrapping a Spring `ApplicationContext` in a GemFire Server when started from *Gfsh*. See Bootstrapping a Spring ApplicationContext in Pivotal GemFire for more details.
- Support for persisting application domain object/entities to multiple GemFire Cache Regions. See Entity Mapping for more details.
- Support for persisting application domain object/entities to GemFire Cache Sub-Regions, avoiding collisions when Sub-Regions are uniquely identifiable, but identically named. See Entity Mapping for more details.
- Adds strict XSD type rules to, and full support for, Data Policies and Region Shortcuts on all GemFire Cache Region types.
- Changed the default behavior of SDG `<*-region>` elements from lookup to always create a new Region along with an option to restore old behavior using the **ignore-if-exists** attribute. See Common Region Attributes and [bootstrap:region:common:regions-subregions-lookups-caution] for more details.
- *Spring Data GemFire* can now be fully built and ran on JDK 7 and JDK 8.

CAUTION	Pivotal GemFire has not yet been fully tested and certified to run JDK 8; See GemFire User Guide for additional details.

3.4. New in the 1.5 Release

- Maintains support for Pivotal GemFire 7.0.2.
- Upgrades to *Spring Data Commons* 1.9.x.RELEASE.
- Upgrades to *Spring Framework* 4.0.x.RELEASE.
- Reference Guide migrated to Asciidoc.
- Renewed support for deploying *Spring Data GemFire* in an OSGi container.
- Removed all default values in the *Spring Data GemFire* XML namespace Region-type elements to rely on GemFire defaults instead.
- Added convenience to automatically create Disk Store directory locations.
- SDG annotated Function implementations can now be executed from *Gfsh*.
- Enable GemFire `GatewayReceivers` to be started manually.
- Support for Auto Region Lookups. See [bootstrap:region:auto-lookup] for further details.
- Support for Region Templates. See [bootstrap:region:common:region-templates] for further details.

3.5. New in the 1.6 Release

- Upgrades to Pivotal GemFire 8.0.0.
- Upgrades to *Spring Data Commons* 1.10.x.RELEASE.
- Maintains support on *Spring Framework* 4.0.x.RELEASE.
- Adds support for GemFire 8's new Cluster-based Configuration.
- Enables 'auto-reconnect' functionality to be employed in Spring-configured GemFire Servers.
- Allows the creation of concurrent and parallel Async Event Queues and Gateway Senders.
- Adds support for GemFire 8's Region data compression.
- Adds attributes to set both critical and warning percentages on Disk Store usage.
- Supports the capability to add the new EventSubstitutionFilters to GatewaySenders.

3.6. New in the 1.7 Release

- Upgrades to Pivotal GemFire 8.1.0.
- Upgrades to *Spring Data Commons* 1.11.x.RELEASE.
- Upgrades to *Spring Framework* 4.1.x.RELEASE.
- Early access support for Pivotal GemFire.
- Support for adding *Spring*-defined Cache Listeners, Loaders and Writers on "existing" GemFire

Regions configured in *Spring* XML, `cache.xml` or even with Pivotal GemFire's *Cluster Config*.

- *Spring* JavaConfig support added to `SpringContextBootstrappingInitializer`.
- Support for custom `ClassLoaders` in `SpringContextBootstrappingInitializer` to load *Spring* -defined bean classes.
- Support for `LazyWiringDeclarableSupport` re-initialization and complete replacement for `WiringDeclarableSupport`.
- Adds `locators` and `servers` attributes to the `<gfe:pool>` element allowing variable Locator/Server endpoint lists configured with *Spring's* property placeholders.
- Enables the use of `<gfe-data:datasource>` element with non-*Spring* configured Pivotal GemFire Servers.
- Multi-Index definition and creation support.
- Annotation-based Data Expiration
- OQL Query Extensions using Annotations
- Configuring the Snapshot Service

3.7. New in the 1.8 Release

- Upgrades to Pivotal GemFire 8.2.0.
- Upgrades to *Spring Data Commons* 1.12.x.RELEASE.
- Upgrades to *Spring Framework* 4.2.x.RELEASE.
- Adds Maven POM to build SDG with Maven.
- Adds support for CDI.
- Enables a `ClientCache` to be configured without a `Pool`.
- `<gfe:cache>` and `<gfe:client-cache>` elements `use-bean-factory-locator` attributes now default to **false**.
- Adds `durable-client-id` and `durable-client-timeout` attributes to `<gfe:client-cache>`.
- GemfirePersistentProperty now properly handles other non-entity, scalar-like types (e.g. `BigDecimal`, `BigInteger`).
- Prevents SDG-defined `Pools` from being destroyed before `Regions` that use those `Pools`.
- Handles case-insensitive GemFire OQL queries defined as *Repository* query methods.
- Changes `GemFireCache.evict(key)` to call `Region.remove(key)` in SDG's *Spring Cache Abstraction* support.
- Fixes `RegionNotFoundException` with *Repository* queries on a client `Region` associated with a specific `Pool` configured for GemFire server groups.
- Changes `Gateway Senders/Receivers` to no longer be tied to the *Spring* container.

3.8. New in the 1.9 Release

- Upgrades to Pivotal GemFire 8.2.4.
- Upgrades to *Spring Data Commons* 1.13.x.RELEASE.
- Upgrades to *Spring Framework* 4.3.x.RELEASE.
- Introduces an entirely new Annotation-based configuration model inspired by *Spring Boot*.
- Adds support for suspend and resume in the `GemfireTransactionManager`.
- Adds support in *Repositories* to use the bean `id` property as the Region key when the `@Id` annotation is not present.
- Uses `MappingPdxSerializer` as the default GemFire serialization strategy when `@EnablePdx` is used.
- Enables `GemfireCacheManager` to explicitly list Region names to be used in the *Spring's Caching Abstraction*.
- Configure GemFire Caches, CacheServers, Locators, Pools, Regions, Indexes, DiskStores, Expiration, Eviction, Statistics, Mcast, HttpService, Auth, SSL, Logging, System Properties.
- Repository support with multiple *Spring Data* modules on the classpath.

3.9. New in the 2.0 Release

- Upgrades to Pivotal GemFire 9.0.x.
- Upgrades to *Spring Data Commons* 2.0.x.RELEASE.
- Upgrades to *Spring Framework* 5.0.x.RELEASE.
- Reorganizes the SDG codebase by better packaging different classes and components by concern.
- Adds extensive support for Java 8 types, particularly in the SD *Repository* abstraction.
- Changes to the *Repository* interface and abstraction, e.g. IDs are no longer required to be `java.io.Serializable`.
- Sets `@EnableEntityDefinedRegions` annotation `ignoreIfExists` attribute to **true** by default.
- Sets `@Indexed` annotation `override` attribute to **false** by default.
- Renames `@EnableIndexes` to `@EnableIndexing`.
- Introduces a `InterestsBuilder` class to easily and conveniently express Interests in keys/values between client and server when using JavaConfig.
- Adds support for Off-Heap, Redis Adapter and GemFire's new Security framework to the Annotation configuration model.

Reference Guide

Chapter 4. Document Structure

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

Bootstrapping Pivotal GemFire with the Spring container describes the configuration support provided for bootstrapping, configuring, initializing and accessing Pivotal GemFire Caches, Regions, and related Distributed System components.

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

Working with Pivotal GemFire Serialization describes the enhancements for Pivotal GemFire (de)serialization and management of associated objects.

POJO mapping describes persistence mapping for POJOs stored in Pivotal GemFire using *Spring Data*.

Spring Data GemFire Repositories describes how to create and use *Spring Data Repositories* to access data in Pivotal GemFire.

Annotation Support for Function Execution describes how to create and use Pivotal GemFire Functions using Annotations.

Bootstrapping a Spring ApplicationContext in Pivotal GemFire describes how to bootstrap a *Spring* `ApplicationContext` running in an Pivotal GemFire server using *Gfsh*.

Sample Applications describes the examples provided with the distribution to illustrate the various features available in *Spring Data GemFire*.

Chapter 5. Bootstrapping Pivotal GemFire with the Spring container

Spring Data GemFire provides full configuration and initialization of the Pivotal GemFire In-Memory Data Grid (IMDG) using the *Spring* IoC container. The framework includes several classes to help simplify the configuration of Pivotal GemFire components including: Caches, Regions, Indexes, DiskStores, Functions, WAN Gateways, persistence backup along with several other Distributed System components in order to support a variety of use cases with minimal effort.

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

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

Spring Data GemFire's XML namespace supports full configuration of the Pivotal GemFire In-Memory Data Grid (IMDG). The XML namespace is the preferred way to configure Pivotal GemFire in a *Spring* context in order to properly manage GemFire's lifecycle inside the *Spring* container. While support for GemFire's native `cache.xml` persists for legacy reasons, GemFire application developers are encouraged to do everything in *Spring* XML to take advantage of the many wonderful things *Spring* has to offer such as modular XML configuration, property placeholders and overrides, SpEL, and environment profiles. Behind the XML namespace, *Spring Data GemFire* makes extensive use of *Spring's* `FactoryBean` pattern to simplify the creation, configuration and initialization of GemFire components.

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

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

5.2. Using the Core Namespace

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:gfe="http://www.springframework.org/schema/geode"① ②
  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/geode
http://www.springframework.org/schema/gemfire/spring-geode.xsd"> ③

  <bean id ... >

  <gfe:cache ...> ④

</beans>
```

① *Spring Data GemFire* XML namespace prefix. Any name will do but through out this reference documentation, `gfe` will be used.

② The XML namespace prefix is mapped to the URI.

③ The XML namespace URI location. Note that even though the location points to an external address (which does exist and is valid), *Spring* will resolve the schema locally as it is included in the *Spring Data GemFire* library.

④ Example declaration using the XML namespace with the `gfe` prefix.

NOTE

It is possible to change the default namespace from `beans` to `gfe`. This is useful for XML configuration composed mainly of GemFire components as it avoids declaring the prefix. To achieve this, simply swap the namespace prefix declaration above:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/geode" ①
  xmlns:beans="http://www.springframework.org/schema/beans" ②
  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/geode
http://www.springframework.org/schema/gemfire/spring-geode.xsd">

  <beans:bean id ... > ③

  <cache ...> ④

</beans>
```

① The default namespace declaration for this XML document points to the *Spring Data GemFire* XML namespace.

② The `beans` namespace prefix declaration for *Spring's* raw bean definitions.

③ Bean declaration using the `beans` namespace. Notice the prefix.

④ Bean declaration using the `gfe` namespace. Notice the lack of prefix since `gfe` is the default namespace.

5.3. Using the Data Access Namespace

In addition to the core XML namespace (`gfe`), *Spring Data GemFire* provides a `gfe-data` XML namespace primarily intended to simplify the development of Pivotal GemFire client applications. This namespace currently contains support for GemFire Repositories and function execution as well as includes a `<datasource>` tag that offers a convenient way to connect to the Pivotal GemFire data grid.

5.3.1. An Easy Way to Connect to GemFire

For many applications, a basic connection to a GemFire data grid using default values is sufficient. *Spring Data GemFire's* `<datasource>` tag provides a simple way to access data. The data source creates a `ClientCache` and connection `Pool`. In addition, it will query the cluster servers for all existing root Regions and create an (empty) client Region proxy for each one.

```
<gfe-data:datasource>
  <locator host="remotehost" port="1234"/>
</gfe-data:datasource>
```

The `<datasource>` tag is syntactically similar to `<gfe:pool>`. It may be configured with one or more nested `locator` or `server` tags to connect to an existing data grid. Additionally, all attributes available to configure a Pool are supported. This configuration will automatically create client Region beans for each Region defined on cluster members connected to the Locator, so they may be seamlessly referenced by *Spring Data* mapping annotations, `GemfireTemplate`, and wired into application classes.

Of course, you can explicitly configure client Regions. For example, if you want to cache data in local memory:

```
<gfe-data:datasource>
  <locator host="remotehost" port="1234"/>
</gfe-data:datasource>

<gfe:client-region id="Example" shortcut="CACHING_PROXY"/>
```

5.4. Configuring a Cache

To use Pivotal GemFire, a developer needs to either create a new `Cache` or connect to an existing one. With the current version of GemFire, there can be only one open Cache per VM (technically, per `ClassLoader`). In most cases, the `Cache` should only be created once.

NOTE

This section describes the creation and configuration of a peer cache member, appropriate in peer-to-peer (P2P) topologies and cache servers. A cache member can also be used in standalone applications and integration tests. However, in most typical production systems, most application processes will act as cache clients, creating a `ClientCache` instance instead. This is described in the sections Configuring a GemFire ClientCache and Client Region.

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

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

During Spring container initialization, any application context containing this cache definition will register a `CacheFactoryBean` that creates a Spring bean named `gemfireCache` referencing a GemFire `Cache` instance. This bean will refer to either an existing cache, or if one does not already 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 that depend on the cache respect this naming convention, so there is no need to explicitly declare the cache dependency. If you prefer, you can make the dependency explicit via the `cache-ref` attribute provided by various SDG XML namespace elements. Also, you can easily override the cache's bean name using the `id` attribute:

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

A GemFire `Cache` can be fully configured using Spring, however, GemFire's native XML configuration file, `cache.xml`, is also supported. For situations where the GemFire cache needs to be configured natively, simply provide a reference to the GemFire XML configuration file using the `cache-xml-location` attribute:

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

In this example, if a cache needs to be created, it will use a file named `cache.xml` located in the classpath root to configure it.

NOTE 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 XML configuration file, a developer may also specify GemFire System properties using any of Spring's `Properties` support features.

For example, the developer may use the `properties` element defined in the `util` namespace to define `Properties` directly or load properties from a properties file:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:gfe="http://www.springframework.org/schema/gemfire"
  xmlns:util="http://www.springframework.org/schema/util"
  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
    http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">

  <util:properties id="gemfireProperties" location="file:/path/to/gemfire.properties
"/>

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

</beans>
```

Using a properties file is recommended for externalizing environment specific settings outside the application configuration.

NOTE Cache settings apply only if a new cache needs to be created. If an open cache already exists in the VM, these settings are ignored.

5.4.1. Advanced Cache Configuration

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

```
①
<gfe:cache
    cache-xml-location=".."
    properties-ref=".."
    close="false"
    copy-on-read="true"
    critical-heap-percentage="90"
    eviction-heap-percentage="70"
    enable-auto-reconnect="false" ②
    lock-lease="120"
    lock-timeout="60"
    message-sync-interval="1"
    pdx-serializer-ref="myPdxSerializer"
    pdx-persistent="true"
    pdx-disk-store="diskStore"
    pdx-read-serialized="false"
    pdx-ignore-unread-fields="true"
    search-timeout="300"
    use-bean-factory-locator="true" ③
    use-cluster-configuration="false" ④
>

  <gfe:transaction-listener ref="myTransactionListener"/> ⑤

  <gfe:transaction-writer> ⑥
    <bean class="org.example.app.gemfire.transaction.TransactionWriter"/>
  </gfe:transaction-writer>

  <gfe:gateway-conflict-resolver ref="myGatewayConflictResolver"/> ⑦

  <gfe:dynamic-region-factory/> ⑧

  <gfe:jndi-binding jndi-name="myDataSource" type="ManagedDataSource"/> ⑨

</gfe:cache>
```

① Various cache options are supported by attributes. For further information regarding anything shown in this example, please consult the GemFire product documentation. The `close` attribute determines whether the cache should be closed when the Spring application context is closed. The default is `true`, however, for use cases in which multiple application contexts use the cache (common in web applications), set this value to `false`.

② Setting the `enable-auto-reconnect` attribute to true (default is false), allows a disconnected GemFire member to automatically reconnect and rejoin the GemFire cluster. See the GemFire product documentation for more details.

③ Setting the `use-bean-factory-locator` attribute to `true` (defaults to `false`) is only applicable when both Spring (XML) configuration meta-data and GemFire `cache.xml` is used to configure the GemFire cache node (whether client or peer). This option allows GemFire components (e.g. `CacheLoader`) expressed in `cache.xml` to be auto-wired with beans (e.g. `DataSource`) defined in the Spring application context. This option is typically used in conjunction with `cache-xml-location`.

④ Setting the `use-cluster-configuration` attribute to `true` (default is `false`) enables a GemFire member to retrieve the common, shared Cluster-based configuration from a Locator. See the GemFire product documentation for more details.

⑤ Example of a `TransactionListener` callback declaration using a bean reference. The referenced bean must implement TransactionListener. A `TransactionListener` can be implemented to handle transaction related events (e.g. afterCommit, afterRollback).

⑥ Example of a `TransactionWriter` callback declaration using an inner bean declaration. The bean must implement TransactionWriter. The `TransactionWriter` is a callback that is allowed to veto a transaction.

⑦ Example of a `GatewayConflictResolver` callback declaration using a bean reference. The referenced bean must implement http://geode.apache.org/releases/latest/javadoc/org/apache/geode/cache/util/GatewayConflictResolver.html [GatewayConflictResolver]. A `GatewayConflictResolver` is a Cache-level plugin that is called upon to decide what to do with events that originate in other systems and arrive through the WAN Gateway.

⑧ Enable GemFire's DynamicRegionFactory, which provides a distributed Region creation service.

⑨ Declares a JNDI binding to enlist an external DataSource in a GemFire transaction.

Enabling PDX Serialization

The example above includes a number of attributes related to GemFire'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 `PdxSerializer` which is specified via the `pdx-serializer` attribute. GemFire provides an implementing class `org.apache.geode.pdx.ReflectionBasedAutoSerializer` that uses Java Reflection, 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 `PdxSerializer` interface.

More information on serialization support can be found in Working with Pivotal GemFire Serialization

Enabling auto-reconnect

Setting the `<gfe:cache enable-auto-reconnect="[true|false*]>` attribute to `true` should be done with care.

Generally, 'auto-reconnect' should only be enabled in cases where *Spring Data GemFire's* XML namespace is used to configure and bootstrap a new, non-application GemFire Server to add to a cluster. In other words, 'auto-reconnect' should not be enabled when *Spring Data GemFire* is used to develop and build an GemFire application that also happens to be a peer cache member of the GemFire cluster.

The main reason for this is that most GemFire applications use references to the GemFire cache or

Regions in order to perform data access operations. These references are "injected" by the Spring container into application components (e.g. DAOs or Repositories) for use by the application. When a peer member is forcefully disconnected from the rest of the cluster, presumably because the peer member has become unresponsive or a network partition separates one or more peer members into a group too small to function as an independent distributed system, the peer member will shutdown and all GemFire component references (e.g. Cache, Regions, etc) become invalid.

Essentially, the current forced-disconnect processing logic in each peer member dismantles the system from the ground up. The JGroups stack shuts down, the Distributed System is put in a shutdown state and finally, the Cache is closed. Effectively, all memory references become stale and are lost.

After being disconnected from the Distributed System a peer member enters a "reconnecting" state and periodically attempts to rejoin the Distributed System. If the peer member succeeds in reconnecting, the member rebuilds its "view" of the Distributed System from existing members and receives a new Distributed System ID. Additionally, all Cache, Regions and other GemFire components are reconstructed. Therefore, all old references, which may have been injected into application by the Spring container are now stale and no longer valid.

GemFire makes no guarantee, even when using the GemFire public Java API, that application Cache, Region or other component references will be automatically refreshed by the reconnect operation. As such, GemFire applications must take care to refresh their own references.

Unfortunately, there is no way to be notified of a disconnect event, and subsequently, a reconnect event. If that were the case, the application developer would have a clean way to know when to call `ConfigurableApplicationContext.refresh()`, if even applicable for an application to do so, which is why this "feature" of Pivotal GemFire is not recommended for peer cache GemFire applications.

For more information about 'auto-reconnect', see GemFire's product documentation.

Using Cluster-based Configuration

Pivotal GemFire's Cluster Configuration Service is a convenient way for any peer member joining the cluster to get a "consistent view" of the cluster by using the shared, persistent configuration maintained by a Locator. Using the Cluster-based Configuration ensures the peer member's configuration will be compatible with the GemFire Distributed System when the member joins.

This feature of *Spring Data GemFire* (setting the `use-cluster-configuration` attribute to `true`) works in the same way as the `cache-xml-location` attribute, except the source of the GemFire configuration meta-data comes from the network via a Locator as opposed to a native `cache.xml` file residing in the local file system.

All GemFire native configuration meta-data, whether from `cache.xml` or from the Cluster Configuration Service, gets applied before any *Spring* (XML) configuration meta-data. As such, *Spring's* config serves to "augment" the native GemFire configuration meta-data and would most likely be specific to the application.

Again, to enable this feature, just specify the following in the *Spring* XML config:

```
  <gfe:cache use-cluster-configuration="true"/>
```

NOTE	While certain GemFire tools, like *Gfsh*, have their actions "recorded" when schema-like changes are made (e.g. `gfsh>create region --name=Example --type=PARTITION`), *Spring Data GemFire's* configuration meta-data is not recorded. The same is true when using GemFire's public Java API directly; it too is not recorded.

For more information on GemFire's Cluster Configuration Service, see the product documentation.

5.4.2. Configuring a GemFire CacheServer

Spring Data GemFire includes dedicated support for configuring a CacheServer, allowing complete configuration through the Spring container:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:gfe="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/context
http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/gemfire
http://www.springframework.org/schema/gemfire/spring-gemfire.xsd
">

  <gfe:cache/>

  <!-- Example depicting serveral GemFire CacheServer configuration options -->
  <gfe:cache-server id="advanced-config" auto-startup="true"
       bind-address="localhost" host-name-for-clients="localhost" port=
"${gemfire.cache.server.port}"
       load-poll-interval="2000" max-connections="22" max-message-count="1000" max-
threads="16"
       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`. property placeholder reads one or more properties files and then replaces property placeholders with values at runtime. This allows administrators to change values without having to touch the main application configuration. *Spring* also provides the SpEL and the environment abstraction to support externalization of environment-specific properties from the main codebase, easing deployment across multiple machines.
NOTE	To avoid initialization problems, the `CacheServer` started by *Spring Data GemFire* will start **after** the *Spring* 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 elements as the server might start after your components and thus not be seen by the clients connecting right away.

5.4.3. Configuring a GemFire ClientCache

In addition to defining a GemFire peer Cache, *Spring Data GemFire* also supports the definition of a GemFire ClientCache in a *Spring* context. A `ClientCache` definition is very similar in configuration and use to the GemFire peer Cache and is supported by the `org.springframework.data.gemfire.client.ClientCacheFactoryBean`.

The simplest definition of a GemFire cache client using default configuration can be accomplished with the following declaration:

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

`client-cache` supports many of the same options as the cache element. However, as opposed to a **full-fledged** peer cache member, a cache client connects to a remote cache server through a Pool. By default, a Pool is created to connect to a server running on `localhost`, listening to port `40404`. The default Pool is used by all client Regions unless the Region is configured to use a specific Pool.

Pools can be defined with the `pool` element. This client-side Pool can be used to configure connectivity directly to a server for individual entities or the entire cache through one or more Locators.

For example, to customize the default Pool used by the `client-cache`, the developer needs to define a Pool and wire it to the cache definition:

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

  <gfe:pool id="myPool" subscription-enabled="true">
    <gfe:locator host="${gemfire.locator.host}" port="${gemfire.locator.port}"/>
  </gfe:pool>
</beans>
```

The `<client-cache>` element also has a `ready-for-events` attribute. If set to `true`, the client cache initialization will include a call to ClientCache.readyForEvents().

Client-side configuration is covered in more detail in Client Region.

GemFire's DEFAULT Pool and Spring Data GemFire Pool Definitions

If a GemFire `ClientCache` is local-only, then no Pool definition is required. For instance, a developer may define:

```
<gfe:client-cache/>

<gfe:client-region id="Example" shortcut="LOCAL"/>
```

In this case, the "Example" Region is `LOCAL` and no data is distributed between the client and a server, therefore, no Pool is necessary. This is true for any client-side, local-only Region, as defined by the GemFire's ClientRegionShortcut (all `LOCAL_*` shortcuts).

However, if a client Region is a (caching) proxy to a server-side Region, then a Pool is required. There are several ways to define and use a Pool in this case.

When a client cache, Pool and proxy-based Region are all defined, but not explicitly identified, *Spring Data GemFire* will resolve the references automatically for you.

For example:

```
<gfe:client-cache/>

<gfe:pool>
  <gfe:locator host="${geode.locator.host}" port="${geode.locator.port}"/>
</gfe:pool>

<gfe:client-region id="Example" shortcut="PROXY"/>
```

In the example above, the client cache is identified as `gemfireCache`, the Pool as `gemfirePool` and the client Region as "Example". However, the client cache will initialize GemFire's DEFAULT Pool from `gemfirePool` and the client Region will use the `gemfirePool` when distributing data between the client and the server.

Basically, *Spring Data GemFire* resolves the above configuration to the following:

```
<gfe:client-cache id="gemfireCache" pool-name="gemfirePool"/>

<gfe:pool id="gemfirePool">
  <gfe:locator host="${geode.locator.host}" port="${geode.locator.port}"/>
</gfe:pool>

<gfe:client-region id="Example" cache-ref="gemfireCache" pool-name="gemfirePool"
shortcut="PROXY"/>
```

GemFire still creates a Pool called "DEFAULT". *Spring Data GemFire* will just cause the "DEFAULT" Pool to be initialized from the `gemfirePool`. This is useful in situations where multiple Pools are defined and client Regions are using separate Pools.

Consider the following:

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

<gfe:pool id="locatorPool">
  <gfe:locator host="${geode.locator.host}" port="${geode.locator.port}"/>
</gfe:pool>

<gfe:pool id="serverPool">
  <gfe:server host="${geode.server.host}" port="${geode.server.port}"/>
</gfe:pool>

<gfe:client-region id="Example" pool-name="serverPool" shortcut="PROXY"/>

<gfe:client-region id="AnotherExample" shortcut="CACHING_PROXY"/>

<gfe:client-region id="YetAnotherExample" shortcut="LOCAL"/>
```

In this setup, the GemFire client cache's "DEFAULT" Pool is initialized from "locatorPool" as specified with the `pool-name` attribute. There is no *Spring Data GemFire*-defined `gemfirePool` since both Pools were explicitly identified (named) "locatorPool" and "serverPool", respectively.

The "Example" Region explicitly refers to and uses the "serverPool" exclusively. The "AnotherExample" Region uses GemFire's "DEFAULT" Pool, which was configured from the "locatorPool" based on the client cache bean definition's `pool-name` attribute.

Finally, the "YetAnotherExample" Region will not use a Pool since it is `LOCAL`.

NOTE	The "AnotherExample" Region would first look for a Pool bean named `gemfirePool`, but that would require the definition of an anonymous Pool bean (i.e. `<gfe:pool/>`) or a Pool bean explicitly named `gemfirePool` (e.g. `<gfe:pool id="gemfirePool"/>`).

NOTE	We could have either named "locatorPool", "gemfirePool", or made the Pool bean definition anonymous and it would have the same effect as the above configuration.

5.5. Configuring a Region

A Region is required to store and retrieve data from the cache. `org.apache.geode.cache.Region` is an interface extending `java.util.Map` and enables basic data access using familiar key-value semantics. The `Region` interface is wired into application 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:

- **REPLICATE** - 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.
- **PARTITION** - Data is partitioned into buckets (sharded) among cache members that define the Region. This provides high read and write performance and is suitable for 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 REPLICATE or PARTITION Region hosted on cache servers in a cluster. 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 up-to-date (synchronized) with changes originating from remote processes that access the same server Region.

For more information about the various Region types and their capabilities as well as configuration options, please refer to Pivotal GemFire's documentation on Region Types.

5.5.1. Using an externally configured Region

To reference Regions already configured in a GemFire native `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 identified as `ordersRegion` for an existing Region named `Orders`, you can use the following bean definition:

```
<gfe:lookup-region id="ordersRegion" name="Orders"/>
```

If `name` is not specified, the bean's `id` will be used as the name of the Region. The example above becomes:

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

CAUTION	If the Region does not exist, an initialization exception will be thrown. To configure new Regions, proceed to the appropriate sections below.

In the previous examples, since no cache name was explicitly defined, the default naming convention (`gemfireCache`) was used. Alternately, one can reference the cache bean with the `cache-ref` attribute:

```
<gfe:cache id="myCache"/>
<gfe:lookup-region id="ordersRegion" name="Orders" cache-ref="myCache"/>
```

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

5.5.2. Auto Region Lookup

"auto-lookup" allows all Regions defined in a GemFire native `cache.xml` file to be imported into a *Spring* application context when using the`cache-xml-location` attribute on the `<gfe:cache>` element.

For instance, given a `cache.xml` file of...

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

  <region name="Parent" refid="REPLICATE">
    <region name="Child" refid="REPLICATE"/>
  </region>

</cache>
```

A developer may import the `cache.xml` file as follows...

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

The developer may then use the `<gfe:lookup-region>` element (e.g. `<gfe:lookup-region id="Parent"/>`) to reference specific Regions as beans in the *Spring* context, or the user may choose to import all Regions defined in `cache.xml` with:

```
<gfe:auto-region-lookup/>
```

Spring Data GemFire will automatically create beans for all GemFire Regions defined in `cache.xml`

that have not been explicitly added to the *Spring* context with explicit `<gfe:lookup-region>` bean declarations.

It is important to realize that *Spring Data GemFire* uses a *Spring* BeanPostProcessor to post process the cache after it is both created and initialized to determine the Regions defined in GemFire to add as beans in the *Spring* application context.

You may inject these "auto-looked-up" Regions like any other bean defined in the *Spring* application context with 1 exception; you may need to define a `depends-on` association with the 'gemfireCache' bean as follows...

```
package example;

import ...

@Repository("appDao")
@DependsOn("gemfireCache")
public class ApplicationDao extends DaoSupport {

    @Resource(name = "Parent")
    private Region<?, ?> parent;

    @Resource(name = "/Parent/Child")
    private Region<?, ?> child;

    ...
}
```

The example above is applicable when using *Spring's* `component-scan` functionality.

If you are declaring your components using *Spring* XML config, then you would do...

```
<bean class="example.ApplicationDao" depends-on="gemfireCache"/>
```

This ensures the GemFire cache and all the Regions defined in `cache.xml` get created before any components with auto-wire references when using the new `<gfe:auto-region-lookup>` element.

5.5.3. Configuring Regions

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

- LOCAL Region: `<local-region>`
- PARTITION Region: `<partitioned-region>`
- REPLICATE Region: `<replicated-region>`
- Client Region: `<client-region>`

Please see the Pivotal GemFire documentation for a comprehensive description of Region Types.

Common Region Attributes

The following table lists attributes available for all Region types:

Table 1. Common Region Attributes

Name	Values	Description
cache-ref	GemFire Cache bean reference	The name of the bean defining the GemFire Cache (by default 'gemfireCache').
cloning-enabled	boolean, default:false	When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.
close	boolean, default:false	Determines whether the Region should be closed at shutdown.
concurrency-checks-enabled	boolean, default:true	Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed Regions.
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	Determines 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.
disk-synchronous	boolean, default:true	Determines whether Disk Store writes are synchronous.
id	Any valid bean name.	Will be the Region name by default if no `name` attribute is specified.
ignore-if-exists	boolean, default:false	Ignores this bean definition if the Region already exists in the cache, resulting in a lookup instead.
ignore-jta	boolean, default:false	Determines whether this Region will participate in JTA transactions.
index-update-type	synchronous or asynchronous, default:synchronous	Determines whether Indices will be updated synchronously or asynchronously on entry creation.
initial-capacity	integer, default:16	The initial memory allocation for the number of Region entries.
key-constraint	Any valid, fully-qualified Java class name.	Expected key type.

Name	Values	Description
load-factor	float, default:.75	Sets the initial parameters on the underlying java.util.ConcurrentHashMap used for storing Region entries.
name	Any valid Region name.	The name of the Region. If not specified, it will assume the value of the `id` attribute (a.k.a. bean name).
persistent	*boolean, default:false	Determines whether the Region will persist entries to local disk (Disk Store).
shortcut	See http://geode.apache.org/releases/latest/javadoc/org/apache/geode/cache/RegionShortcut.html	The `RegionShortcut` for this Region. Allows easy initialization of the Region based on pre-defined defaults.
statistics	boolean, default:false	Determines whether the Region reports statistics.
template	The name of a Region Template.	A reference to a bean created via one of the `*region-template` elements.
value-constraint	Any valid, fully-qualified Java class name.	Expected value type.

CacheListeners

`CacheListeners` are registered with a Region to handle Region events such as when entries are created, updated, destroyed and so on. A `CacheListener` can be any bean that implements the `CacheListener` interface. A Region may have multiple listeners, declared using the `cache-listener` element nested in the containing `*-region` element.

In the example below, there are two `CacheListener's` declared. The first references a named, top-level *Spring* bean; the second is an anonymous inner bean definition.

```
<gfe:replicated-region id="regionWithListeners">
  <gfe:cache-listener>
    <!-- nested CacheListener bean reference -->
    <ref bean="myListener"/>
    <!-- nested CacheListener bean definition -->
    <bean class="org.example.app.geode.cache.AnotherSimpleCacheListener"/>
  </gfe:cache-listener>

  <bean id="myListener" class="org.example.app.geode.cache.SimpleCacheListener"/>
</gfe:replicated-region>
```

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

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

```
<beans>
  <gfe:replicated-region id="exampleReplicateRegionWithCacheListener">
    <gfe:cache-listener ref="myListener"/>
  </gfe:replicated-region>

  <bean id="myListener" class="example.CacheListener"/>
</beans>
```

NOTE	*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.

CacheLoaders and CacheWriters

Similar to `cache-listener`, the namespace provides `cache-loader` and `cache-writer` elements to register these GemFire components respectively for a Region.

A `CacheLoader` is invoked on a cache miss to allow an entry to be loaded from an external data source, such as a database. A `CacheWriter` is invoked before 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 `CacheLoader` and `CacheWriter` per Region. However, either declaration style may be used.

Example:

```
<beans>
  <gfe:replicated-region id="exampleReplicateRegionWithCacheLoaderAndCacheWriter">
    <gfe:cache-loader ref="myLoader"/>
    <gfe:cache-writer>
      <bean class="example.CacheWriter"/>
    </gfe:cache-writer>
  </gfe:replicated-region>

  <bean id="myLoader" class="example.CacheLoader">
    <property name="dataSource" ref="mySqlDataSource"/>
  </bean>

  <!-- DataSource bean definition -->
</beans>
```

See `CacheLoader` and `CacheWriter` in the Pivotal GemFire documentation for more details.

5.5.4. Compression

GemFire Regions may also be compressed in order to reduce JVM memory consumption and pressure to possibly avoid stop the world GCs. When you enable compression for a Region, all values stored in the Region, in-memory are compressed while keys and indexes remain uncompressed. New values are compressed when put into Region and all values are decompressed automatically when read back from the Region. Values are not compressed when persisted to disk or when sent over the wire to other peer members or clients.

Example:

```
<beans>
  <gfe:replicated-region id="exampleReplicateRegionWithCompression">
    <gfe:compressor>
      <bean class="org.apache.geode.compression.SnappyCompressor"/>
    </gfe:compressor>
  </gfe:replicated-region>
</beans>
```

Please refer to Pivotal GemFire's documentation for more information on Region Compression.

5.5.5. Subregions

Spring Data GemFire also supports 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 `Monospaced ([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 application beans that need them, such as `GemfireTemplate`. The full path should also be used in OQL query strings.

5.5.6. Region Templates

Spring Data GemFire also supports Region Templates. This feature allows developers to define common Region configuration settings and attributes once and reuse the configuration among many Region bean definitions declared in the *Spring* application context.

Spring Data GemFire includes 5 Region template tags in namespace:

Table 2. Region Template Tags

Tag Name	Description
`<gfe:region-template>`	Defines common, generic Region attributes; extends `regionType` in the namespace.
`<gfe:local-region-template>`	Defines common, 'Local' Region attributes; extends `localRegionType` in the namespace.
`<gfe:partitioned-region-template>`	Defines common, 'PARTITION' Region attributes; extends `partitionedRegionType` in the namespace.
`<gfe:replicated-region-template>`	Defines common, 'REPLICATE' Region attributes; extends `replicatedRegionType` in the namespace.
`<gfe:client-region-template>`	Defines common, 'Client' Region attributes; extends `clientRegionType` in the namespace.

In addition to the tags, concrete `<gfe:*-region>` elements along with the abstract `<gfe:*-region-template>` elements have a `template` attribute used to define the Region Template from which the Region will inherit its configuration. Region Templates may even inherit from other Region Templates.

Here is an example of 1 possible configuration…

```
<beans>
  <gfe:async-event-queue id="AEQ" persistent="false" parallel="false" dispatcher-
threads="4">
    <gfe:async-event-listener>
      <bean class="example.AeqListener"/>
    </gfe:async-event-listener>
  </gfe:async-event-queue>

  <gfe:region-template id="BaseRegionTemplate" initial-capacity="51" load-factor="
0.85" persistent="false" statistics="true"
      key-constraint="java.lang.Long" value-constraint="java.lang.String">
    <gfe:cache-listener>
      <bean class="example.CacheListenerOne"/>
      <bean class="example.CacheListenerTwo"/>
    </gfe:cache-listener>
    <gfe:entry-ttl timeout="600" action="DESTROY"/>
    <gfe:entry-tti timeout="300 action="INVLIDATE"/>
  </gfe:region-template>

  <gfe:region-template id="ExtendedRegionTemplate" template="BaseRegionTemplate" load-
factor="0.55">
    <gfe:cache-loader>
      <bean class="example.CacheLoader"/>
    </gfe:cache-loader>
    <gfe:cache-writer>
      <bean class="example.CacheWriter"/>
    </gfe:cache-writer>
    <gfe:async-event-queue-ref bean="AEQ"/>
  </gfe:region-template>

  <gfe:partitioned-region-template id="PartitionRegionTemplate" template=
"ExtendedRegionTemplate"
      copies="1" load-factor="0.70" local-max-memory="1024" total-max-memory="16384"
value-constraint="java.lang.Object">
    <gfe:partition-resolver>
      <bean class="example.PartitionResolver"/>
    </gfe:partition-resolver>
    <gfe:eviction type="ENTRY_COUNT" threshold="8192000" action="OVERFLOW_TO_DISK"/>
  </gfe:partitioned-region-template>

  <gfe:partitioned-region id="TemplateBasedPartitionRegion" template=
"PartitionRegionTemplate"
      copies="2" local-max-memory="8192" persistent="true" total-buckets="91"/>
</beans>
```

Region Templates work for Subregions as well. Notice that 'TemplateBasedPartitionRegion' extends 'PartitionRegionTemplate', which extends 'ExtendedRegionTemplate' that extends 'BaseRegionTemplate'. Attributes and sub-elements defined in subsequent, inherited Region bean definitions override what is in the parent.

How Templating Works

Spring Data GemFire applies Region Templates when the *Spring* application context configuration meta-data is **parsed**, and therefore, **must be declared in the order of inheritance**. In other words, parent templates must be defined before children. This ensures the proper configuration is applied, especially when element attributes or sub-elements are "overridden".

IMPORTANT	It is equally important to remember the Region types must only inherit from other similar typed Regions. For instance, it is not possible for a `<gfe:replicated-region>` to inherit from a `<gfe:partitioned-region-template>`.

NOTE	Region Templates are single-inheritance.

Caution concerning Regions, Subregions and Lookups

Previously, one of the underlying properties of the `replicated-region`, `partitioned-region`, `local-region` and `client-region` elements in the *Spring Data GemFire* XML namespace was to perform a lookup first before attempting to create a Region. This was done in case the Region already existed, which would be the case if the Region was defined in an imported GemFire native `cache.xml` configuration file. Therefore, the lookup was performed first to avoid any errors. This was by design and subject to change.

This behavior has been altered and the default behavior is now to create the Region first. If the Region already exists, then the creation logic fails-fast and an appropriate exception is thrown. However, much like the `CREATE TABLE IF NOT EXISTS …` DDL syntax, the *Spring Data GemFire* `<*-region>` namespace elements now includes a `ignore-if-exists` attribute, which re-instates the old behavior by performing a lookup of an existing Region identified by name, first. If an existing Region by name is found and `ignore-if-exists` is set to `true`, then the Region bean definition defined in *Spring* config is ignored.

WARNING	The *Spring* team highly recommends that the `replicated-region`, `partitioned-region`, `local-region` and `client-region` namespace elements be strictly used for defining new Regions only. One problem that could arise if the Regions defined by these elements already existed and the Region elements performed a lookup first is if the developer defined different Region semantics and behaviors for eviction, expiration, subscription, etc in his/her application config, then the Region definition may not match and could exhibit contrary behaviors to those required by the application. Even worse, the application developer may want to define the Region as a distributed Region (e.g. PARTITION) but in fact the existing Region definition is LOCAL.

IMPORTANT	Recommended Practice - Only use `replicated-region`, `partitioned-region`, `local-region` and `client-region` namespace elements to define new Regions.

Consider the following native GemFire `cache.xml` configuration file…

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

  <region name="Customers" refid="REPLICATE">
    <region name="Accounts" refid="REPLICATE">
      <region name="Orders" refid="REPLICATE">
        <region name="Items" refid="REPLICATE"/>
      </region>
    </region>
  </region>

</cache>
```

Also consider that you may have defined an application DAO as follows...

```
public class CustomerAccountDao extends GemDaoSupport {

    @Resource(name = "Customers/Accounts")
    private Region customersAccounts;

    ...
}
```

Here, we are injecting a reference to the `Customers/Accounts` Region in our application DAO. As such, it is not uncommon for a developer to define beans for all or even some of these Regions in *Spring* XML configuration meta-data as follows...

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:gfe="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/geode
http://www.springframework.org/schema/gemfire/spring-geode.xsd
">

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

  <gfe:lookup-region name="Customers/Accounts"/>
  <gfe:lookup-region name="Customers/Accounts/Orders"/>

</beans>
```

The `Customers/Accounts` and `Customers/Accounts/Orders` Regions are referenced as beans in the *Spring* application context as "Customers/Accounts" and "Customers/Accounts/Orders", respectively. The nice thing about using the `lookup-region` element and the corresponding syntax above is that it allows a developer to reference a Subregion directly without unnecessarily defining a bean for the parent Region (i.e. `Customers`).

However, if now the developer changes his/her configuration meta-data syntax to using the nested format, like so...

```
<gfe:lookup-region name="Customers">
  <gfe:lookup-region name="Accounts">
    <gfe:lookup-region name="Orders"/>
  </gfe:lookup-region>
</gfe:lookup-region>
```

Or, perhaps the developer erroneously chooses to use the top-level `replicated-region` element along with the `ignore-if-exists` attribute set to perform a lookup first, as in...

```
<gfe:replicated-region name="Customers" persistent="true" ignore-if-exists="true">
  <gfe:replicated-region name="Accounts" persistent="true" ignore-if-exists="true">
    <gfe:replicated-region name="Orders" persistent="true" ignore-if-exists="true"/>
  </gfe:replicated-region>
</gfe:replicated-region>
```

Then the Region beans defined in the *Spring* application context will consist of the following: `{ "Customers", "/Customers/Accounts", "/Customers/Accounts/Orders" }`. This means the dependency injected reference above (i.e. `@Resource(name = "Customers/Accounts"))` is now broken since no bean with name "Customers/Accounts" is actually defined.

GemFire is flexible in referencing both parent Regions and Subregions with or without the leading forward slash. For example, the parent can be referenced as "/Customers" or "Customers" and the child as "/Customers/Accounts" or just "Customers/Accounts". However, _Spring Data _GemFire is very specific when it comes to naming beans after Regions, typically always using the forward slash (/) to represent Subregions (e.g. "/Customers/Accounts").

Therefore, it is recommended that users either use the nested `lookup-region` syntax as shown above, or define direct references with a leading forward slash (/) like so...

```
<gfe:lookup-region name="/Customers/Accounts"/>
<gfe:lookup-region name="/Customers/Accounts/Orders"/>
```

The example above where the nested `replicated-region` elements were used to reference the Subregions serves to illustrate the problem stated earlier. Are the Customers, Accounts and Orders Regions/Subregions persistent or not? Not, since the Regions were defined in the native GemFire `cache.xml` configuration file as `REPLICATES` and will exist by the time the cache is initialized, or once the `<gfe:cache>` bean is processed.

5.5.7. Data Eviction (with Overflow)

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 (referred to as **overflow** to disk).

Spring Data GemFire supports all eviction policies (entry count, memory and heap usage) for PARTITION Regions, REPLICATE Regions and client, local Regions using the nested `eviction` element.

For example, to configure a PARTITION Region to overflow to disk if the memory size exceeds more than 512 MB, a developer would specify the following configuration:

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

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

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

For a detailed description of eviction policies, please refer to the GemFire documentation on Eviction.

5.5.8. Data Expiration

Pivotal GemFire allows you to control how long entries exist in the cache. Expiration is driven by

elapsed time, as opposed to Eviction, which is driven by the entry count or heap/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 that an 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 (TTI)** - The amount of time in seconds that an 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.

For example:

```
<gfe:partitioned-region id="examplePartitionRegionWithExpiration">
  <gfe:region-ttl timeout="30000" action="INVALIDATE"/>
  <gfe:entry-tti timeout="600" action="LOCAL_DESTROY"/>
</gfe:replicated-region>
```

For a detailed description of expiration policies, please refer to the GemFire documentation on Expiration.

Annotation-based Data Expiration

With *Spring Data GemFire*, a developer has the ability to define Expiration policies and settings on individual Region Entry values, or rather, application domain objects directly. For instance, a developer might define Expiration settings on a Session-based application domain object like so...

```
@Expiration(timeout = "1800", action = "INVALIDATE")
public class SessionBasedApplicationDomainObject {
  ...
}
```

In addition, a developer may also specify Expiration type specific settings on Region Entries using `@IdleTimeoutExpiration` and `@TimeToLiveExpiration` annotations for Idle Timeout (TTI) and Time-to-Live (TTL) Expiration, respectively...

```
@TimeToLiveExpiration(timeout = "3600", action = "LOCAL_DESTROY")
@IdleTimeoutExpiration(timeout = "1800", action = "LOCAL_INVALIDATE")
@Expiration(timeout = "1800", action = "INVALIDATE")
public class AnotherSessionBasedApplicationDomainObject {
  ...
}
```

Both `@IdleTimeoutExpiration` and `@TimeToLiveExpiration` take precedence over the generic `@Expiration` annotation when more than one Expiration annotation type is specified, as shown above. Though, neither `@IdleTimeoutExpiration` nor `@TimeToLiveExpiration` overrides the other; rather they may compliment each other when different Region Entry Expiration types, such as TTL and TTI, are configured.

NOTE

All @Expiration-based annotations apply only to Region Entry values. Expiration for a "Region" is not covered by *Spring Data GemFire's* Expiration annotation support. However, Pivotal GemFire and *Spring Data GemFire* do allow you to set Region Expiration using the SDG XML namespace, like so...

```
<gfe:*-region id="Example" persistent="false">
  <gfe:region-ttl timeout="600" action="DESTROY"/>
  <gfe:region-tti timeout="300" action="INVALIDATE"/>
</gfe:*-region>
```

Spring Data GemFire's `@Expiration` annotation support is implemented with GemFire's `CustomExpiry` interface. Refer to GemFire's documentation on Configuring Data Expiration for more details

The *Spring Data GemFire* `AnnotationBasedExpiration` class (and `CustomExpiry` implementation) is responsible for processing the SDG `@Expiration` annotations and applying the Expiration policy and settings appropriately for Region Entry Expiration on request.

To use *Spring Data GemFire* to configure specific GemFire Regions to appropriately apply the Expiration policy and settings applied to your application domain objects annotated with `@Expiration`-based annotations, you must...

1. Define a bean in the *Spring* `ApplicationContext` of type `AnnotationBasedExpiration` using the appropriate constructor or one of the convenient factory methods. When configuring Expiration for a specific Expiration type, such as *Idle Timeout* or *Time-to-Live*, then you should use one of the factory methods in the `AnnotationBasedExpiration` class, like so...

```
<bean id="ttlExpiration" class=
"org.springframework.data.gemfire.expiration.AnnotationBasedExpiration"
      factory-method="forTimeToLive"/>

<gfe:partitioned-region id="Example" persistent="false">
    <gfe:custom-entry-ttl ref="ttlExpiration"/>
</gfe:partitioned-region>
```

NOTE	To configure *Idle Timeout* (TTI) Expiration instead, then you would of course use the `forIdleTimeout` factory method along with the `<gfe:custom-entry-tti ref="ttiExpiration"/>` element to set TTI.

2. (optional) Annotate your application domain objects that will be stored in the Region with Expiration policies and custom settings using one of *Spring Data GemFire's* `@Expiration` annotations: `@Expiration`, `@IdleTimeoutExpiration` and/or `@TimeToLiveExpiration`
3. (optional) In cases where particular application domain objects have not been annotated with *Spring Data GemFire's* `@Expiration` annotations at all, but the GemFire Region is configured to use SDG's custom `AnnotationBasedExpiration` class to determine the Expiration policy and settings for objects stored in the Region, then it is possible to set "default" Expiration attributes on the `AnnotationBasedExpiration` bean by doing the following…

```
<bean id="defaultExpirationAttributes" class=
"org.apache.geode.cache.ExpirationAttributes">
    <constructor-arg value="600"/>
    <constructor-arg value="#{T(org.apache.geode.cache.ExpirationAction).DESTROY}"/>
</bean>

<bean id="ttiExpiration" class=
"org.springframework.data.gemfire.expiration.AnnotationBasedExpiration"
      factory-method="forIdleTimeout">
    <constructor-arg ref="defaultExpirationAttributes"/>
</bean>

<gfe:partitioned-region id="Example" persistent="false">
    <gfe:custom-entry-tti ref="ttiExpiration"/>
</gfe:partitioned-region>
```

You may have noticed that *Spring Data GemFire's* `@Expiration` annotations use a String as the attributes type rather than, and perhaps more appropriately, being strongly typed, i.e. `int` for 'timeout' and SDG'S `ExpirationActionType` for 'action'. Why is that?

Well, enter one of *Spring Data GemFire's* other features, leveraging *Spring's* core infrastructure for configuration convenience: *Property Placeholders* and *Spring Expression Language* (SpEL).

For instance, a developer can specify both the Expiration 'timeout' and 'action' using *Property Placeholders* in the `@Expiration` annotation attributes…

```
@TimeToLiveExpiration(timeout = "${geode.region.entry.expiration.ttl.timeout}"
    action = "${geode.region.entry.expiration.ttl.action}")
public class ExampleApplicationDomainObject {
  ...
}
```

Then, in your *Spring* XML config or in JavaConfig, you would declare the following beans…

```
<util:properties id="expirationSettings">
  <prop key="geode.region.entry.expiration.ttl.timeout">600</prop>
  <prop key="geode.region.entry.expiration.ttl.action">INVALIDATE</prop>
  ...
</util:properties>

<context:property-placeholder properties-ref="expirationProperties"/>
```

This is both convenient when multiple application domain objects might share similar Expiration policies and settings, or when you wish to externalize the configuration.

However, a developer may want more dynamic Expiration configuration determined by the state of the running system. This is where the power of SpEL comes in and is the recommended approach, actually. Not only can you refer to beans in the *Spring* context and access bean properties, invoke methods, etc, the values for Expiration 'timeout' and 'action' can be strongly typed. For example (building on the example above)...

```
<util:properties id="expirationSettings">
  <prop key="geode.region.entry.expiration.ttl.timeout">600</prop>
  <prop key="geode.region.entry.expiration.ttl.action"
>#{T(org.springframework.data.gemfire.expiration.ExpirationActionType).DESTROY}</prop>
  <prop key="geode.region.entry.expiration.tti.action"
>#{T(org.apache.geode.cache.ExpirationAction).INVALIDATE}</prop>
  ...
</util:properties>

<context:property-placeholder properties-ref="expirationProperties"/>
```

Then, on your application domain object...

```
@TimeToLiveExpiration(timeout =
"@expirationSettings['geode.region.entry.expiration.ttl.timeout']"
    action = "@expirationSetting['geode.region.entry.expiration.ttl.action']")
public class ExampleApplicationDomainObject {
  ...
}
```

You can imagine that the 'expirationSettings' bean could be a more interesting and useful object rather than a simple instance of `java.util.Properties`. In this example, even the Properties (`expirationSettings`) uses SpEL to base the action value on the actual Expiration action enumerated type leading to more quickly identified failures if the types ever change.

All of this has been demonstrated and tested in the *Spring Data GemFire* test suite, by way of example. See the source for further details.

5.5.9. Data Persistence

Regions can be 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 is recoverable the next time you recreate the Region. This allows data to be recovered after machine or process failure, or even after an orderly shutdown and subsequent restart of the GemFire data node.

To enable persistence with *Spring Data GemFire*, simply set the `persistent` attribute to `true` on any of the `<*-region>` elements. For example…

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

Persistence may also be configured using the `data-policy` attribute; set the attribute's value to one of GemFire's DataPolicy settings. For example…

```
<gfe:partitioned-region id="anotherExamplePersistentPartitionRegion" data-policy=
"PERSISTENT_PARTITION"/>
```

The `DataPolicy` must match the Region type and must also agree with the `persistent` attribute if also explicitly set. An initialization exception will be thrown if the `persistent` attribute is set to `false` yet a persistent `DataPolicy` was specified (e.g. PERSISTENT_REPLICATE, PERSISTENT_PARTITION).

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="yetAnotherExamplePersistentPartitionRegion" persistent=
"true"
    disk-store-ref="myDiskStore" disk-synchronous="true"/>
```

This is discussed further in Configuring a DiskStore

5.5.10. Subscription Policy

GemFire allows configuration of peer-to-peer (P2P) event messaging to control the entry events that the Region will receive. *Spring Data GemFire* provides the `<gfe:subscription/>` sub-element to set the subscription policy on REPLICATE and PARTITION Regions to either `ALL` or `CACHE_CONTENT`.

```
<gfe:partitioned-region id="examplePartitionRegionWithCustomSubscription">
  <gfe:subscription type="CACHE_CONTENT"/>
</gfe:partitioned-region>
```

5.5.11. 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 apply.

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="exampleLocalRegion"/>
```

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`.

5.5.12. Replicated Region

One of the common Region types is a **REPLICATE** Region or **replica**. In short, when a Region is configured to be a REPLICATE, every member that hosts the Region stores a copy of the Region's entries locally. Any update to a REPLICATE 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 *replica*.

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

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

Refer to GemFire's documentation on Distributed and Replicated Regions for more details.

5.5.13. Partitioned Region

Another Region type supported out-of-the-box by the *Spring Data GemFire* namespace is the PARTITION 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 partition 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:

```
<gfe:partitioned-region id="examplePartitionRegion" copies="2" total-buckets="17">
  <gfe:partition-resolver>
    <bean class="example.PartitionResolver"/>
  </gfe:partition-resolver>
</gfe:partitioned-region>
```

Refer to GemFire's documentation on Partitioned Regions for more details.

Partitioned Region Attributes

The following table offers a quick overview of configuration options specific to PARTITION Regions. These are in addition to the common Region configuration options described above.

Table 3. partitioned-region attributes

Name	Values	Description
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 PARTITION Region with which this newly created PARTITION Region is collocated.
local-max-memory	**positive integer**	The maximum amount of memory in megabytes used by the Region in **this** process.
total-max-memory	**any integer value**	The maximum amount of memory in megabytes used by the Region in **all** processes.
partition-listener	**bean name**	The name of the `PartitionListener` used by this Region, for handling partition events.
partition-resolver	**bean name**	The name of the `PartitionResolver` used by this Region, for custom partitioning.
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.

5.5.14. Client Region

Pivotal GemFire supports various deployment topologies for managing and distributing data. GemFire topologies is outside the scope of this documentation. However, to quickly recap, GemFire's supported topologies can be classified in short as: *peer-to-peer* (p2p), *client-server*, and *wide area network* (WAN). In the last two configurations, it is common to declare **client** Regions which connect to a cache server.

Spring Data GemFire offers dedicated support for such configuration through client-cache, `client-region` and `pool` elements. As the names imply, the former defines a client Region while the latter defines a Pool of connections to be used/shared by the various client Regions.

Below is a typical client Region configuration:

```
<bean id="myListener" class="example.CacheListener"/>

<!-- client Region using the default SDG gemfirePool Pool -->
<gfe:client-region id="Example">
  <gfe:cache-listener ref="myListener"/>
</gfe:client-region>

<!-- client Region using its own dedicated Pool -->
<gfe:client-region id="AnotherExample" pool-name="myPool">
  <gfe:cache-listener ref="myListener"/>
</gfe:client-region>

<!-- Pool definition -->
<gfe:pool id="myPool" subscription-enabled="true">
  <gfe:locator host="remoteHost" port="12345"/>
</gfe:pool>
```

As with the other Region types, `client-region` supports `CacheListener``s` as well as a `CacheLoader` and `CacheWriter`. It also requires a connection `Pool` for connecting to either a set of Locators or Servers. Each client Region can have its own Pool or they can share the same one.

NOTE	In the above example, the Pool is configured with `locator`. A Locator is a separate process used to discover cache servers and peer data members 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 (Spring Data GemFire Schema) and GemFire's documentation on Client/Server Configuration.

Client Interests

To minimize network traffic, each client can separately define its own 'interests' policies, indicating to GemFire the data it actually requires. In *Spring Data GemFire*, 'interests' can be defined for each client Region separately. Both Key-based and Regular Expression-based interest types are supported.

For example:

```
<gfe:client-region id="Example" pool-name="myPool">
    <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. The same can be accomplished using a regex of `".\*"`.

The `<gfe:*-interest>` *Key* and *Regular Expression* elements support 3 attributes: `durable`, `receive-values` and `result-policy`.

`durable` indicates whether the 'interest' policy and subscription queue created for the client when the client connects to 1 or more servers in the cluster is maintained across client sessions. If the client goes away and comes back, a "durable" subscription queue on the server(s) for the client is maintained while the client is disconnected, and when the client reconnects, the client will receive any events that occurred while the client was disconnected from the servers(s) in the cluster.

A subscription queue on the servers in the cluster is maintained for each `Pool` of connections defined in the client where subscription has also been "enabled" for that `Pool`. The subscription queue is used to store, and possibly conflate, events sent to the client. If the subscription queue is durable, it persists between client sessions (i.e. connections), potentially up to a specified timeout (if the client does not return within a given time frame in order to reduce resource consumption on servers in the cluster). If the subscription queue is not "durable", then it will be destroyed when the client disconnects. All you need to decide is, for your application use case, is it important for the cache client to receive events while it is disconnected, or is it only important for the application

(cache client) to receive the "latest" events after it reconnects.

The `receive-values` attribute indicates whether or not the entry values are received for create and update events. If **true**, values are received; if **false**, only invalidation events are received.

And finally, the 'result-policy` is an enumeration of: `KEYS`, `KEYS_VALUE` and `NONE`. The default is `KEYS_VALUES`. The `result-policy` controls the initial dump when the client first connects to initialize the local cache, essentially seeding the client with events for all the entries that match the interest policy.

Client-side interests registration does not do much good without enabling subscription on the `Pool` as mentioned above. In fact, it is an error to attempt interests registration without subscription enabled. To do so, you simply...

```
<gfe:pool ... subscription-enabled="true">
  ...
</gfe:pool>
```

In addition to `subscription-enabled`, can you also set `subscription-ack-interval`, `subscription-message-tracking-timeout` and `subscription-redundancy`. `subscription-redundancy` is used to control how many copies of the subscription queue should be maintained by the servers in the cluster. If redundancy is greater than 1, and the "primary" subscription queue (i.e. server) goes down, then a "secondary" subscription queue will take over, keeping the client from missing events in a HA scenario.

In addition to the `Pool` settings, the server-side Regions use an additional attribute, `enable-subscription-conflation`, to control the conflation of events that will be sent to the clients. This can also help further minimize network traffic and is useful in situations where the application only cares about the latest value of an entry. However, in cases where the application is keeping a time series of events that occurred, conflation is going to hinder that use case. The default value is **false**. An example Region configuration on the server for which the client contains a corresponding client [CACHING_]PROXY Region with interests in Keys in this server Region, would look like...

```
<gfe:partitioned-region name="ServerSideRegion" enable-subscription-conflation="true">
  ...
</gfe:partitioned-region>
```

To control the amount of time in seconds that "durable" subscription queue is maintained after a client is disconnected from the server(s) in the cluster, set the `durable-client-timeout` attribute on the `<gfe:client-cache>` element like so...

```
<gfe:client-cache durable-client-timeout="600">
  ...
</gfe:client-cache>
```

A full, in-depth discussion of how client interests work and capabilities is beyond the scope of this

document.

Please refer to Pivotal GemFire's documentation on Client-to-Server Event Distribution for more details.

5.5.15. JSON Support

Pivotal GemFire has support for caching JSON documents in Regions along with the ability to query stored JSON documents using the GemFire OQL. JSON documents are stored internally as PdxInstance types using the JSONFormatter class to perform conversion to and from JSON documents (as a `String`).

Spring Data GemFire provides the `<gfe-data:json-region-autoproxy/>` element to enable a AOP, *Spring* component to advise appropriate, proxied Region operations, which effectively encapsulates the `JSONFormatter`, thereby allowing your applications to work directly with JSON Strings.

In addition, Java objects written to JSON configured Regions will be automatically converted to JSON using Jackson's `ObjectMapper`. Reading these values back will be returned as a JSON String.

By default, `<gfe-data:json-region-autoproxy/>` performs the conversion for all Regions. To apply this feature to selected Regions, provide a comma delimited list of Region bean ids via the `region-refs` attribute. Other attributes include a `pretty-print` flag (defaults to **false**) and `convert-returned-collections`.

Also by default, the results of the `getAll()` and `values()` Region operations will be converted for configured Regions. This is done by creating a parallel data structure in local memory. This can incur significant overhead for large collections, so set the `convert-returned-collections` to **false** if you would like to disable automatic conversion for these Region operations.

NOTE	Certain Region operations, specifically those that use GemFire's proprietary `Region.Entry` such as: `entries(boolean)`, `entrySet(boolean)` and `getEntry()` type are not targeted for AOP advice. In addition, the `entrySet()` method which returns a `Set<java.util.Map.Entry<?, ?>>` is also not affected.

Example configuration:

```
<gfe-data:json-region-autoproxy region-refs="myJsonRegion" pretty-print="true"
convert-returned-collections="false"/>
```

This feature also works seamlessly with `GemfireTemplate` operations, provided that the template is declared as a *Spring* bean. Currently, the native `QueryService` operations are not supported.

5.6. Configuring an Index

Pivotal GemFire allows Indexes (or Indices) to be created to improve the performance of OQL queries.

In *Spring Data GemFire*, Indexes are declared with the `index` element:

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

Before creating the `Index`, *Spring Data GemFire* will verify whether an `Index` with the same name already exists. By default, SDG overrides an existing `Index` if an `Index` with the same name already exists. The existing Index is overridden by removing the old `Index` first followed by creating a new `Index` with the same name defined by the new bean definition, regardless if the old `Index` definition was the same or not.

To prevent the named `Index` definition change, especially when the old and new `Index` definitions differ in a significant way, then set the `override` attribute to **false**, which effectively returns the existing `Index` definition given the same name.

`Index` declarations are not bound to a Region but rather are top-level elements (just like `cache`). This allows one to declare any number of Indices on any Region whether they are just created or already exist - an improvement over GemFire's native `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 XML schema for a full set of options.

5.7. Configuring a DiskStore

Spring Data GemFire supports `DiskStore` configuration via the `disk-store` element.

For example:

```
<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>
```

`DiskStores` are used by Regions for file system persistent backup and overflow of evicted entries as well as persistent backup of WAN Gateways. Multiple GemFire components may share the same `DiskStore`. Additionally, multiple file system directories may be defined for a single `DiskStore`.

Please refer to Pivotal GemFire's documentation for a complete explanation of the configuration options.

5.8. Configuring the Snapshot Service

Spring Data GemFire supports `Cache` and `Region` snapshots using Pivotal GemFire's Snapshot Service. The out-of-the-box Snapshot Service support offers several convenient features to simplify the use of GemFire's Cache and Region Snapshot Service APIs.

As the Pivotal GemFire documentation describes, snapshots allow you to save and subsequently reload the cached data later, which can be useful for moving data between environments, such as from production to a staging or test environment in order to reproduce data-related issues in a

controlled context. You can imagine combining *Spring Data GemFire's* Snapshot Service support with Spring's bean definition profiles to load snapshot data specific to the environment as necessary.

Spring Data GemFire's support for Pivotal GemFire's Snapshot Service begins with the `<gfe-data:snapshot-service>` element from the `<gfe-data>` namespace.

For example, I might want to define Cache-wide snapshots to be loaded as well as saved using a couple snapshot imports and a data export definition as follows:

```
<gfe-data:snapshot-service id="gemfireCacheSnapshotService">
  <gfe-data:snapshot-import location=
"/absolute/filesystem/path/to/import/fileOne.snapshot"/>
  <gfe-data:snapshot-import location=
"relative/filesystem/path/to/import/fileTwo.snapshot"/>
  <gfe-data:snapshot-export
      location="/absolute/or/relative/filesystem/path/to/export/directory"/>
</gfe-data:snapshot-service>
```

You can define as many imports and/or exports as you like. You can define just imports or just exports. The file locations and directory paths can be absolute, or relative to the *Spring Data GemFire* application, JVM process's working directory.

This is a pretty simple example and the Snapshot Service defined in this case refers to the GemFire `Cache` with the default name of `gemfireCache` (as described in Configuring a Cache). If you name your cache bean definition something other than the default, than you can use the `cache-ref` attribute to refer to the cache bean by name:

```
<gfe:cache id="myCache"/>
...
<gfe-data:snapshot-service id="mySnapshotService" cache-ref="myCache">
  ...
</gfe-data:snapshot-service>
```

It is also straightforward to define a Snapshot Service for a particular GemFire Region by specifying the `region-ref` attribute:

```
<gfe:partitioned-region id="Example" persistent="false" .../>
...
<gfe-data:snapshot-service id="gemfireCacheRegionSnapshotService" region-ref="Example
">
  <gfe-data:snapshot-import location="relative/path/to/import/example.snapshot/>
  <gfe-data:snapshot-export location="/absolute/path/to/export/example.snapshot/>
</gfe-data:snapshot-service>
```

When the `region-ref` attribute is specified, *Spring Data GemFire's* `SnapshotServiceFactoryBean` resolves the `region-ref` attribute value to a Region bean defined in the *Spring* context and proceeds

to create a RegionSnapshotService. The snapshot import and export definitions function the same way, however, the `location` must refer to a file on export.

NOTE	GemFire is strict about imported snapshot files actually existing before they are referenced. For exports, GemFire will create the snapshot file if it does not already exist. If the snapshot file for export already exists, the data will be overwritten.

TIP	*Spring Data GemFire* includes a `suppress-import-on-init` attribute on the `<gfe-data:snapshot-service>` element to suppress the configured Snapshot Service from trying to import data into the Cache or Region on initialization. This is useful when data exported from 1 Region is used to feed the import of another Region, for example.

5.8.1. Snapshot Location

For a `Cache`-based Snapshot Service (i.e. CacheSnapshotService) a developer would typically pass it a directory containing all the snapshot files to load rather than individual snapshot files, as the overloaded load method in the `CacheSnapshotService` API indicates.

NOTE	Of course, a developer may use the other, overloaded `load(:File[], :SnapshotFormat, :SnapshotOptions)` method variant to get specific about which snapshot files are to be loaded into the GemFire `Cache`.

However, *Spring Data GemFire* recognizes that a typical developer workflow might be to extract and export data from one environment into several snapshot files, zip all of them up, and then conveniently move the ZIP file to another environment for import.

Therefore, *Spring Data GemFire* enables the developer to specify a JAR or ZIP file on import for a `Cache`-based Snapshot Service as follows:

```
<gfe-data:snapshot-service id="cacheBasedSnapshotService" cache-ref="gemfireCache">
  <gfe-data:snapshot-import location="/path/to/snapshots.zip"/>
</gfe-data:snapshot-service>
```

Spring Data GemFire will conveniently extract the provided ZIP file and treat it like a directory import (load).

5.8.2. Snapshot Filters

The real power of defining multiple snapshot imports and exports is realized through the use of snapshot filters. Snapshot filters implement Pivotal GemFire's SnapshotFilter interface and are used to filter Region entries for inclusion into the Region on import and for inclusion into the snapshot on export.

Spring Data GemFire makes it brain dead simple to utilize snapshot filters on import and export using the `filter-ref` attribute or an anonymous, nested bean definition:

```
<gfe:cache/>

<gfe:partitioned-region id="Admins" persistent="false"/>
<gfe:partitioned-region id="Guests" persistent="false"/>

<bean id="activeUsersFilter" class="
example.gemfire.snapshot.filter.ActiveUsersFilter/>

<gfe-data:snapshot-service id="adminsSnapshotService" region-ref="Admins">
  <gfe-data:snapshot-import location="/path/to/import/users.snapshot">
    <bean class="example.gemfire.snapshot.filter.AdminsFilter/>
  </gfe-data:snapshot-import>
  <gfe-data:snapshot-export location="/path/to/export/active/admins.snapshot" filter-
ref="activeUsersFilter"/>
</gfe-data:snapshot-service>

<gfe-data:snapshot-service id="guestsSnapshotService" region-ref="Guests">
  <gfe-data:snapshot-import location="/path/to/import/users.snapshot">
    <bean class="example.gemfire.snapshot.filter.GuestsFilter/>
  </gfe-data:snapshot-import>
  <gfe-data:snapshot-export location="/path/to/export/active/guests.snapshot" filter-
ref="activeUsersFilter"/>
</gfe-data:snapshot-service>
```

In addition, more complex snapshot filters can be expressed with the `ComposableSnapshotFilter` *Spring Data GemFire* provided class. This class implements GemFire's SnapshotFilter interface as well as the Composite software design pattern.

In a nutshell, the Composite software design pattern allows developers to compose multiple objects of the same type and treat the aggregate as single instance of the object type, a very powerful and useful abstraction.

`ComposableSnapshotFilter` has two factory methods, `'and'` and `'or'`, allowing developers to logically combine individual snapshot filters using the AND and OR logical operators, respectively. The factory methods take a list of `SnapshotFilters`.

In this case, the developer is only limited by his/her imagination to leverage this powerful construct.

For instance:

```
<bean id="activeUsersSinceFilter" class=
"org.springframework.data.gemfire.snapshot.filter.ComposableSnapshotFilter"
      factory-method="and">
  <constructor-arg index="0">
    <list>
      <bean class="org.example.app.gemfire.snapshot.filter.ActiveUsersFilter"/>
      <bean class="org.example.app.gemfire.snapshot.filter.UsersSinceFilter"
            p:since="2015-01-01"/>
    </list>
  </constructor-arg>
</bean>
```

The developer could then go onto combine the `activesUsersSinceFilter` with another filter using `'or'` like so:

```
<bean id="covertOrActiveUsersSinceFilter" class=
"org.springframework.data.gemfire.snapshot.filter.ComposableSnapshotFilter"
      factory-method="or">
  <constructor-arg index="0">
    <list>
      <ref bean="activeUsersSinceFilter"/>
      <bean class="example.gemfire.snapshot.filter.CovertUsersFilter"/>
    </list>
  </constructor-arg>
</bean>
```

5.8.3. Snapshot Events

By default, *Spring Data GemFire* uses Pivotal GemFire's Snapshot Services on startup to import data and shutdown to export data. However, you may want to trigger periodic, event-based snapshots, for either import or export from within your *Spring* application.

For this purpose, *Spring Data GemFire* defines two additional *Spring* application events, extending *Spring's* ApplicationEvent class for imports and exports, respectively: `ImportSnapshotApplicationEvent` and `ExportSnapshotApplicationEvent`.

The two application events can be targeted at the entire GemFire Cache, or individual GemFire Regions. The constructors in these classes accept an optional Region pathname (e.g. "/Example") as well as 0 or more `SnapshotMetadata` instances.

The array of `SnapshotMetadata` is used to override the snapshot meta-data defined by `<gfe-data:snapshot-import>` and `<gfe-data:snapshot-export>` sub-elements in XML, which will be used in cases where snapshot application events do not explicitly provide `SnapshotMetadata`. Each individual `SnapshotMetadata` instance can define it's own `location` and `filters` properties.

Import/export snapshot application events are received by all snapshot service beans defined in the *Spring* `ApplicationContext`. However, import/export events are only processed by "matching" Snapshot Service beans.

A Region-based `[Import|Export]SnapshotApplicationEvent` matches if the Snapshot Service bean defined is a `RegionSnapshotService` and it's Region reference (as determined by the `region-ref` attribute) matches the Region's pathname specified by the snapshot application event.

A Cache-based `[Import|Export]SnapshotApplicationEvent` (i.e. a snapshot application event without a Region pathname) triggers all Snapshot Service beans, including any `RegionSnapshotService` beans, to perform either an import or export, respectively.

It is very easy to use *Spring's* ApplicationEventPublisher interface to fire import and/or export snapshot application events from your application like so:

```
@Component
public class ExampleApplicationComponent {

  @Autowired
  private ApplicationEventPublisher eventPublisher;

  @Resource(name = "Example")
  private Region<?, ?> example;

  public void someMethod() {
    ...

    SnapshotFilter myFilter = ...;

    SnapshotMetadata exportSnapshotMetadata = new SnapshotMetadata(new File(System
.getProperty("user.dir"),
      "/path/to/export/data.snapshot"), myFilter, null);

    eventPublisher.publishEvent(new ExportSnapshotApplicationEvent(this, example
.getFullPath(), exportSnapshotMetadata);

    ...
  }
}
```

In this particular example, only the "/Example" Region's Snapshot Service bean will pick up and handle the export event, saving the filtered, "/Example" Region's data to the "data.snapshot" file in a sub-direcrtory of the application's working directory.

Using *Spring* application events and messaging subsystem is a good way to keep your application loosely coupled. It is also not difficult to imagine that the snapshot application events could be fired on a periodic basis using *Spring's* Scheduling services.

5.9. Configuring the Function Service

Spring Data GemFire provides annotation support for implementing and registering Pivotal GemFire Functions.

Spring Data GemFire also provides namespace support for registering Pivotal GemFire Functions for remote Function execution.

Please refer to Pivotal GemFire' documentation for more information on the Function execution framework.

GemFire Functions are declared as *Spring* beans and must implement the `org.apache.geode.cache.execute.Function` interface or extend `org.apache.geode.cache.execute.FunctionAdapter`.

The namespace uses a familiar pattern to declare functions:

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

<bean id="function2" class="example.FunctionTwo"/>
```

5.10. Configuring WAN Gateways

WAN Gateways provide a way to synchronize Pivotal GemFire Distributed Systems across geographic areas. *Spring Data GemFire* provides namespace support for configuring WAN Gateways as illustrated in the following examples.

5.10.1. WAN Configuration in GemFire 7.0

In the example below, `GatewaySenders` are configured for a PARTITION Region by adding child elements to the Region (`gateway-sender` and `gateway-sender-ref`).

A `GatewaySender` may register `EventFilters` and `TransportFilters`. Also shown below is an example configuration of an `AsyncEventQueue` which must also be wired into a Region (not shown).

```
<gfe:partitioned-region id="region-with-inner-gateway-sender" >
    <gfe:gateway-sender remote-distributed-system-id="1">
        <gfe:event-filter>
            <bean class="org.springframework.data.gemfire.example.SomeEventFilter"/>
        </gfe:event-filter>
        <gfe:transport-filter>
            <bean class="org.springframework.data.gemfire.example.SomeTransportFilter
"/>
        </gfe:transport-filter>
    </gfe:gateway-sender>
    <gfe:gateway-sender-ref bean="gateway-sender"/>
</gfe:partitioned-region>

<gfe:async-event-queue id="async-event-queue" batch-size="10" persistent="true" disk-
store-ref="diskstore"
        maximum-queue-memory="50">
    <gfe:async-event-listener>
        <bean class="example.AsyncEventListener"/>
    </gfe:async-event-listener>
</gfe:async-event-queue>

<gfe:gateway-sender id="gateway-sender" remote-distributed-system-id="2">
    <gfe:event-filter>
        <ref bean="event-filter"/>
        <bean class="org.springframework.data.gemfire.example.SomeEventFilter"/>
    </gfe:event-filter>
    <gfe:transport-filter>
        <ref bean="transport-filter"/>
        <bean class="org.springframework.data.gemfire.example.SomeTransportFilter"/>
    </gfe:transport-filter>
</gfe:gateway-sender>

<bean id="event-filter" class=
"org.springframework.data.gemfire.example.AnotherEventFilter"/>
<bean id="transport-filter" class=
"org.springframework.data.gemfire.example.AnotherTransportFilter"/>
```

On the other end of a `GatewaySender` is a corresponding `GatewayReceiver` to receive Gateway events. The `GatewayReceiver` may also be configured with `EventFilters` and `TransportFilters`.

```
<gfe:gateway-receiver id="gateway-receiver" start-port="12345" end-port="23456" bind-
address="192.168.0.1">
    <gfe:transport-filter>
        <bean class="org.springframework.data.gemfire.example.SomeTransportFilter"/>
    </gfe:transport-filter>
</gfe:gateway-receiver>
```

Please refer to the Pivotal GemFire documentation for a detailed explanation of all the

configuration options.

Chapter 6. Working with Pivotal GemFire APIs

Once the Pivotal GemFire Cache and Regions have been configured, they can be injected and used inside application objects. This chapter describes the integration with *Spring's* Transaction Management functionality and DAO exception hierarchy. This chapter also covers support for dependency injection of GemFire managed objects.

6.1. GemfireTemplate

As with many other high-level abstractions provided by *Spring* projects, *Spring Data GemFire* provides a **template** to simplify GemFire data access. The class provides several **one-liner** methods containing common Region operations, but also has the ability to **execute** code against the native GemFire API without having to deal with GemFire checked exceptions by using a `GemfireCallback`.

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

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

Once the template is configured, a developer 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 region) throws GemFireCheckedException,
GemFireException {
        Region<String, String> localRegion = (Region<String, String>) region;

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

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

For accessing the full power of the Pivotal GemFire query language, a developer can use the `find` and `findUnique` methods, 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.

6.2. Exception Translation

Using a new data access technology requires not only accommodating a new API but also handling exceptions specific to that technology.

To accommodate the exception handling case, the *Spring Framework* provides a technology agnostic and consistent exception hierarchy that abstracts the application from proprietary, and usually "checked", exceptions to a set of focused runtime exceptions.

As mentioned in *Spring Framework's* documentation, Exception translation can be applied transparently to your Data Access Objects (DAO) 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 the `CacheFactoryBean` is declared, e.g. using either a `<gfe:cache/>` or `<gfe:client-cache>` declaration, which acts as an exception translator and is automatically detected by the *Spring* infrastructure and used accordingly.

6.3. Transaction Management

One of the most popular features of the *Spring Framework* is Transaction Management.

If you are not familiar with *Spring's* transaction abstraction then we strongly recommend reading about it as it offers a *consistent programming model* that works transparently across multiple APIs and can be configured either programmatically or declaratively (the most popular choice).

For Pivotal GemFire, *Spring Data GemFire* provides a dedicated, per-cache, `PlatformTransactionManager` that, once declared, allows Region operations to be executed atomically through *Spring*:

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

NOTE

The example above can be simplified even further 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 bean name is not configured, the aforementioned naming convention will be used. Additionally, the transaction manager name is `"gemfireTransactionManager"` if not explicitly specified.

Currently, Pivotal GemFire supports optimistic transactions with **read committed** isolation. Furthermore, to guarantee this isolation, developers should avoid making **in-place** changes that manually modify values present in the cache. To prevent this from happening, the transaction manager configures 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 and bevior of the underlying GemFire transaction manager, please refer to the GemFire CacheTransactionManager Javadoc as well as the documentation.

6.4. Continuous Query (CQ)

A powerful functionality offered by Pivotal GemFire is Continuous Query (or CQ). In short, CQ allows one to create and register an OQL query, and then automatically be notified when new data that gets added to GemFire matches the query predicate. *Spring Data 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 the *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 called to be notified when there is a match. *Spring Data GemFire* takes care of the rest. This is very similar to Java EE's message-driven bean style, but without any requirement for base class or interface implementations, based on Pivotal GemFire.

NOTE	Currently, Continuous Query is only supported in GemFire's client/server topology. Additionally, the client Pool used is required to have the subscription enabled. Please refer to the GemFire documentation for more information.

6.4.1. Continuous Query Listener Container

Spring Data GemFire simplifies creation, registration, life-cycle and dispatch of CQ events by taking care of the infrastructure around CQ with the use of SDG's `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 provided in the *Spring Framework* with its Message-driven POJOs (MDPs).

The SDG `ContinuousQueryListenerContainer` acts as an event (or message) listener container; it is used to receive the events from the registered CQs and invoke 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 delegate the boilerplate GemFire infrastructure concerns to the framework.

The listener container is fully customizable. A developer can chose either to use the CQ thread to perform the dispatch (synchronous delivery) or a new thread (from an existing pool) 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, the developer 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 proper `TaskExecutor` to take advantage of its runtime.

6.4.2. The `ContinuousQueryListener` and `ContinuousQueryListenerAdapter`

The `ContinuousQueryListenerAdapter` class is the final component in *Spring Data GemFire* CQ support. In a nutshell, class allows you to expose almost **any** implementing class as an EDP with

minimal constraints. `ContinuousQueryListenerAdapter` implements the `ContinuousQueryListener` interface, a simple listener interface similar to GemFire's 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 throwable);
     void handleQuery(CqQuery cq);
     void handleEvent(CqEvent event, Operation baseOp, byte[] deltaValue);
     void handleEvent(CqEvent event, Operation baseOp, Operation queryOp, Object key,
Object newValue);
}
```

```
package example;

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

In particular, note how the above implementation of the `EventDelegate` interface has **no** GemFire dependencies at all. It truly is a POJO that we can and will make into an EDP via the following configuration.

NOTE	the class does not have to implement an interface; an interface is only used to better showcase the decoupling between the contract and the implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:gfe="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
">

    <gfe:client-cache/>

    <gfe:pool 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 /SomeRegion"/>
       <gfe:listener ref="another-listener" query="SELECT * FROM /AnotherRegion" name
="myQuery" method="handleQuery"/>
    </gfe:cq-listener-container>

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

NOTE

The example above shows a few 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.ContinuousQueryListenerAdapter">
    <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.ContinuousQueryListenerContainer">
    <property name="cache" ref="gemfireCache"/>
    <property name="queryListeners">
      <!-- set of CQ listeners -->
      <set>
        <bean class=
"org.springframework.data.gemfire.listener.ContinuousQueryDefinition" >
              <constructor-arg value="SELECT * FROM /SomeRegion" />
              <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).

6.5. Wiring `Declarable` Components

Pivotal GemFire XML configuration (usually referred to as `cache.xml`) allows **user** objects to be declared as part of the configuration. Usually these objects are `CacheLoaders` 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 when 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 collaborators.

However, if you are starting a green field project, it is recommended that you configure Cache, Region, and other pluggable GemFire 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

A developer can configure custom types entirely through *Spring* as mentioned in Configuring a Region. That way, a developer 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 auto-wiring through the *Spring* IoC 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 Pivotal GemFire.

6.5.1. Configuration using template bean definitions

When used, `WiringDeclarableSupport` tries to first locate an existing bean definition and use that as the 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:

```
class DBLoader extends WiringDeclarableSupport implements CacheLoader {

  private DataSource dataSource;

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:p="http://www.springframework.org/schema/p"
       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
">

  <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:p="http://www.springframework.org/schema/p"
       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
">

  <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).

6.5.2. Configuration using auto-wiring and annotations

By default, if no bean definition is found, `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, a developer can also use JDK 5 annotations to provide additional information to the auto-wiring process.

TIP 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:

```
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 since the class is auto-wired -->
</cache-loader>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       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/context
http://www.springframework.org/schema/context/spring-context.xsd
">

     <!-- enable annotation processing -->
     <context:annotation-config/>

</beans>
```

By using the JSR-330 annotations, the `CacheLoader` 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 easily be configured and changed through the *Spring* container without touching the `DBLoader` code.

6.6. Support for Spring Cache Abstraction

Spring Data GemFire provides an implementation of the *Spring* Cache Abstraction to position Pivotal GemFire as a *caching provider* in Spring's caching infrastructure.

To use Pivotal GemFire as a backing implementation, simply add `GemfireCacheManager` to your configuration:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:cache="http://www.springframework.org/schema/cache"
  xmlns:gfe="http://www.springframework.org/schema/gemfire"
  xmlns:p="http://www.springframework.org/schema/p"
  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
    http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache.xsd">

  <!-- enable declarative caching -->
  <cache:annotation-driven/>

  <gfe:cache id="gemfire-cache"/>

  <!-- declare GemfireCacheManager; must have a bean ID of 'cacheManager' -->
  <bean id="cacheManager" class=
"org.springframework.data.gemfire.cache.GemfireCacheManager"
      p:cache-ref="gemfire-cache">

</beans>
```

NOTE	The `cache-ref` attribute on the `CacheManager` bean definition is not necessary if the default cache bean name is used (i.e. "gemfireCache"), that is, `<gfe:cache>` without an explicit ID.

When the `GemfireCacheManager` (Singleton) bean instance is declared and declarative caching is enabled (either in XML with `<cache:annotation-driven/>` or in JavaConfig with *Spring's* `@EnableCaching` annotation), the *Spring* caching annotations (e.g. `@Cacheable`) identify the "caches" that will cache data in-memory using GemFire Regions.

These caches (i.e. Regions) must exist before the caching annotations that use them otherwise an error will occur.

By way of example, suppose you have a Customer Service application with a `CustomerService` application component that does caching…

```
@Service
class CustomerService {

@Cacheable(cacheNames="Accounts", key="#customer.id")
Account createAccount(Customer customer) {
  ...
}
```

Then you will need the following config.

XML:

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:cache="http://www.springframework.org/schema/cache"
  xmlns:gfe="http://www.springframework.org/schema/gemfire"
  xmlns:p="http://www.springframework.org/schema/p"
  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
    http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache.xsd">

  <!-- enable declarative caching -->
  <cache:annotation-driven/>

  <bean id="cacheManager" class=
"org.springframework.data.gemfire.cache.GemfireCacheManager">

  <gfe:cache/>

  <gfe:partitioned-region id="accontsRegion" name="Accounts" persistent="true" ...>
    ...
  </gfe:partitioned-region>
</beans>
```

JavaConfig:

```
@Configuration
@EnableCaching
class ApplicationConfiguration {

  @Bean
  CacheFactoryBean gemfireCache() {
    return new CacheFactoryBean();
  }

  @Bean
  GemfireCacheManager cacheManager() {
    return new GemfireCacheManager(gemfireCache());
  }

  @Bean("Accounts")
  PartitionedRegionFactoryBean accountsRegion() {
    PartitionedRegionFactoryBean accounts = new PartitionedRegionFactoryBean();

    accounts.setCache(gemfireCache());
    accounts.setClose(false);
    accounts.setPersistent(true);

    return accounts;
  }
}
```

Of course, you are free to choose whatever Region type you like (e.g. REPLICATE, PARTITION, LOCAL, etc).

For more details on *Spring's Cache Abstraction*, again, please refer to the documentation.

Chapter 7. Working with Pivotal GemFire Serialization

To improve overall performance of the Pivotal GemFire In-memory Data Grid, GemFire supports a dedicated serialization protocol, called PDX, that is both faster and offers more compact results over standard Java serialization in addition to works transparently across various language platforms (Java, C++, .NET). Please refer to PDX Serialization Features and PDX Serialization Internals for more details.

This chapter discusses the various ways in which *Spring Data GemFire* simplifies and improves GemFire's 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 system or environment where it lives at a certain point in time. For instance, a `DataSource` is environment specific. 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 falling back to autowiring otherwise.

Please refer to the previous section (Wiring `Declarable` Components) for more details on wiring functionality.

To use this SDG `Instantiator`, simply declare it as a 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 *Spring* container startup, once it is being initialized, the `Instantiator` will, by default, register itself with the GemFire serialization 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="instantiatorFactory" class=
"org.springframework.data.gemfire.serialization.InstantiatorFactoryBean">
  <property name="customTypes">
    <map>
      <entry key="org.pkg.CustomTypeA" value="1025"/>
      <entry key="org.pkg.CustomTypeB" value="1026"/>
    </map>
  </property>
</bean>
```

The definition above, automatically generates two `Instantiators` for two classes, namely `CustomTypeA` and `CustomTypeB` and registers them with GemFire, under user id `1025` and `1026`. The two `Instantiators` avoid the use of reflection and create the instances directly through Java code.

Chapter 8. POJO mapping

8.1. Entity Mapping

Spring Data GemFire provides support to map entities that will be stored in a Region in the GemFire In-Memory Data Grid. The mapping metadata is defined using annotations on application domain classes just like this:

Example 1. Mapping a domain class to a GemFire Region

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

  @Id Long id;

  String firstname;
  String lastname;

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

  ···
}
```

The first thing you notice here is the `@Region` annotation that can be used to customize the Region in which an instance of the `Person` class is stored. The `@Id` annotation can be used to annotate the property that shall be used as the cache (Region) key, identifying the Region entry. The `@PersistenceConstructor` annotation helps to disambiguate multiple, potentially available constructors taking parameters and explicitly marking the constructor annotated as the constructor to be used to construct entities. In an application domain class with no or only a single constructor you can omit the annotation.

In addition to storing entities in top-level Regions, entities can be stored in Sub-Regions as well.

For instance:

```
@Region("/Users/Admin")
public class Admin extends User {
  ...
}

@Region("/Users/Guest")
public class Guest extends User {
  ...
}
```

Be sure to use the full-path of the GemFire Region, as defined with the *Spring Data GemFire* XML namespace using the `id` or `name` attributes of the `<*-region>` element.

8.1.1. Entity Mapping by Region Type

In addition to the `@Region` annotation, *Spring Data GemFire* also recognizes the Region type-specific mapping annotations: `@ClientRegion`, `@LocalRegion`, `@PartitionRegion` and `@ReplicateRegion`.

Functionally, these annotations are treated exactly the same as the generic `@Region` annotation in the SDG mapping infrastructure. However, these additional mapping annotations are useful in *Spring Data GemFire's` Annotation configuration model. When combined with the* `@EnableEntityDefinedRegions` *configuration annotation on _Spring* `@Configuration` annotated class, it is possible to generate Regions in the local cache, whether the application is a client or peer.

These annotations allow you, the developer, to be more specific about what type of Region that your application entity class should be mapped to, and also has an impact on the data management policies of the Region (e.g. partition (a.k.a. sharding) vs. just replicating data).

Using these Region type-specific mapping annotations with the SDG Annotation config model saves you from having to explicitly define these Regions in config.

The details of the new Annotation configuration model will be discussed in more detail in a subsequent releaase.

8.1.2. Repository Mapping

As an alternative to specifying the Region in which the entity will be stored using the `@Region` annotation on the entity class, you can also specify the `@Region` annotation on the entity's `Repository`. See Spring Data GemFire Repositories for more details.

However, let's say you want to store a `Person` in multiple GemFire Regions (e.g. `People` and `Customers`), then you can define your corresponding `Repository` interface extensions like so:

```
@Region("People")
public interface PersonRepository extends GemfireRepository<Person, String> {
...
}

@Region("Customers")
public interface CustomerRepository extends GemfireRepository<Person, String> {
...
}
```

Then, using each Repository individually, you can store the entity in multiple GemFire Regions.

```
@Service
class CustomerService {

  CustomerRepository customerRepo;

  PersonRepository personRepo;

  Customer update(Customer customer) {
    customerRepo.save(customer);
    personRepo.save(customer);
    return customer;
  }
```

It is not difficult to imagine wrapping the `update` service method in a *Spring* managed transaction, either as a local cache transaction or a global transaction.

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 default constructor, a singly declared constructor or an explicitly annotated constructor annotated with the `@PersistenceConstructor` annotation).

To provide values for constructor parameters it will read fields with name of the constructor parameters from the supplied PdxReader.

Example 2. Using @Value on entity constructor parameters

```
public class Person {

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

An entity class annotated in this way will have the field `foo` read from the `PdxReader` and passed to the constructor parameter value for `firstname`. The value for `lastname` will be the *Spring* bean with the name `bean`.

Chapter 9. Spring Data GemFire Repositories

9.1. Introduction

Spring Data GemFire provides support to use the *Spring Data Repository* abstraction to easily persist entities into GemFire along with execute queries. A general introduction to the *Repository programming model* is provided here.

9.2. Spring Configuration

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

Example 3. Bootstrap Spring Data GemFire Repositories

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:gfe-data="http://www.springframework.org/schema/data/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/data/gemfire
http://www.springframework.org/schema/data/gemfire/spring-data-gemfire.xsd>

  <gfe-data:repositories base-package="com.example.acme.repository"/>

</beans>
```

This configuration snippet looks for interfaces below the configured base package and creates Repository instances for those interfaces backed by a `SimpleGemFireRepository`.

IMPORTANT	You must have your application domain classes correctly mapped to configured Regions or the bootstrap process will fail otherwise.

9.3. Executing OQL Queries

Spring Data GemFire Repositories enable the definition of query methods to easily execute GemFire OQL Queries against the Region the managed entity is mapped to.

Example 4. Sample Repository

```
@Region("People")
public class Person { ⋯ }
```

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

  Person findByEmailAddress(String emailAddress);

  Collection<Person> findByFirstname(String firstname);

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

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

The first query method listed here will cause the following OQL query to be derived: `SELECT x FROM /People x WHERE x.emailAddress = $1`. The second query method works the same way except it's returning all entities found whereas the first query method expects a single result to be found.

In case the supported keywords are not sufficient to expresss and declare your OQL query, or the method name becomes too verbose, you can annotate the query methods with `@Query` as seen for methods 3 and 4.

Table 4. 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`
`IgnoreCase`	`findByFirstnameIgnoreCase(String firstName)`	`x.firstname.equalsIgnoreCase($1)`
(No keyword)	`findByFirstname(String name)`	`x.firstname = $1`
`Like`	`findByFirstnameLike(String name)`	`x.firstname LIKE $1`
`Not`	`findByFirstnameNot(String name)`	`x.firstname != $1`
`IsTrue`, `True`	`findByActiveIsTrue()`	`x.active = true`

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

9.4. OQL Query Extensions using Annotations

Many query languages, such as Pivotal GemFire's OQL (Object Query Language), have extensions that are not directly supported by *Spring Data Commons' Repository* infrastructure.

One of *Spring Data Commons' Repository* infrastructure goals is to function as the lowest common denominator in order to maintain support for and portability across the widest array of data stores available and in use for application development today. Technically, this means developers can access multiple different data stores supported by *Spring Data Commons* within their applications by reusing their existing application-specific Repository interfaces, a very convenient and powerful abstraction.

To support GemFire's OQL Query language extensions and preserve portability across different data stores, *Spring Data GemFire* adds support for OQL Query extensions using Java Annotations. These Annotations will be ignored by other *Spring Data Repository* implementations (e.g. *Spring Data* JPA or *Spring Data Redis*) that do not have similar query language extensions.

For instance, many data stores will most likely not implement GemFire's OQL `IMPORT` keyword. By implementing `IMPORT` as an Annotation (i.e. `@Import`) rather than as part of the query method signature (specifically, the method 'name'), then this will not interfere with the parsing infrastructure when evaluating the query method name to construct another data store language appropriate query.

Currently, the set of GemFire OQL Query language extensions that are supported by *Spring Data GemFire* include:

Table 5. Supported GemFire OQL extensions for Repository query methods

Keyword	Annotation	Description	Arguments
HINT	`@Hint`	OQL Query Index Hints	`String[]` (e.g. @Hint({ "IdIdx", "TxDateIdx" }))
IMPORT	`@Import`	Qualify application-specific types.	`String` (e.g. @Import("org.example.app.domain.Type"))
LIMIT	`@Limit`	Limit the returned query result set.	`Integer` (e.g. @Limit(10); default is Integer.MAX_VALUE)
TRACE	`@Trace`	Enable OQL Query specific debugging.	NA

As an example, suppose you have a `Customers` application domain class and corresponding GemFire Region along with a `CustomerRepository` and a query method to lookup `Customers` by last name, like so...

Example 5. Sample Customers Repository

```
package ...;

import org.springframework.data.annotation.Id;
import org.springframework.data.gemfire.mapping.annotation.Region;
...

@Region("Customers")
public class Customer ... {

  @Id
  private Long id;

  ...
}
```

```
package ...;

import org.springframework.data.gemfire.repository.GemfireRepository;
...

public interface CustomerRepository extends GemfireRepository<Customer, Long> {

  @Trace
  @Limit(10)
  @Hint("LastNameIdx")
  @Import("org.example.app.domain.Customer")
  List<Customer> findByLastName(String lastName);

  ...
}
```

This will result in the following OQL Query:

```
<TRACE> <HINT 'LastNameIdx'> IMPORT org.example.app.domain.Customer; SELECT * FROM /Customers x
WHERE x.lastName = $1 LIMIT 10
```

Spring Data GemFire's Repository extension and support is careful not to create conflicting declarations when the OQL Annotation extensions are used in combination with the `@Query` annotation.

As another example, suppose you have a raw `@Query` annotated query method defined in your `CustomerRepository` like so...

Example 6. CustomerRepository

```
public interface CustomerRepository extends GemfireRepository<Customer, Long> {

  @Trace
  @Limit(10)
  @Hint("CustomerIdx")
  @Import("org.example.app.domain.Customer")
  @Query("<TRACE> <HINT 'ReputationIdx'> SELECT DISTINCT * FROM /Customers c WHERE
c.reputation > $1 ORDER BY c.reputation DESC LIMIT 5")
  List<Customer>
findDistinctCustomersByReputationGreaterThanOrderByReputationDesc(Integer
reputation);
}
```

This query method results in the following OQL Query:

```
IMPORT org.example.app.domain.Customer; <TRACE> <HINT 'ReputationIdx'> SELECT DISTINCT * FROM
/Customers x WHERE x.reputation > $1 ORDER BY c.reputation DESC LIMIT 5
```

As you can see, the `@Limit(10)` annotation will not override the `LIMIT` defined explicitly in the raw query. As well, `@Hint("CustomerIdx")` annotation does not override the `HINT` explicitly defined in the raw query. Finally, the `@Trace` annotation is redundant and has no additional effect.

NOTE

The "ReputationIdx" Index is probably not the most sensible index given the number of Customers who will possibly have the same value for their reputation, which will effectively reduce the effectiveness of the index. Please choose indexes and other optimizations wisely as an improper or poorly choosen index can have the opposite effect on your performance given the overhead in maintaining the index. The "ReputationIdx" was only used to serve the purpose of the example.

Chapter 10. Annotation Support for Function Execution

10.1. Introduction

Spring Data GemFire includes annotation support to simplify working with GemFire Function Execution. Under-the-hood, the Pivotal GemFire API provides classes to implement and register GemFire Functions that are deployed on GemFire servers, which may then be invoked by other peer member applications or remotely from cache clients.

Functions can execute in parallel, distributed among multiple GemFire servers in the cluster, aggregating results with the map-reduce pattern that are sent back to the caller. Functions can also be targeted to run on a single server or Region. The Pivotal GemFire API supports remote execution of Functions targeted using various predefined scopes: on Region, on members [in groups], on servers, etc. The implementation and execution of remote Functions, as with any RPC protocol, requires some boilerplate code.

Spring Data GemFire, true to *Spring's* core value proposition, aims to hide the mechanics of remote Function execution and allow developers to focus on core POJO programming and business logic. To this end, *Spring Data GemFire* introduces annotations to declaratively register public methods of a POJO class as GemFire Functions along with the ability to invoke registered Functions [remotely] via annotated interfaces.

10.2. Implementation vs Execution

There are two separate concerns to address implementation and execution.

First is Function implementation (server-side), which must interact with the FunctionContext to access the invocation arguments, ResultsSender as well as other execution context information. The Function implementation typically accesses the Cache and/or Regions and is registered with the FunctionService under a unique Id.

A cache client application invoking a Function does not depend on the implementation. To invoke a Function, the application instantiates an Execution providing the Function ID, invocation arguments and the Function target, which defines its scope: Region, server, servers, member or members. If the Function produces a result, the invoker uses a ResultCollector to aggregate and acquire the execution results. In certain cases, a custom `ResultCollector` implementation is required and may be registered with the `Execution`.

NOTE	'Client' and 'Server' are used here in the context of Function execution, which may have a different meaning than client and server in GemFire's client-server topology. While it is common for an application using a `ClientCache` to invoke a Function on one or more GemFire servers in a cluster, it is also possible to execute Functions in a peer-to-peer (P2P) configuration, where the application is a member of the cluster hosting a peer `Cache`. Keep in mind that a peer member cache application is subject to all the same constraints of being a peer member of the cluster.

10.3. Implementing a Function

Using GemFire APIs, the `FunctionContext` provides a runtime invocation context that includes the client's calling arguments and a `ResultSender` implementation to send results back to the client. Additionally, if the Function is executed on a Region, the `FunctionContext` is actually an instance of `RegionFunctionContext`, which provides additional information such as the target Region on which the Function was invoked and any Filter (set of specific keys) associated with the `Execution`, etc. If the Region is a PARTITION Region, the Function should use the `PartitionRegionHelper` to extract only the local data.

Using *Spring*, a developer can write a simple POJO and use the *Spring* container to bind one or more of it's public methods to a Function. The signature for a POJO method intended to be used as a Function must generally conform to the client's execution arguments. However, in the case of a Region execution, the Region data may also be provided (presumably the data held in the local partition if the Region is a PARTITION Region). Additionally, the Function may require the Filter that was applied, if any. This suggests that the client and server share a contract for the calling arguments but that the method signature may include additional parameters to pass values provided by the `FunctionContext`. One possibility is for the client and server to share a common interface, but this is not strictly required. The only constraint is that the method signature includes the same sequence of calling arguments with which the Function was invoked after the additional parameters are resolved.

For example, suppose the client provides a String and int as the calling arguments. These are provided in the `FunctionContext` as an array:

```
Object[] args = new Object[] { "test", 123 };
```

Then, the *Spring* container should be able to bind to any method signature similar to the following. Let's ignore the return type for the moment:

```
public Object method1(String s1, int i2) {...}
public Object method2(Map<?, ?> data, String s1, int i2) {...}
public Object method3(String s1, Map<?, ?> data, int i2) {...}
public Object method4(String s1, Map<?, ?> data, Set<?> filter, int i2) {...}
public void method4(String s1, Set<?> filter, int i2, Region<?,?> data) {...}
public void method5(String s1, ResultSender rs, int i2);
public void method6(FunctionContest context);
```

The general rule is that once any additional arguments, i.e. Region data and Filter, are resolved, the remaining arguments must correspond exactly, in order and type, to the expected Function method parameters. The method's return type must be void or a type that may be serialized (either as a `java.io.Serializable`, `DataSerializable` or `PdxSerializable`). The latter is also a requirement for the calling arguments. The Region data should normally be defined as a Map, to facilitate unit testing, but may also be of type Region if necessary. As shown in the example above, it is also valid to pass the `FunctionContext` itself, or the `ResultSender`, if you need to control how the results are returned to the client.

10.3.1. Annotations for Function Implementation

The following example illustrates how SDG's Function annotations are used to expose POJO methods as GemFire Functions:

```
@Component
public class ApplicationFunctions {

   @GemfireFunction
   public String function1(String value, @RegionData Map<?, ?> data, int i2) { ... }

   @GemfireFunction("myFunction", batchSize=100, HA=true, optimizedForWrite=true)
   public List<String> function2(String value, @RegionData Map<?, ?> data, int i2,
@Filter Set<?> keys) { ... }

   @GemfireFunction(hasResult=true)
   public void functionWithContext(FunctionContext functionContext) { ... }

}
```

Note, the class itself must be registered as a *Spring* bean and each GemFire Function is annotated with `@GemfireFunction`. In this example, *Spring's* `@Component` annotation was used, but you may register the bean by any method supported by *Spring* (e.g. XML configuration or with a Java configuration class using *Spring Boot*). This allows the *Spring* container to create an instance of this class and wrap it in a PojoFunctionWrapper. *Spring* creates a wrapper instance for each method annotated with `@GemfireFunction`. Each wrapper instance shares the same target object instance to invoke the corresponding method.

TIP The fact that the POJO Function class is a *Spring* bean may offer other benefits since it shares the `ApplicationContext` with GemFire components such as the Cache and Regions. These may be injected into the class if necessary.

Spring creates the wrapper class and registers the Function(s) with GemFire's Function Service. The Function id used to register the Functions must be unique. Using convention it defaults to the simple (unqualified) method name. The name can be explicitly defined using the `id` attribute of the `@GemfireFunction` annotation. The `@GemfireFunction` annotation also provides other configuration attributes, `HA` and `optimizedForWrite`, which correspond to properties defined by GemFire's Function interface. If the method's return type is void, then the `hasResult` property is automatically set to `false`; otherwise, if the method returns a value the `hasResult` attributes is set to `true`.

Even for `void` return types, the annotation's `hasResult` attribute can be set to `true` to override this convention, as shown in the `functionWithContext` method above. Presumably, the intention is to use the `ResultSender` directly to send results to the caller.

The `PojoFunctionWrapper` implements GemFire's `Function` interface, binds method parameters and invokes the target method in its `execute()` method. It also sends the method's return value using the `ResultSender`.

10.3.2. Batching Results

If the return type is an array or Collection, then some consideration must be given to how the results are returned. By default, the `PojoFunctionWrapper` returns the entire array or Collection at once. If the number of elements in the array or Collection quite is large, it may incur a performance penalty. To divide the payload into smaller, more maneable chunks, you can set the `batchSize` attribute, as illustrated in `function2`, above.

TIP If you need more control of the `ResultSender`, especially if the method itself would use too much memory to create the Collection, you can pass the `ResultSender`, or access it via the `FunctionContext` and use it directly within the method to sends results back to the caller.

10.3.3. Enabling Annotation Processing

In accordance with *Spring* standards, you must explicitly activate annotation processing for `@GemfireFunction` annotations.

Using XML:

```
<gfe:annotation-driven/>
```

Or by annotating a Java configuration class:

```
@Configuration
@EnableGemfireFunctions
class ApplicationConfiguration { .. }
```

10.4. Executing a Function

A process invoking a remote Function needs to provide the Function's ID, calling arguments, the execution target (onRegion, onServers, onServer, onMember, onMembers) and optionally, a Filter set. Using *Spring Data GemFire*, all a developer need do is define an interface supported by annotations. *Spring* will create a dynamic proxy for the interface, which will use the `FunctionService` to create an `Execution`, invoke the `Execution` and coerce the results to the defined return type, if necessary. This technique is very similar to the way *Spring Data GemFire's Repository extension* works, thus some of the configuration and concepts should be familiar. Generally, a single interface definition maps to multiple Function executions, one corresponding to each method defined in the interface.

10.4.1. Annotations for Function Execution

To support client-side Function execution, the following SDG Function annotations are provided: `@OnRegion`, `@OnServer`, `@OnServers`, `@OnMember`, `@OnMembers`. These annotations correspond to the `Execution` implementations prodided by GemFire's FunctionService. Each annotation exposes the appropriate attributes. These annotations also provide an optional `resultCollector` attribute whose

value is the name of a *Spring* bean implementing the ResultCollector to use for the execution.

CAUTION	The proxy interface binds all declared methods to the same execution configuration. Although, it is expected that single method interfaces will be common, all methods in the interface are backed by the same proxy instance and therefore all share the same configuration.

Here are a few examples:

```
@OnRegion(region="SomeRegion", resultCollector="myCollector")
public interface FunctionExecution {

    @FunctionId("function1")
    String doIt(String s1, int i2);

    String getString(Object arg1, @Filter Set<Object> keys);

}
```

By default, the Function ID is the simple (unqualified) method name. The `@FunctionId` annotation can be used to bind this invocation to a different Function ID.

10.4.2. Enabling Annotation Processing

The client-side uses *Spring's* classpath component scanning capability to discover annotated interfaces. To enable Function execution annotation processing in XML:

```
<gfe-data:function-executions base-package="org.example.myapp.gemfire.functions"/>
```

The `function-executions` element is provided in the `gfe-data` namespace. The `base-package` attribute is required to avoid scanning the entire classpath. Additional filters are provided as described in the *Spring* reference documentation.

Optionally, a developer can annotate her Java configuration class:

```
@EnableGemfireFunctionExecutions(basePackages = "org.example.myapp.gemfire.functions")
```

10.5. Programmatic Function Execution

Using the Function execution annotated interface defined in the previous section, simply auto-wire your interface into an application bean that will invoke the Function:

```
@Component
public class MyApplication {

    @Autowired
    FunctionExecution functionExecution;

    public void doSomething() {
         functionExecution.doIt("hello", 123);
    }
}
```

Alternately, you can use a Function execution template directly. For example, `GemfireOnRegionFunctionTemplate` creates an `onRegion` Function `Execution`.

Example 7. Using the `GemfireOnRegionFunctionTemplate`

```
Set<?, ?> myFilter = getFilter();
Region<?, ?> myRegion = getRegion();
GemfireOnRegionOperations template = new GemfireOnRegionFunctionTemplate(myRegion
);
String result = template.executeAndExtract("someFunction", myFilter, "hello",
"world", 1234);
```

Internally, Function `Executions` always return a `List`. `executeAndExtract` assumes a singleton `List` containing the result and will attempt to coerce that value into the requested type. There is also an `execute` method that returns the `List` as is. The first parameter is the Function ID. The Filter argument is optional. The following arguments are a variable argument `List`.

10.6. Function Execution with PDX

When using *Spring Data GemFire's* Function annotation support combined with Pivotal GemFire's PDX Serialization, there are a few logistical things to keep in mind.

As explained above, and by way of example, typically developers will define GemFire Functions using POJO classes annotated with Spring Data GemFire Function annotations like so...

```
public class OrderFunctions {

  @GemfireFunction(...)
  Order process(@RegionData data, Order order, OrderSource orderSourceEnum, Integer
count) { ... }

}
```

NOTE	The Integer type, count parameter is arbitrary as is the separation of the `Order` class and `OrderSource` Enum, which might be logical to combine. However, the arguments were setup this way to demonstrate the problem with Function executions in the context of PDX.

Your `Order` and `OrderSource` enum might be as follows...

```
public class Order ... {

  private Long orderNumber;
  private Calendar orderDateTime;
  private Customer customer;
  private List<Item> items

  ...
}


public enum OrderSource {
  ONLINE,
  PHONE,
  POINT_OF_SALE
  ...
}
```

Of course, a developer may define a Function `Execution` interface to call the 'process' GemFire Server Function...

```
@OnServer
public interface OrderProcessingFunctions {
  Order process(Order order, OrderSource orderSourceEnum, Integer count);
}
```

Clearly, this `process(..) Order` Function is being called from a client-side with a `ClientCache` (i.e. `<gfe:client-cache/>`) based application. This implies that the Function arguments must also be serializable. The same is true when invoking peer-to-peer member Functions (e.g. `@OnMember(s)) between peers in the cluster. Any form of `distribution` requires the data transmitted between client and server, or peers, to be serialized.

Now, if the developer has configured GemFire to use PDX for serialization (instead of Java serialization, for instance) it is common for developers to also set the `pdx-read-serialized` attribute to **true** in their configuration of the GemFire server(s)...

```
<gfe:cache ... pdx-read-serialized="true"/>
```

Or from a GemFire cache client application...

```
<gfe:client-cache ... pdx-read-serialized="true"/>
```

This causes all values read from the cache (i.e. Regions) as well as information passed between client and servers, or peers, to remain in serialized form, including, but not limited to, Function arguments.

GemFire will only serialize application domain object types that you have specifically configured (registered), with either GemFire's ReflectionBasedAutoSerializer, or specifically (and recommended) using a "custom" GemFire PdxSerializer. If you are using *Spring Data GemFire's* Repository extension to *Spring Data Common's* Repository abstraction and infrastructure, you might even want to consider using *Spring Data GemFire's* MappingPdxSerializer, which uses a entity's mapping meta-data to determine data from the application domain object that will be serialized to the PDX instance.

What is less than apparent, though, is that GemFire automatically handles Java Enum types regardless of whether they are explicitly configured or not (i.e. registered with a `ReflectionBasedAutoSerializer` using a regex pattern and the `classes` parameter, or are handled by a "custom" GemFire `PdxSerializer`), despite the fact that Java Enums implement `java.io.Serializable`.

So, when a developer sets `pdx-read-serialized` to **true** on GemFire Servers where the GemFire Functions (including Spring Data GemFire Function annotated POJO classes) are registered, then the developer may encounter surprising behavior when invoking the Function `Execution`.

What the developer may pass as arguments when invoking the Function is...

```
orderProcessingFunctions.process(new Order(123, customer, Calendar.getInstance(),
items), OrderSource.ONLINE, 400);
```

But, what the GemFire Function on the Server gets is...

```
process(regionData, order:PdxInstance, :PdxInstanceEnum, 400);
```

The `Order` and `OrderSource` have been passed to the Function as PDX instances. Again, this is all because `pdx-read-serialized` is set to **true**, which may be necessary in cases where the GemFire Servers are interacting with multiple different clients (e.g. Java, native clients, such as C++/C#, etc).

This flies in the face of *Spring Data GemFire's* "strongly-typed", Function annotated POJO class method signatures, as the developer is expecting application domain object types, not PDX serialized instances.

So, *Spring Data GemFire* includes enhanced Function support to automatically convert method arguments passed to the Function that are of type PDX to the desired application domain object types defined by the Function method's parameter types.

However, this also requires the developer to explicitly register a GemFire `PdxSerializer` on the GemFire Servers where *Spring Data GemFire* Function annotated POJOs are registered and used,

e.g. ...

```
<bean id="customPdxSerializer" class=
"x.y.z.gemfire.serialization.pdx.MyCustomPdxSerializer"/>

<gfe:cache ... pdx-serializer-ref="customPdxSerializeer" pdx-read-serialized="true"/>
```

Alternatively, a developer my use GemFire's ReflectionBasedAutoSerializer for convenience. Of course, it is recommended that you use a "custom" `PdxSerializer` where possible to maintain finer grained control over your serialization strategy.

Finally, *Spring Data GemFire* is careful not to convert your Function arguments if you treat your Function arguments generically, or as one of GemFire's PDX types...

```
@GemfireFunction
public Object genericFunction(String value, Object domainObject, PdxInstanceEnum enum)
{
 ...
}
```

Spring Data GemFire only converts PDX type data to the corresponding application domain types if and only if the corresponding application domain types are on the classpath the the Function annotated POJO method expects it.

For a good example of "custom", "composed" application-specific GemFire `PdxSerializers` as well as appropriate POJO Function parameter type handling based on the method signatures, see Spring Data GemFire's ClientCacheFunctionExecutionWithPdxIntegrationTest class.

Chapter 11. Apache Lucene Integration

Pivotal GemFire integrates with Apache Lucene to allow developers to index and search on data stored in Pivotal GemFire using Lucene queries. Search-based queries also includes the capability to page through query results.

Additionally, *Spring Data GemFire* adds support for query projections based on *Spring Data Commons* Projection infrastructure. This feature enables the query results to be projected into first-class, application domain types as needed or required by the application use case.

However, a Lucene `Index` must be created first before any Lucene search-based query can be ran. A `LuceneIndex` can be created in *Spring (Data GemFire)* XML config like so...

```
<gfe:lucene-index id="IndexOne" fields="fieldOne, fieldTwo" region-path="/Example"/>
```

Additionally, Apache Lucene allows the specification of Analyzers per field and can be configured using...

```
<gfe:lucene-index id="IndexTwo" lucene-service-ref="luceneService" region-path=
"/AnotherExample">
    <gfe:field-analyzers>
        <map>
            <entry key="fieldOne">
                <bean class="example.AnalyzerOne"/>
             </entry>
            <entry key="fieldTwo">
                <bean class="example.AnalyzerTwo"/>
             </entry>
        </map>
    </gfe:field-analyzers>
</gfe:lucene-index>
```

Of course, the `Map` can be specified as a top-level bean definition and referenced using the `ref` attribute on the nested `<gfe:field-analyzers>` element like this, `<gfe-field-analyzers ref="refToTopLevelMapBeanDefinition"/>`.

Alternatively, a `LuceneIndex` can be declared in *Spring* Java config, inside a `@Configuration` class with...

```
@Bean(name = "People")
@DependsOn("personTitleIndex")
PartitionedRegionFactoryBean<Long, Person> peopleRegion(GemFireCache gemfireCache) {
    PartitionedRegionFactoryBean<Long, Person> peopleRegion = new
PartitionedRegionFactoryBean<>();

    peopleRegion.setCache(gemfireCache);
    peopleRegion.setClose(false);
    peopleRegion.setPersistent(false);

    return peopleRegion;
}

@Bean
LuceneIndexFactoryBean personTitleIndex(GemFireCache gemFireCache) {
    LuceneIndexFactoryBean luceneIndex = new LuceneIndexFactoryBean();

    luceneIndex.setCache(gemFireCache);
    luceneIndex.setFields("title");
    luceneIndex.setRegionPath("/People");

    return luceneIndex;
}
```

There are a few limitations of Pivotal GemFire's, Apache Lucene integration support. First, a `LuceneIndex` can only be created on a GemFire `PARTITION` Region. Second, all `LuceneIndexes` must be created before the the Region on which the `LuceneIndex` is applied.

It is possible that these Pivotal GemFire restrictions will not apply in a future release which is why the SDG `LuceneIndexFactoryBean` API takes a reference to the Region directly as well, rather than just the Region path.

This is more ideal if think about the case in which users may want to define a `LuceneIndex` on an existing Region with data at a later point during the application's lifecycle and as requirements demand. Where possible, SDG strives to stick to strongly-typed objects.

Now that we have a `LuceneIndex` we can perform Lucene based data access operations, such as queries.

11.1. Lucene Template Data Accessors

Spring Data GemFire provides 2 primary templates for Lucene data access operations, depending on how low a level your application is prepared to deal with.

The `LuceneOperations` interface defines query operations using Pivotal GemFire Lucene types.

```
public interface LuceneOperations {

    <K, V> List<LuceneResultStruct<K, V>> query(String query, String defaultField [,
int resultLimit]
        , String... projectionFields);

    <K, V> PageableLuceneQueryResults<K, V> query(String query, String defaultField,
        int resultLimit, int pageSize, String... projectionFields);

    <K, V> List<LuceneResultStruct<K, V>> query(LuceneQueryProvider queryProvider [,
int resultLimit]
        , String... projectionFields);

    <K, V> PageableLuceneQueryResults<K, V> query(LuceneQueryProvider queryProvider,
        int resultLimit, int pageSize, String... projectionFields);

    <K> Collection<K> queryForKeys(String query, String defaultField [, int
resultLimit]);

    <K> Collection<K> queryForKeys(LuceneQueryProvider queryProvider [, int
resultLimit]);

    <V> Collection<V> queryForValues(String query, String defaultField [, int
resultLimit]);

    <V> Collection<V> queryForValues(LuceneQueryProvider queryProvider [, int
resultLimit]);
}
```

NOTE The `[, int resultLimit]` indicates that the `resultLimit` parameter is optional.

The operations in the `LuceneOperations` interface match the operations provided by the Pivotal GemFire's LuceneQuery interface. However, SDG has the added value of translating proprietary GemFire or Lucene `Exceptions` into *Spring's* highly consistent and expressive DAO Exception Hierarchy, particularly as many modern data access operations involve more than single store or repository.

Additionally, SDG's `LuceneOperations` interface can shield your application from interface breaking changes introduced by the underlying Pivotal GemFire or Apache Lucene APIs when they do and will occur.

However, it would be remorse to only offer a Lucene Data Access Object that only uses Pivotal GemFire and Apache Lucene data types (e.g. GemFire's `LuceneResultStruct`), therefore SDG gives you the `ProjectingLuceneOperations` interface to remedy these important application concerns.

```
public interface ProjectingLuceneOperations {

    <T> List<T> query(String query, String defaultField [, int resultLimit], Class<T>
projectionType);

    <T> Page<T> query(String query, String defaultField, int resultLimit, int
pageSize, Class<T> projectionType);

    <T> List<T> query(LuceneQueryProvider queryProvider [, int resultLimit], Class<T>
projectionType);

    <T> Page<T> query(LuceneQueryProvider queryProvider, int resultLimit, int
pageSize, Class<T> projectionType);
}
```

The `ProjectingLuceneOperations` interface primarily uses application domain object types to work with your application data. The `query` method variants accept a projection type and the template applies the query results to instances of the given projection type using the *Spring Data Commons* Projection infrastructure.

Additionally, the template wraps the paged Lucene query results in an instance of the *Spring Data Commons* abstraction representing a `Page`. The same projection logic can still be applied to the results in the page and are lazily projected as each page in the collection is accessed.

By way of example, suppose I have a class representing a `Person` like so…

```
class Person {

    Gender gender;

    LocalDate birthDate;

    String firstName;
    String lastName;

    ...

    String getName() {
        return String.format("%1$s %2$s", getFirstName(), getLastName());
    }
}
```

Additionally, I might have a single interface to represent people as `Customers` depending on my application view…

```
interface Customer {

    String getName()
}
```

If I define the following `LuceneIndex`...

```
@Bean
LuceneIndexFactoryBean personLastNameIndex(GemFireCache gemfireCache) {
    LuceneIndexFactoryBean personLastNameIndex = new LuceneIndexFactoryBean();

    personLastNameIndex.setCache(gemfireCache);
    personLastNameIndex.setFields("lastName");
    personLastNameIndex.setRegionPath("/People");

    return personLastNameIndex;
}
```

Then it is a simple matter to query for people as either `Person` objects...

```
List<Person> people = luceneTemplate.query("lastName: D*", "lastName", Person.class);
```

Or as a `Page` of type `Customer`...

```
Page<Customer> customers = luceneTemplate.query("lastName: D*", "lastName", 100, 20,
Customer.class);
```

The `Page` can then be used to fetch individual pages of results...

```
List<Customer> firstPage = customers.getContent();
```

Conveniently, the *Spring Data Commons* `Page` interface implements `java.lang.Iterable<T>` too making it very easy to iterate over the content as well.

The only restriction to the *Spring Data Commons* Projection infrastructure is that the projection type must be an interface. However, it is possible to extend the provided, out-of-the-box (OOTB) SDC Projection infrastructure and provide a custom ProjectionFactory that uses CGLIB to generate proxy classes as the projected entity.

A custom `ProjectionFactory` can be set on a Lucene template using `setProjectionFactory(:ProjectionFactory)`.

11.2. Annotation configuration support

Finally, *Spring Data GemFire* provides Annotation configuration support for `LuceneIndexes`. Eventually, the SDG Lucene support will find its way into the *Repository* infrastructure extension for Pivotal GemFire so that Lucene queries can be expressed as methods on an application `Repository` interface, much like the OQL support today.

However, in the meantime, if you want to conveniently express `LuceneIndexes`, you can do so directly on your application domain objects like so...

```
@PartitionRegion("People")
class Person {

    Gender gender;

    @Index
    LocalDate birthDate;

    String firstName;

    @LuceneIndex;
    String lastName;

    ...
}
```

You must be using the SDG Annotation configuration support along with the `@EnableEntityDefineRegions` and `@EnableIndexing` Annotations to enable this feature...

```
@PeerCacheApplication
@EnableEntityDefinedRegions
@EnableIndexing
class ApplicationConfiguration {

  ...
}
```

Given our definition of the `Person` class above, the SDG Annotation configuration support will find the `Person` entity class definition, determine that people will be stored in a `PARTITION` Region called "People" and that the Person will have an OQL `Index` on `birthDate` along with a `LuceneIndex` on `lastName`.

More will be described with this feature in subsequent releases.

Chapter 12. Bootstrapping a Spring ApplicationContext in Pivotal GemFire

12.1. Introduction

Normally, a *Spring*-based application will bootstrap Pivotal GemFire using *Spring Data GemFire's. Just by specifying a* `<gfe:cache/>` *element using the _Spring Data GemFire* XML namespace, a single, embedded GemFire peer `Cache` instance is created and initialized with default settings in the same JVM process as your application.

However, it is sometimes necessary, perhaps a requirement imposed by your IT organization, that GemFire be fully managed and operated using the provided Pivotal GemFire tool suite, such as with Gfsh. By using *Gfsh*, GemFire will bootstrap your *Spring* application context rather than the other way around. Instead of an application server, or a Java main class using *Spring Boot*, whatever, GemFire does the bootstrapping and will host your application.

Keep in mind, however, that GemFire is not an application server. In addition, there are limitations to using this approach where GemFire cache configuration is concerned.

12.2. Using Pivotal GemFire to Bootstrap a Spring Context Started with Gfsh

In order to bootstrap a *Spring* application context in GemFire when starting a GemFire Server process using *Gfsh*, a user must make use of GemFire's Initalizer functionality. An Initializer block can declare a callback application that is launched after the cache is initialized by GemFire.

An Initializer is declared within an initializer element using a minimal snippet of GemFire's native `cache.xml`. The `cache.xml` file is required in order to bootstrap the *Spring* application context, much like a minimal snippet of *Spring* XML config is needed to bootstrap a *Spring* application context configured with component scanning (e.g. `<context:component-scan base-packages="⋯"/>`)

Fortunately, such an Initializer is already conveniently provided by the framework, the SpringContextBootstrappingInitializer. A typical, yet very minimal configuration for this class inside GemFires's `cache.xml` file will look like this:

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

  <initializer>
    <class-
name>org.springframework.data.gemfire.support.SpringContextBootstrappingInitializer</c
lass-name>
    <parameter name="contextConfigLocations">
      <string>classpath:application-context.xml</string>
    </parameter>
  </initializer>

</cache>
```

The `SpringContextBootstrappingInitializer` class follows similar conventions as *Spring's* `ContextLoaderListener` class used to bootstrap a *Spring* application context inside a Web Application, where application context configuration files are specified with the `contextConfigLocations` Servlet Context Parameter.

In addition, the `SpringContextBootstrappingInitializer` class can also be used with a `basePackages` parameter to specify a comma-separated list of base packages containing appropriately annotated application components that the *Spring* container will search in order to find and create *Spring* beans and other application components on the classpath:

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

  <initializer>
    <class-
name>org.springframework.data.gemfire.support.SpringContextBootstrappingInitializer</c
lass-name>
    <parameter name="basePackages">
      <string>org.mycompany.myapp.services,org.mycompany.myapp.dao,...</string>
    </parameter>
  </initializer>

</cache>
```

Then, with a properly configured and constructed `CLASSPATH` along with `cache.xml` file shown above, specified as a command-line option when starting a GemFire Server in *Gfsh*, the command-line

would be:

```
gfsh>start server --name=Server1 --log-level=config ...
    --classpath="/path/to/application/classes.jar:/path/to/spring-data-geode
-<major>.<minor>.<maint>.RELEASE.jar"
    --cache-xml-file="/path/to/geode/cache.xml"
```

The `application-context.xml` can be any valid *Spring* context configuration meta-data including all the SDG namespace elements. The only limitation with this approach is that a GemFire cache cannot be configured using the *Spring Data GemFire* namespace. In other words, none of the `<gfe:cache/>` element attributes, such as `cache-xml-location`, `properties-ref`, `critical-heap-percentage`, `pdx-serializer-ref`, `lock-lease`, etc, can be specified. If used, these attributes will be ignored.

The reason for this is that GemFire itself has already created an initialized the cache before the Initializer gets invoked. As such, the cache will already exist and since it is a "Singleton", it cannot be re-initialized or have any of it's configuration augmented.

12.3. Lazy-Wiring GemFire Components

Spring Data GemFire already provides existing support for wiring GemFire components, such as `CacheListeners`, `CacheLoaders`, `CacheWriters` and so on, that are declared and created by GemFire in `cache.xml` using SDG's `WiringDeclarableSupport` class as described in Configuration using auto-wiring and annotations. However, this only works when *Spring* is the one doing the bootstrapping (i.e. bootstrapping GemFire).

When your *Spring* application context is bootstrapped by GemFire, then these GemFire application components go unnoticed since the *Spring* application context does not even exist yet! The *Spring* application context will not get created until GemFire calls the Initializer block, which only occurs after all the other GemFire components and configuration have already been created and initialized.

So, in order to solve this problem, a new `LazyWiringDeclarableSupport` class was introduced that is, in a sense, *Spring* application context aware. The intention of this abstract base class is that any implementing class will register itself to be configured by the *Spring* container that will eventually be created by GemFire once the Initializer is called. In essence, this give your GemFire defined application components a chance to be configured and auto-wired with *Spring* beans defined in the *Spring* application context.

In order for your GemFire application components to be auto-wired by the *Spring* container, create an application class that extends the `LazyWiringDeclarableSupport` and annotate any class member that needs to be provided as a *Spring* bean dependency, similar to:

```
public class UserDataSourceCacheLoader extends LazyWiringDeclarableSupport
    implements CacheLoader<String, User> {

  @Autowired
  private DataSource userDataSource;

  ...
}
```

As implied in the `CacheLoader` example above, you might necessarily (although, rarely) have defined both a Region and `CacheListener` component in GemFire `cache.xml`. The `CacheLoader` may need access to an application DAO, or perhaps a *Spring* application context defined JDBC `DataSource` for loading `Users` into a GemFire `REPLICATE` Region on start.

CAUTION	Be careful when mixing the different life-cycles of Pivotal GemFire and the *Spring* Container together in this manner as not all use cases and scenarios are supported. The GemFire `cache.xml` configuration would be similar to the following (which comes from SDG's test suite):

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

  <region name="Users" refid="REPLICATE">
    <region-attributes initial-capacity="101" load-factor="0.85">
      <key-constraint>java.lang.String</key-constraint>
      <value-constraint>
org.springframework.data.gemfire.repository.sample.User</value-constraint>
      <cache-loader>
        <class-name>

org.springframework.data.gemfire.support.SpringContextBootstrappingInitializerIntegrat
ionTest$UserDataStoreCacheLoader
        </class-name>
      </cache-loader>
    </region-attributes>
  </region>

  <initializer>
    <class-
name>org.springframework.data.gemfire.support.SpringContextBootstrappingInitializer</c
lass-name>
    <parameter name="basePackages">
      <string>org.springframework.data.gemfire.support.sample</string>
    </parameter>
  </initializer>

</cache>
```

Chapter 13. Sample Applications

NOTE Sample applications are now maintained in the Spring GemFire Examples repository.

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

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

13.1. Hello World

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

NOTE

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 (e.g. `-Djava.net.preferIPv4Stack=true`). For an alternative (global) fix especially on Ubuntu see SGF-28.

13.1.1. Starting and stopping the sample

Hello World is designed as a stand-alone Java application. It features a `main` class which can be started either from your IDE of choice (in Eclipse/STS through `Run As/Java Application`) or from the command-line through Maven using `mvn exec:java`. A developer 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 JVM and shutdown the *Spring* container.

13.1.2. Using the sample

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

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

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

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

```
-> Bold Section qName:emphasis level:5, chunks:[put 1 unu] attrs:[role:bold]
INFO: Added [1=unu] to the cache
null
-> Bold Section qName:emphasis level:5, chunks:[put 1 one] attrs:[role:bold]
INFO: Updated [1] from [unu] to [one]
unu
-> Bold Section qName:emphasis level:5, chunks:[size] attrs:[role:bold]
1
-> Bold Section qName:emphasis level:5, chunks:[put 2 two] attrs:[role:bold]
INFO: Added [2=two] to the cache
null
-> Bold Section qName:emphasis level:5, chunks:[size] attrs:[role:bold]
2
```

Multiple instances can be ran 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!
...

-> Bold Section qName:emphasis level:5, chunks:[size] attrs:[role:bold]
2
-> Bold Section qName:emphasis level:5, chunks:[map] attrs:[role:bold]
[2=two] [1=one]
-> Bold Section qName:emphasis level:5, chunks:[query length = 3] attrs:[role:bold]
[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.

13.1.3. Hello World Sample Explained

Hello World uses both *Spring* XML and annotations for its configuration. The initial bootstrapping configuration is `app-context.xml`, which includes the cache configuration defined in the `cache-context.xml` file and performs classpath component scanning for *Spring* components.

The cache configuration defines the GemFire cache, Region and for illustrative purposes, a simple `CacheListener` 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.

Resources

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

Chapter 14. Useful Links

- Spring Data GemFire Project Page
- Spring Data GemFire source code
- Spring Data GemFire JIRA
- Spring Data GemFire on StackOverflow
- Archive of the Spring Data GemFire Forum on Spring IO
- Pivotal GemFire Home Page
- Pivotal GemFire Documentation
- Apache Geode Community
- Apache Geode source code
- Apache Geode JIRA
- Pivotal GemFire on StackOverflow

Appendices

Appendix A: Namespace reference

The <repositories /> element

The `<repositories />` element triggers the setup of the Spring Data repository infrastructure. The most important attribute is `base-package` which defines the package to scan for Spring Data repository interfaces. [1: see [repositories.create-instances.spring]]

Table 6. Attributes

Name	Description
`base-package`	Defines the package to be used to be scanned for repository interfaces extending *Repository (actual interface is determined by specific Spring Data module) in auto detection mode. All packages below the configured package will be scanned, too. Wildcards are allowed.
`repository-impl-postfix`	Defines the postfix to autodetect custom repository implementations. Classes whose names end with the configured postfix will be considered as candidates. Defaults to `Impl`.
`query-lookup-strategy`	Determines the strategy to be used to create finder queries. See [repositories.query-methods.query-lookup-strategies] for details. Defaults to `create-if-not-found`.
`named-queries-location`	Defines the location to look for a Properties file containing externally defined queries.
`consider-nested-repositories`	Controls whether nested repository interface definitions should be considered. Defaults to `false`.

Appendix B: Populators namespace reference

The <populator /> element

The `<populator />` element allows to populate the a data store via the Spring Data repository infrastructure. [2: see [repositories.create-instances.spring]]

Table 7. Attributes

Name	Description
`locations`	Where to find the files to read the objects from the repository shall be populated with.

Appendix C: Repository query keywords

Supported query keywords

The following table lists the keywords generally supported by the Spring Data repository query derivation mechanism. However, consult the store-specific documentation for the exact list of supported keywords, because some listed here might not be supported in a particular store.

Table 8. Query keywords

Logical keyword	Keyword expressions
`AND`	`And`
`OR`	`Or`
`AFTER`	`After`, `IsAfter`
`BEFORE`	`Before`, `IsBefore`
`CONTAINING`	`Containing`, `IsContaining`, `Contains`
`BETWEEN`	`Between`, `IsBetween`
`ENDING_WITH`	`EndingWith`, `IsEndingWith`, `EndsWith`
`EXISTS`	`Exists`
`FALSE`	`False`, `IsFalse`
`GREATER_THAN`	`GreaterThan`, `IsGreaterThan`
`GREATER_THAN_EQUALS`	`GreaterThanEqual`, `IsGreaterThanEqual`
`IN`	`In`, `IsIn`
`IS`	`Is`, `Equals`, (or no keyword)
`IS_EMPTY`	`IsEmpty`, `Empty`
`IS_NOT_EMPTY`	`IsNotEmpty`, `NotEmpty`
`IS_NOT_NULL`	`NotNull`, `IsNotNull`
`IS_NULL`	`Null`, `IsNull`
`LESS_THAN`	`LessThan`, `IsLessThan`
`LESS_THAN_EQUAL`	`LessThanEqual`, `IsLessThanEqual`
`LIKE`	`Like`, `IsLike`
`NEAR`	`Near`, `IsNear`
`NOT`	`Not`, `IsNot`
`NOT_IN`	`NotIn`, `IsNotIn`
`NOT_LIKE`	`NotLike`, `IsNotLike`
`REGEX`	`Regex`, `MatchesRegex`, `Matches`
`STARTING_WITH`	`StartingWith`, `IsStartingWith`, `StartsWith`
`TRUE`	`True`, `IsTrue`
`WITHIN`	`Within`, `IsWithin`

Appendix D: Repository query return types

Supported query return types

The following table lists the return types generally supported by Spring Data repositories. However, consult the store-specific documentation for the exact list of supported return types, because some listed here might not be supported in a particular store.

NOTE Geospatial types like (`GeoResult`, `GeoResults`, `GeoPage`) are only available for data stores that support geospatial queries.

Table 9. Query return types

Return type	Description
`void`	Denotes no return value.
Primitives	Java primitives.
Wrapper types	Java wrapper types.
`T`	An unique entity. Expects the query method to return one result at most. In case no result is found `null` is returned. More than one result will trigger an `IncorrectResultSizeDataAccessException`.
`Iterator<T>`	An `Iterator`.
`Collection<T>`	A `Collection`.
`List<T>`	A `List`.
`Optional<T>`	A Java 8 or Guava `Optional`. Expects the query method to return one result at most. In case no result is found `Optional.empty()` /`Optional.absent()` is returned. More than one result will trigger an `IncorrectResultSizeDataAccessException`.
`Option<T>`	An either Scala or JavaSlang `Option` type. Semantically same behavior as Java 8's `Optional` described above.
`Stream<T>`	A Java 8 `Stream`.
`Future<T>`	A `Future`. Expects method to be annotated with `@Async` and requires Spring's asynchronous method execution capability enabled.
`CompletableFuture<T>`	A Java 8 `CompletableFuture`. Expects method to be annotated with `@Async` and requires Spring's asynchronous method execution capability enabled.
`ListenableFuture`	A `org.springframework.util.concurrent.ListenableFuture`. Expects method to be annotated with `@Async` and requires Spring's asynchronous method execution capability enabled.
`Slice`	A sized chunk of data with information whether there is more data available. Requires a `Pageable` method parameter.
`Page<T>`	A `Slice` with additional information, e.g. the total number of results. Requires a `Pageable` method parameter.
`GeoResult<T>`	A result entry with additional information, e.g. distance to a reference location.

Return type	Description
`GeoResults<T>`	A list of `GeoResult<T>` with additional information, e.g. average distance to a reference location.
`GeoPage<T>`	A `Page` with `GeoResult<T>`, e.g. average distance to a reference location.

Appendix E: 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:tool="http://www.springframework.org/schema/tool"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.springframework.org/schema/gemfire"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            version="2.0">
    <xsd:import namespace="http://www.springframework.org/schema/beans" />
    <xsd:import namespace="http://www.springframework.org/schema/context" />
    <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>
    <!-- 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>
    <!-- Abstract Cache Type -->
    <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 org.apache.geode.cache.TransactionListener
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 org.apache.geode.cache.TransactionWriter
and may be nested or referenced.
            ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="gateway-conflict-resolver" minOccurs="0" maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation
                        source="org.apache.geode.cache.util.GatewayConflictResolver"
><![CDATA[
A gateway conflict resolver for this cache. A gateway conflict resolver handles
conflicts in the case of concurrent updates using a WAN gateway. The bean
must implement org.apache.geode.cache.util.GatewayConflictResolver. Requires Gemfire
version 7.0 or higher.
                            ]]></xsd:documentation>
                    <xsd:appinfo>
                        <tool:annotation>
                            <tool:exports
                                type=
"org.apache.geode.cache.util.GatewayConflictResolver" />
                        </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 conflict resolver.
                                        ]]></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 conflict resolver bean referred by this declaration. Used as a
```

```
convenience method. If no reference exists,
use inner bean declarations.
                                    ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:complexType>
            </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="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 properties
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="false">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates whether a bean factory locator is enabled (default) for this cache
definition or not. The locator stores
the enclosing bean factory reference to allow auto-wiring of Spring beans into GemFire
managed classes. Usually disabled
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="close" default="true">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Determines if the cache should be closed when the application context is closed. This
value is
true by default but should be set to false if deploying multiple applications in a jvm
that share the
same cache instance.
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <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 cache (false)
or copies of the objects (true).
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="critical-heap-percentage">
            <xsd:annotation>
                <xsd:documentation
                    source="org.apache.geode.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 LowMemoryException to
be thrown during certain cache operation.  This feature requires additional VM flags
to perform properly (see the
JavaDocs for org.apache.geode.cache.control.ResourceManager for more information).
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="eviction-heap-percentage">
            <xsd:annotation>
                <xsd:documentation
                    source="org.apache.geode.cache.control.ResourceManager"><![CDATA[
Set the percentage of heap at or above which the eviction should begin on Regions
configured for HeapLRU eviction.
This feature requires additional VM flags to perform properly (see the
JavaDocs for org.apache.geode.cache.control.ResourceManager for more information).
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="pdx-serializer-ref" 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 serialize any
domain classes which are added to the cache in portable data exchange (PDX) format.
                ]]></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.
Set to true if you are using persistent regions, WAN gateways or GemFire's JSON
support.
                ]]></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 persistence.
If not set, the metadata will go in the default disk store.
                ]]></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 from the cache
a PdxInstance will be returned instead of the actual domain class. The PdxInstance is
an interface that provides run time
access to the fields of a PDX without deserializing the entire PDX. The PdxInstance
implementation is a light weight wrapper
that simply refers to the raw bytes of the PDX that are kept in the cache. Using this
method applications can choose to
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 deserialized PDX,
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 unread fields be
including their data during serialization. But if you configure the cache to ignore
unread fields then their data will be
lost during serialization.

You should only set this attribute to true if you know this member will only be
reading cache data. In this use case you
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:complexType>
    <!-- Peer Cache Type -->
    <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="org.apache.geode.cache.Cache" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="cacheBaseType">
                    <xsd:attribute name="enable-auto-reconnect" type="xsd:string" use
="optional" default="false">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
By default, GemFire 8.0 and later will attempt to reconnect and reinitialize the cache
when the peer member has been
forced out of the distributed system by a network-partition event, or has otherwise
been shunned by other members.

An auto-reconnect causes all GemFire component references (e.g. Cache, Regions, AEQs,
Gateways, etc) that may have been
injected into SDG application components to become stale.  Even when using GemFire's
public Java API directly,
GemFire makes no guarantees to automatically refresh any stale references used by
application objects.

Therefore, in Spring Data GemFire, the default behavior will be to not 'auto-
reconnect'.  Automatically reconnecting
is not recommended for applications that are also peer member Caches (i.e. GemFire
Servers) that inject
GemFire components, such as the Cache or Regions, into application objects (e.g.
@Repository POJOs).

Enabling 'auto-reconnect' is only recommended when bootstrapping a GemFire Server's
within a Spring context using
Spring Data GemFire XML namespace and configuration meta-data to configure GemFire
instead of GemFire's cache.xml.
                            ]]></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 and manual locking.
Once a lock is obtained, it can remain in force for the lock lease time period before
being automatically cleared by the system
```

```
                ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <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 period,
  it is cancelled and returns with a failure.
                ]]></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 redundancy is used.
Sets the frequency (in seconds) at which the primary server sends messages to its
secondary servers to remove queued events
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:attribute name="use-cluster-configuration" type="xsd:string"
use="optional" default="false">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Enables this Spring Data GemFire configured peer member Cache node to participate in
cluster-wide configuration
by receiving it's configuration meta-data (in the form of cache.xml) from a Locator,
with persistent configuration
enabled, running in the cluster.

The cluster-wide configuration is a shared, persistent and consistent view of the
cluster when the member joins
the cluster and requests the cluster configuration from the Locator.
```

```
Spring Data GemFire disables this GemFire 8 feature by default assuming that the
primary configuration for this
member will be defined in Spring Data GemFire's XML namespace configuration meta-data.
However, this feature is useful
in a production setting to set the base configuration of the member and augment that
cluster-wide configuration
with Spring configuration meta-data that is specific to the application's needs.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
<!-- Client Cache Type -->
<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="org.apache.geode.cache.client.ClientCache" />
            </tool:annotation>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="cacheBaseType">
                <xsd:attribute name="durable-client-id" type="xsd:string" use=
"optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Used only for clients in a client/server installation. If set, this indicates that the
client is durable
and identifies the client. The ID is used by servers to reestablish any messaging that
was interrupted
by client downtime. The default value is unset. In addition, this attribute value
overrides any setting
specified in gemfire.properties passed using 'properties-ref'.

                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="durable-client-timeout" type="xsd:string"
use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Used only for clients in a client/server installation. Number of seconds this client
```

```
can remain disconnected
from its server and have the server continue to accumulate durable events for it. The
default value is 300 seconds.
In addition, this attribute value overrides any setting specified in
gemfire.properties passed using 'properties-ref'.
Also, durable-client-timeout is only used if durable-client-id is set.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="keep-alive" type="xsd:string" use="optional"
default="false">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Boolean value indicating whether the server should keep the durable client's queues
alive for the timeout period.
                        ]]></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.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="ready-for-events" type="xsd:string" use=
"optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Notifies the server that this durable client is ready to receive updates.
                ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
<!-- Abstract Region Type  -->
<xsd:complexType name="basicRegionType">
    <xsd:annotation>
        <xsd:appinfo>
            <tool:annotation>
                <tool:exports type="org.apache.geode.cache.Region" />
            </tool:annotation>
        </xsd:appinfo>
    </xsd:annotation>
</xsd:complexType>
<!-- -->
<xsd:complexType name="basicSubRegionType">
    <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>
    <!-- Lookup Region Type -->
    <xsd:element name="auto-region-lookup">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
GFE namespace element enabling GemFire Cache Regions to be automatically looked up and
defined as beans in the Spring
context when those Regions are defined outside of Spring config, such as in GemFire's
native cache.xml or with
GemFire 8's new cluster-based configuration service.
            ]]></xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <!-- -->
    <xsd:complexType name="baseLookupRegionType">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
Defines a lookup Subregion
                    ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.Region" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="basicRegionType">
                <xsd:sequence>
                    <xsd:element name="cache-listener" minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation source=
"org.apache.geode.cache.CacheListener"><![CDATA[
A cache listener definition for this region. A cache listener handles region or entry
related events (that occur after
various operations on the region). Multiple listeners can be declared in a nested
manner.

Note: Avoid the risk of deadlock. Since the listener is invoked while holding a lock
on the entry generating the event,
it is easy to generate a deadlock by interacting with the region. For this reason, it
is highly recommended to use some
other thread for accessing the region and not waiting for it to complete its task.
                            ]]></xsd:documentation>
```

```
                            <xsd:appinfo>
                                <tool:annotation>
                                    <tool:exports type=
"org.apache.geode.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 reference exists,
use inner bean declarations.
                                    ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="cache-loader" type="beanDeclarationType"
minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation source=
"org.apache.geode.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=
"org.apache.geode.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=
"org.apache.geode.cache.CacheWriter"><![CDATA[
```

```
The cache writer definition for this region. A cache writer acts as a dedicated
synchronous listener that is notified
before a region or an entry is modified. A typical example would be a writer that
updates the database.

Note: Only one CacheWriter is invoked. GemFire will always prefer the local one (if it
exists) otherwise it will
arbitrarily pick one.
                            ]]></xsd:documentation>
                            <xsd:appinfo>
                                <tool:annotation>
                                    <tool:exports type=
"org.apache.geode.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:choice>
                        <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="custom-entry-ttl" type=
"customExpirationType" minOccurs="0" maxOccurs="1">
                            <xsd:annotation>
                                <xsd:appinfo>
                                    <tool:annotation>
                                        <tool:exports type=
"org.apache.geode.cache.CustomExpiry" />
                                    </tool:annotation>
```

```
                                </xsd:appinfo>
                                <xsd:documentation><![CDATA[
CustomExpiry Time-to-Live (TTL) configuration for the Region Entries. The default is
no expiration.
                            ]]></xsd:documentation>
                            </xsd:annotation>
                        </xsd:element>
                    </xsd:choice>
                    <xsd:choice>
                        <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:element name="custom-entry-tti" type=
"customExpirationType" minOccurs="0" maxOccurs="1">
                            <xsd:annotation>
                                <xsd:appinfo>
                                    <tool:annotation>
                                        <tool:exports type=
"org.apache.geode.cache.CustomExpiry" />
                                    </tool:annotation>
                                </xsd:appinfo>
                                <xsd:documentation><![CDATA[
CustomExpiry Time-to-Idle (or Idle Timeout, TTI) configuration for the Region entries.
The default is no expiration.
                            ]]></xsd:documentation>
                            </xsd:annotation>
                        </xsd:element>
                    </xsd:choice>
                    <xsd:choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="gateway-sender" type="
baseGatewaySenderType"/>
                        <xsd:element name="gateway-sender-ref">
                            <xsd:complexType>
                                <xsd:attribute name="bean" type="xsd:string" use=
"optional">
                                    <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The name of the gateway sender bean referred by this declaration. Used as a
convenience method. If no reference exists,
use inner bean declarations.
                                ]]></xsd:documentation>
                                    </xsd:annotation>
                                </xsd:attribute>
                            </xsd:complexType>
                        </xsd:element>
```

```
                    </xsd:choice>
                    <xsd:choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="async-event-queue" type=
"baseAsyncEventQueueType"/>
                        <xsd:element name="async-event-queue-ref">
                            <xsd:complexType>
                                <xsd:attribute name="bean" type="xsd:string" use=
"optional">
                                    <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The name of the gateway sender bean referred by this declaration. Used as a
convenience method. If no reference exists,
use inner bean declarations.
                                ]]></xsd:documentation>
                                    </xsd:annotation>
                                </xsd:attribute>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:choice>
                    <xsd:group ref="subRegionGroup" minOccurs="0" maxOccurs="
unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="cloning-enabled" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Determines how fromDelta applies deltas to the local cache for delta propagation. When
true, the updates are applied
to a clone of the value and then the clone is saved to the cache. When false, the
value is modified in place
in the cache. GemFire default is false.
                            ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="eviction-maximum" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
                        ]]></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:element name="lookup-region" type="lookupRegionType"/>
    <!-- -->
    <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=
"org.apache.geode.cache.CacheListener"><![CDATA[
A cache listener definition for this region. A cache listener handles region or entry
related events (that occur after
various operations on the region). Multiple listeners can be declared in a nested
manner.

Note: Avoid the risk of deadlock. Since the listener is invoked while holding a lock
on the entry generating the event,
it is easy to generate a deadlock by interacting with the region. For this reason, it
is highly recommended to use some
other thread for accessing the region and not waiting for it to complete its task.
                            ]]></xsd:documentation>
                            <xsd:appinfo>
                                <tool:annotation>
                                    <tool:exports type=
"org.apache.geode.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 reference exists,
use inner bean declarations.
                                    ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="compressor" type="beanDeclarationType"
minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation source=
"org.apache.geode.compression.Compressor"><![CDATA[
The Compressor definition for this Region. A Compressor registers a custom class that
implements Compressor
to support compression on a Region.
                            ]]></xsd:documentation>
                            <xsd:appinfo>
                                <tool:annotation>
                                    <tool:exports type=
"org.apache.geode.compression.Compressor"/>
                                </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:choice>
                        <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="custom-entry-ttl" type=
"customExpirationType" minOccurs="0" maxOccurs="1">
                            <xsd:annotation>
                                <xsd:appinfo>
                                    <tool:annotation>
                                        <tool:exports type=
"org.apache.geode.cache.CustomExpiry" />
                                    </tool:annotation>
                                </xsd:appinfo>
                                <xsd:documentation><![CDATA[
CustomExpiry Time-to-Live (TTL) configuration for the Region Entries. The default is
no expiration.
                            ]]></xsd:documentation>
                            </xsd:annotation>
                        </xsd:element>
                    </xsd:choice>
                    <xsd:choice>
                        <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:element name="custom-entry-tti" type=
"customExpirationType" minOccurs="0" maxOccurs="1">
                            <xsd:annotation>
                                <xsd:appinfo>
                                    <tool:annotation>
                                        <tool:exports type=
"org.apache.geode.cache.CustomExpiry" />
                                    </tool:annotation>
                                </xsd:appinfo>
                                <xsd:documentation><![CDATA[
CustomExpiry Time-to-Idle (or Idle Timeout, TTI) configuration for the Region entries.
The default is no expiration.
                            ]]></xsd:documentation>
                            </xsd:annotation>
                        </xsd:element>
                    </xsd:choice>
                </xsd:sequence>
                <xsd:attribute name="cloning-enabled" type="xsd:string">
                    <xsd:annotation>
```

```
                        <xsd:documentation><![CDATA[
Determines how fromDelta applies deltas to the local cache for delta propagation. When
true, the updates are applied
to a clone of the value and then the clone is saved to the cache. When false, the
value is modified in place
in the cache. GemFire default is false.
                            ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="close" type="xsd:string" default="false">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether the defined Region should be closed at shutdown. Close performs a
local destroy but leaves behind
the Region disk files. Additionally, close notifies listeners and callbacks.

Default is false

Note, Regions are automatically closed when the Cache closes.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="concurrency-checks-enabled" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Determines whether members perform checks to provide consistent handling for
concurrent or out-of-order updates to
distributed Regions. GemFire default is true.
                        ]]></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 and subregions.
After the destroy, this region object can not be used any more and any attempt to use
this region object will get
RegionDestroyedException.

Default is false, meaning that regions are not destroyed.
                        ]]></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="disk-synchronous" type="xsd:string">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
For Regions that write to disk, boolean that specifies whether disk writes are done
synchronously for the region.
GemFire default is true.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="ignore-if-exists" type="xsd:string" default=
"false">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether the Region bean definition should perform a "lookup" first, using
any existing Region already defined
with the same name in the Cache (thus, reverting to pre-1.4.0 behavior, e.g. 1.3.x),
before attempting to create
the Region

The default is false, meaning the default behavior is always attempt to "create" the
Region first.

Prior to 1.4.0, the default behavior was to perform a "lookup" first and then try to
"create" the Region.
This functionality is useful in situations where multiple Spring context configuration
files exists and 1 or more
define the same Region with the same semantics, such that the application dynamically
loads configuration files,
or creates new Spring contexts as needed and application components (e.g. application
DAOs) have runtime dependencies
on those Regions.

WARNING...

It is recommended that this feature be used carefully as the first bean definition to
create the Region wins.
So if there are multiple, conflicting bean definitions (with difference semantics for
evictions/expiration, etc)
for the "same" Region (by name), then this can cause confusion or have unexpected
consequences for the application.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="ignore-jta" type="xsd:string" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Indicates whether operations on this Region participates in active JTA transactions or
ignores them and operates
outside of transactions.  This is primarily used in cache loaders, writers, and
```

```
listeners that need to perform
non-transactional operations on a Region, such as caching a result set.  GemFire
default is false.
                            ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="initial-capacity" type="xsd:string" use="
optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Sets the initial capacity (number of entries) for the Region. GemFire default is 16.
                            ]]></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="load-factor" type="xsd:string" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[[
Together with the initial-capacity Region attribute, sets the initial parameters on
the underlying
java.util.ConcurrentHashMap used for storing Region entries. This must be a floating
point number
between 0 and 1, inclusive.  GemFire default value is 0.75.
                           ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <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 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.

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="statistics" type="xsd:string">
                    <xsd:annotation>
```

```
                        <xsd:documentation><![CDATA[
Boolean specifying whether to gather statistics on the Region. Statistics must be
enabled to use expiration on the
Region.  GemFire default is false.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="template" type="xsd:string" use="optional">
                    <xsd:annotation>
                        <xsd:documentation>
                            Specifies the parent, template Region from which to
inherit the Region attribute configuration.
                        </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: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" abstract="true">
        <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=
"org.apache.geode.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=
"org.apache.geode.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=
"org.apache.geode.cache.CacheWriter"><![CDATA[
The cache writer definition for this region. A cache writer acts as a dedicated
synchronous listener that is notified
before a region or an entry is modified. A typical example would be a writer that
updates the database.

Note: Only one CacheWriter is invoked. GemFire will always prefer the local one (if it
exists) otherwise it will
arbitrarily pick one.
                            ]]></xsd:documentation>
                            <xsd:appinfo>
                                <tool:annotation>
                                    <tool:exports type=
"org.apache.geode.cache.CacheWriter" />
                                </tool:annotation>
                            </xsd:appinfo>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:element name="membership-attributes" minOccurs="0" maxOccurs
="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Establishes reliability requirements and behavior for a region. Use this to configure
the region to require one or more membership roles to be running in the system for
reliable access to the region.
                            ]]></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"
default="no-access">
                                <xsd:annotation>
                                    <xsd:documentation><![CDATA[
Set one of the following values to specify how access to the Region is affected when
one or more required roles are lost:
(full-access, limited-access, no-access, or reconnect).  GemFire default is 'no-
access' when required-roles is
specified, and in SDG, the required-roles attribute is required.
                            ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                            <xsd:attribute name="resumption-action" type="xsd:string"
default="reinitialize">
                                <xsd:annotation>
                                    <xsd:documentation><![CDATA[
Specifies how the Region is affected by resumption of reliability when one or more
missing required roles return
to the distributed membership: (none or reinitialize).  GemFire default is
reinitialize when required-roles is
specified, and in SDG, the required-roles attribute is required.
                            ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="gateway-sender" type="
baseGatewaySenderType"/>
                        <xsd:element name="gateway-sender-ref">
                            <xsd:complexType>
                                <xsd:attribute name="bean" type="xsd:string" use=
"optional">
                                    <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The name of the gateway sender bean referred by this declaration. Used as a
convenience method. If no reference exists,
use inner bean declarations.
                                ]]></xsd:documentation>
                                    </xsd:annotation>
                                </xsd:attribute>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:choice>
```

```
                    <xsd:choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="async-event-queue" type=
"baseAsyncEventQueueType"/>
                        <xsd:element name="async-event-queue-ref">
                            <xsd:complexType>
                                <xsd:attribute name="bean" type="xsd:string" use=
"optional">
                                    <xsd:annotation>
                                        <xsd:documentation><![CDATA[
The name of the gateway sender bean referred by this declaration. Used as a
convenience method. If no reference exists,
use inner bean declarations.
                                ]]></xsd:documentation>
                                    </xsd:annotation>
                                </xsd:attribute>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:choice>
                </xsd:sequence>
                <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)
(Deprecated since Gemfire v 7.0)
                                ]]></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. (Deprecated since Gemfire v
7.0)
                                ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="index-update-type" type="xsd:string" use=
"optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Specifies whether Region indexes are maintained synchronously with Region
modifications, or asynchronously
in a background thread.  GemFire default is synchronous.
                        ]]></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:element name="region-template" type="regionType">
        <xsd:annotation>
            <xsd:documentation source=
"org.springframework.data.gemfire.RegionFactoryBean"><![CDATA[
Defines a template for creating multiple GemFire Regions that all share a common
attribute configuration.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.Region"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:element>
    <!-- -->
    <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:element name="client-region" type="clientSubRegionType"/>
        </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 bean name).
Required for subregions.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:attributeGroup>
    <!-- -->
    <xsd:attributeGroup name="distributedRegionAttributes">
        <xsd:attribute name="enable-async-conflation" type="xsd:string">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
For TCP/IP distributions between peers, specifies whether to allow aggregation of
asynchronous messages sent
by the producer member for the Region. This is a special-purpose boolean attribute
that applies only when
asynchronous queues are used for slow consumers. A false value disables conflation so
that all asynchronous messages
are sent individually.  GemFire default is false (even though GemFire User Guide
states it is true).
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="enable-subscription-conflation" type="xsd:string">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates whether the Region can conflate its messages to the client.  GemFire default
is false.
                ]]></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, multicast must be enabled for the
distributed system with the mcast-port gemfire.properties setting.
                                ]]></xsd:documentation>
```

```
            </xsd:annotation>
        </xsd:attribute>
    </xsd:attributeGroup>
    <!-- Client Region Type -->
    <xsd:complexType name="baseClientRegionType" abstract="true">
        <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 servers from
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="org.apache.geode.cache.Region" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="baseReadOnlyRegionType">
                <xsd:sequence>
                    <xsd:element name="cache-loader" type="beanDeclarationType"
minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation source=
"org.apache.geode.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=
"org.apache.geode.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=
"org.apache.geode.cache.CacheWriter"><![CDATA[
The cache writer definition for this region. A cache writer acts as a dedicated
synchronous listener that is notified
before a region or an entry is modified. A typical example would be a writer that
updates the database.

Note: Only one CacheWriter is invoked. GemFire will always prefer the local one (if it
exists) otherwise it will
```

```
arbitrarily pick one.
                            ]]></xsd:documentation>
                            <xsd:appinfo>
                                <tool:annotation>
                                    <tool:exports type=
"org.apache.geode.cache.CacheWriter" />
                                </tool:annotation>
                            </xsd:appinfo>
                        </xsd:annotation>
                    </xsd:element>
                    <xsd:choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="key-interest">
                            <xsd:annotation>
                                <xsd:documentation><![CDATA[
Key based interest. If the key is a List, then all the keys in the List will be
registered. The key can also be the
special token 'ALL_KEYS', which will register interest in all keys in the region. In
effect, this will cause an update
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 reference exists,
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 client.
                                    ]]></xsd:documentation>
                            </xsd:annotation>
                            <xsd:complexType>
                                <xsd:complexContent>
                                    <xsd:extension base="interestType">
                                        <xsd:attribute name="pattern" type="
xsd:string" use="required"/>
                                    </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="xsd:string"
default="LOCAL_DESTROY">
                                        <xsd:annotation>
                                            <xsd:documentation><![CDATA[
Set to one of the following Eviction Actions:

LOCAL_DESTROY - Entry is destroyed locally. Not available for Replicated Regions.

OVERFLOW_TO_DISK - Entry is overflowed to disk and the value set to null in memory.
For Partitioned Regions,
this provides the most reliable read behavior across the region.
                                            ]]></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="concurrency-level">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Provides an estimate of the maximum number of application threads that will
concurrently access a Region entry
at one time.  This attribute does not apply to Partitioned Regions. This attribute
helps GemFire optimize the use of
```

```
system resources and reduce thread contention. This sets an initial parameter on the
underlying
java.util.ConcurrentHashMap used for storing Region entries.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="data-policy" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The data policy for this client region. Can be either 'EMPTY' or 'NORMAL' (the
default). In case persistence or overflow
are 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 for zero
footprint producers that only want to distribute their data to others and for zero
footprint consumers that only want
to see events.

NORMAL - causes data that this region is interested in to be stored in local memory.
It allows the contents in this
cache to differ from other caches.
                    ]]></xsd:documentation>
                </xsd:annotation>
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="EMPTY"/>
                        <xsd:enumeration value="NORMAL"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="pool-name" 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) will be used.
                        ]]></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 pre-defined 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_HEAP_LRU"/>
                            <xsd:enumeration value="LOCAL_OVERFLOW"/>
                            <xsd:enumeration value="LOCAL_PERSISTENT"/>
                            <xsd:enumeration value="LOCAL_PERSISTENT_OVERFLOW"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="clientRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseClientRegionType">
                <xsd:attributeGroup ref="topLevelRegionAttributes" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="clientSubRegionType">
        <xsd:complexContent>
            <xsd:extension base="baseClientRegionType">
                <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="client-region-template" type="clientRegionType">
        <xsd:annotation>
            <xsd:documentation source=
"org.springframework.data.gemfire.ClientRegionFactoryBean"><![CDATA[
Defines a template for creating multiple GemFire Client Regions that all share a
common attribute configuration.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.Region"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:element>
    <!-- -->
    <xsd:element name="client-region" type="clientRegionType">
```

```
        <xsd:annotation>
            <xsd:documentation source=
"org.springframework.data.gemfire.ClientRegionFactoryBean"><![CDATA[
Defines a GemFire Client Region
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.Region"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:element>
    <!-- Local Region Type -->
    <xsd:complexType name="baseLocalRegionType" abstract="true">
        <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="org.apache.geode.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="xsd:string">
                                        <xsd:annotation>
                                            <xsd:documentation><![CDATA[
Set to one of the following Eviction Actions:

LOCAL_DESTROY - Entry is destroyed locally. Not available for Replicated Regions.

OVERFLOW_TO_DISK - Entry is overflowed to disk and the value set to null in memory.
For Partitioned Regions,
this provides the most reliable read behavior across the region.
                                            ]]></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="concurrency-level">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Provides an estimate of the maximum number of application threads that will
concurrently access a region entry at one time.
This attribute does not apply to partitioned regions. This attribute helps GemFire
optimize the use of system resources and
reduce thread contention. This sets an initial parameter on the underlying
java.util.ConcurrentHashMap used for storing region entries.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="data-policy" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Specifies the data policy for this region (NORMAL or PRELOADED).  Setting 'data-
policy' is not stictly necessary,
but if set, then the value must agree with the 'persistent' attribute if also
specified.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="NORMAL"/>
                            <xsd:enumeration value="PRELOADED"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="shortcut" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The RegionShortcut for this region. Allows easy initialization of the region based on
pre-defined defaults.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="LOCAL"/>
                            <xsd:enumeration value="LOCAL_HEAP_LRU"/>
                            <xsd:enumeration value="LOCAL_OVERFLOW"/>
                            <xsd:enumeration value="LOCAL_PERSISTENT"/>
                            <xsd:enumeration value="LOCAL_PERSISTENT_OVERFLOW"/>
                        </xsd:restriction>
```

```
                    </xsd:simpleType>
                </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-template" type="localRegionType">
        <xsd:annotation>
            <xsd:documentation source=
"org.springframework.data.gemfire.LocalRegionFactoryBean"><![CDATA[
Defines a template for creating multiple GemFire Local Regions that all share a common
attribute configuration.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.Region"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:element>
    <!-- -->
    <xsd:element name="local-region" type="localRegionType"/>
    <!-- Partition Region Type -->
    <xsd:complexType name="basePartitionedRegionType" abstract="true">
        <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="org.apache.geode.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=
"org.apache.geode.cache.PartitionResolver"><![CDATA[
The partition resolver definition for this region, allowing for custom partitioning.
GemFire uses the resolver to
colocate data based on custom criterias (such as colocating trades by month and year).
                            ]]></xsd:documentation>
                        <xsd:appinfo>
                            <tool:annotation>
                                <tool:exports type=
"org.apache.geode.cache.PartitionResolver" />
                            </tool:annotation>
                        </xsd:appinfo>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="partition-listener" minOccurs="0" maxOccurs="1
">
                    <xsd:annotation>
                        <xsd:documentation
                            source=
"org.apache.geode.cache.partition.PartitionListener"><![CDATA[
The PartitionListener definition for this Region. Defines a callback for Partitioned
Regions, invoked when
a Partition Region is created or any Bucket in a Partitioned Region becomes primary.
                            ]]></xsd:documentation>
                        <xsd:appinfo>
                            <tool:annotation>
                                <tool:exports type=
"org.apache.geode.cache.partition.PartitionListener" />
                            </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 ParitionListener.
                                        ]]></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 PartitionListener bean referred to by this declaration. Used for
convenience. If no reference exists,
use inner bean declarations.
                                    ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="fixed-partition" minOccurs="0" maxOccurs=
"unbounded">
                        <xsd:annotation>
                            <xsd:documentation
                                source=
"org.apache.geode.cache.partition.FixedPartitionAttributes"><![CDATA[
 Create a fixed partition with the given attributes. Required for a
FixedPartitionResolver.
                            ]]></xsd:documentation>
                        </xsd:annotation>
                        <xsd:complexType>
                            <xsd:attribute name="partition-name" type="xsd:string"
                                           use="required">
                                <xsd:annotation>
                                    <xsd:documentation
                                        source=
"org.apache.geode.cache.partition.FixedPartitionAttributes"><![CDATA[
Specifies the fixed partition name
                                ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                            <xsd:attribute name="primary" use="optional"
                                           default="true">
                                <xsd:annotation>
                                    <xsd:documentation
                                        source=
"org.apache.geode.cache.partition.FixedPartitionAttributes"><![CDATA[
Specifies if this member is primary for this partition
                                ]]></xsd:documentation>
                                </xsd:annotation>
```

```
                            </xsd:attribute>
                            <xsd:attribute name="num-buckets" use="optional">
                                <xsd:annotation>
                                    <xsd:documentation
                                        source=
"org.apache.geode.cache.partition.FixedPartitionAttributes"><![CDATA[
Specifies the number of buckets to allocate to the fixed partition
                            ]]></xsd:documentation>
                                </xsd:annotation>
                            </xsd:attribute>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="subscription" minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Subscription policy for the partitioned region.
                             ]]></xsd:documentation>
                        </xsd:annotation>
                        <xsd:complexType>
                            <xsd:attribute name="type" type="xsd:string" use="
optional"/>
                        </xsd:complexType>
                    </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="xsd:string"
default="LOCAL_DESTROY">
                                        <xsd:annotation>
                                            <xsd:documentation><![CDATA[
Set to one of the following Eviction Actions:

LOCAL_DESTROY - Entry is destroyed locally. Not available for Replicated Regions.

OVERFLOW_TO_DISK - Entry is overflowed to disk and the value set to null in memory.
For Partitioned Regions,
this provides the most reliable read behavior across the region.
                                            ]]></xsd:documentation>
                                        </xsd:annotation>
                                    </xsd:attribute>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
```

```
                <xsd:attributeGroup ref="distributedRegionAttributes" />
                <xsd:attribute name="copies" type="xsd:string" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The number of copies (0-3) of 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. GemFire default is 0, or
no redundancy.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </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="data-policy" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Specifies the data policy for this region (PARTITION, PERSISTENT_PARTITION).  Setting
'data-policy' is not
stictly necessary, but if set, then the value must agree with the 'persistent'
attribute if also specified.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="PARTITION"/>
                            <xsd:enumeration value="PERSISTENT_PARTITION"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </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 90%
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 Buckets may
move from one VM to another. Buckets may also have copies, depending on redundancy to
provide high availability in the
face of VM failure.

The number of Buckets should be prime, and as a rough guide, at the least four times
the number of partition VMs.
However, 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">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Applies when copies is greater than zero. The number of milliseconds to wait after a
member crashes
before reestablishing redundancy for the Region. A setting of -1 disables automatic
recovery of redundancy
after member failure. Gemfire default is -1.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:attribute>
                <xsd:attribute name="shortcut" use="optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
The RegionShortcut for this region. Allows easy initialization of the region based on
pre-defined defaults.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="PARTITION"/>
                            <xsd:enumeration value="PARTITION_HEAP_LRU"/>
                            <xsd:enumeration value="PARTITION_OVERFLOW"/>
                            <xsd:enumeration value="PARTITION_PERSISTENT"/>
                            <xsd:enumeration value="PARTITION_PERSISTENT_OVERFLOW"/>
                            <xsd:enumeration value="PARTITION_PROXY"/>
```

```
                            <xsd:enumeration value="PARTITION_PROXY_REDUNDANT"/>
                            <xsd:enumeration value="PARTITION_REDUNDANT"/>
                            <xsd:enumeration value="PARTITION_REDUNDANT_HEAP_LRU"/>
                            <xsd:enumeration value="PARTITION_REDUNDANT_OVERFLOW"/>
                            <xsd:enumeration value="PARTITION_REDUNDANT_PERSISTENT"/>
                            <xsd:enumeration value=
"PARTITION_REDUNDANT_PERSISTENT_OVERFLOW"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="startup-recovery-delay" type="xsd:string" use=
"optional">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Applies when copies is greater than zero. The number of milliseconds a newly started
member should wait
before trying to satisfy redundancy of Region data stored on other members. A setting
of -1 disables automatic recovery
of redundancy after new members join.  GemFire default is 0, meaning the default is to
recover redundancy immediately
when a new member joins the cluster.
                        ]]></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-template" type="partitionedRegionType">
        <xsd:annotation>
            <xsd:documentation source=
"org.springframework.data.gemfire.PartitionedRegionFactoryBean"><![CDATA[
Defines a template for creating multiple GemFire PARTITION Regions that all share a
common attribute configuration.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.Region"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:element>
    <!-- -->
    <xsd:element name="partitioned-region" type="partitionedRegionType"/>
    <!-- Replicate Region Type -->
    <xsd:complexType name="baseReplicatedRegionType" abstract="true">
        <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="org.apache.geode.cache.Region" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexContent>
            <xsd:extension base="baseRegionType">
                <xsd:sequence>
                    <xsd:element name="subscription" minOccurs="0" maxOccurs="1">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[
Subscription policy for the replicated region.
                            ]]></xsd:documentation>
                        </xsd:annotation>
                        <xsd:complexType>
                            <xsd:attribute name="type" type="xsd:string" use="
optional"/>
                        </xsd:complexType>
                    </xsd:element>
                    <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="xsd:string"
fixed="OVERFLOW_TO_DISK">
                                    <xsd:annotation>
                                        <xsd:documentation><![CDATA[
Set to the following Eviction Actions (Note LOCAL_DESTROY is not available for
Replicated Regions):

OVERFLOW_TO_DISK - Entry is overflowed to disk and the value set to null in memory.
For Partitioned Regions,
this provides the most reliable read behavior across the region.
                                        ]]></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 entry at one time.
This attribute does not apply to partitioned regions. This attribute helps GemFire
optimize the use of system resources and
reduce thread contention. This sets an initial parameter on the underlying
java.util.ConcurrentHashMap used for storing region entries.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="data-policy" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Specifies the data policy for this region (EMPTY, REPLICATE, PERSISTENT_REPLICATE).
Setting 'data-policy' is not
stictly necessary, but if set, then the value must agree with the 'persistent'
attribute if also specified.
                    ]]></xsd:documentation>
                </xsd:annotation>
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="EMPTY"/>
```

```
                        <xsd:enumeration value="REPLICATE"/>
                        <xsd:enumeration value="PERSISTENT_REPLICATE"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="is-lock-grantor" type="xsd:string" use="optional
">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Indicates whether the Region is a lock grantor. This attribute is only relevant for
Regions with global scope,
as only they allow locking.  GemFire default is false.
                            ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="scope" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Specifies the Scope for this Region: distributed-ack, distributed-no-ack, global

Scope determines how updates to Region Entries are distributed to the other Caches in
the Distributed System where
the Region and Entry are defined.  Scope also determines whether to allow remote
invocation of some of
the Region's event handlers.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="shortcut" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The RegionShortcut for this region. Allows easy initialization of the region based on
pre-defined defaults.
                    ]]></xsd:documentation>
                </xsd:annotation>
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="REPLICATE"/>
                        <xsd:enumeration value="REPLICATE_HEAP_LRU"/>
                        <xsd:enumeration value="REPLICATE_OVERFLOW"/>
                        <xsd:enumeration value="REPLICATE_PERSISTENT"/>
                        <xsd:enumeration value="REPLICATE_PERSISTENT_OVERFLOW"/>
                        <xsd:enumeration value="REPLICATE_PROXY"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </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-template" type="replicatedRegionType">
        <xsd:annotation>
            <xsd:documentation source=
"org.springframework.data.gemfire.ReplicatedRegionFactoryBean"><![CDATA[
Defines a template for creating multiple GemFire REPLICATE Regions that all share a
common attribute configuration.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.Region"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:element>
    <!-- -->
    <xsd:element name="replicated-region" type="replicatedRegionType"/>
    <!-- Disk Store Type -->
    <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=
"org.apache.geode.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 two petabytes.
                            ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <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="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="compaction-threshold" type="xsd:string" default="50">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Sets the threshold at which an oplog will become compactable. Until it reaches this
threshold the oplog will not be compacted.
The threshold is a percentage in the range 0..100. When the amount of garbage in an
oplog exceeds this percentage then when a
compaction is done and this garbage will be cleaned up freeing up disk space. Garbage
is created by entry destroys,
entry updates, and region destroys.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="disk-usage-critical-percentage" type="xsd:string">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Disk usage above this threshold generates an error message and shuts down the member's
```

```
cache. For example,
if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB
drive generates the error
and shuts down the cache.

Set to "0" (zero) to disable. GemFire default is 99%.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="disk-usage-warning-percentage" type="xsd:string">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Disk usage above this threshold generates a warning message. For example, if the
threshold is set to 90%,
then on a 1 TB drive falling under 100 GB of free disk space generates the warning.

Set to "0" (zero) to disable.  GemFire default is 90%.
                ]]></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 created this
amount of file space will be immediately reserved.
                ]]></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 have been
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="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="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" />
    <!-- Eviction -->
    <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=
"org.apache.geode.cache.util.ObjectSizer" />
                        </tool:annotation>
                    </xsd:appinfo>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="threshold" type="xsd:string">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The threshold (or limit) against which the eviction algorithm runs. Once the threshold
```

```
is reached, eviction is performed.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="type" type="xsd:string" default="ENTRY_COUNT">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The 'type' of eviction performed, or algorithm used to perform eviction on the Region
entries.  Eviction types include:
ENTRY_COUNT: Considers the number of entries in the Region before performing an
eviction.
HEAP_PERCENTAGE: Considers the amount of heap used (through the GemFire resource
manager) before performing an eviction.
MEMORY_SIZE: Considers the amount of memory consumed by the Region before performing
an eviction.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- Expiration -->
    <xsd:complexType name="expirationType">
        <xsd:attribute name="timeout" type="xsd:string" default="0">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Number of seconds before a region or an entry expires. If timeout is not specified, it
defaults to zero
(which means no expiration).
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="action" type="xsd:string" default="INVALIDATE">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Action that should take place when a Region or an Entry expires.  Valid values
include: DESTROY, INVALIDATE,
LOCAL_DESTROY, LOCAL_INVALIDATE.  Note, the default GemFire Expiration Action is
INVALIDATE.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="customExpirationType">
        <xsd:sequence>
            <xsd:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Inner bean definition of the CustomExpiry.
                ]]></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 CustomExpiry bean referred by this declaration. Used as a convenience
method. If no reference exists,
use inner bean declarations.
           ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- Functions -->
    <xsd:element name="function-service">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="function" minOccurs="0" maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation source=
"org.apache.geode.cache.execute.Function"><![CDATA[
Declares one or more remote functions for this cache and register's with them the
FunctionService. each bean
must implement org.apache.geode.cache.execute.Function
                            ]]></xsd:documentation>
                        <xsd:appinfo>
                            <tool:annotation>
                                <tool:exports type=
"org.apache.geode.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 reference exists,
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>
    <!-- Function Annotation support -->
    <xsd:element name="annotation-driven">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
    Enables gemfire annotations.
            ]]></xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <!-- Index -->
    <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="org.apache.geode.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 well.
                    ]]></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 index.
                    ]]></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:attribute name="define" type="xsd:string" use="optional" default=
"false">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Boolean condition used to lazily create this index once all indexes with define set to
true are defined
(i.e. declared and defined in the Spring context as Spring beans).
                    ]]></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:annotation>
                    <xsd:documentation><![CDATA[
Corresponds to the regionPath parameter in createIndex methods.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <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="type" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The type of index: FUNCTIONAL, HASH, PRIMARY_KEY.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <!-- Lucene Index & Service -->
```

```
    <xsd:element name="lucene-index">
        <xsd:annotation>
            <xsd:documentation
                source=
"org.springframework.data.gemfire.search.lucene.LuceneIndexFactoryBean"><![CDATA[
Defines a GemFire Lucene index.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.lucene.LuceneIndex"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="field-analyzers" type="beanDeclarationType"
minOccurs="0" maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Mapping of field names to Lucene (per field) Analyzers.
                    ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the LuceneIndex bean definition. If property 'name' is not set, it will be
used as the index name as well.
                    ]]></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 LuceneIndex.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="destroy" type="xsd:string" use="optional" default=
"false">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
```

```
Determines whether the LuceneIndex is destroyed on shutdown.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="fields" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
List of fields included in the Lucene Index.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="lucene-service-ref" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Reference to the single LuceneService.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="region-ref" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Reference to the Region
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="region-path" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Fully-qualified pathname of the Region.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:element name="lucene-service">
        <xsd:annotation>
            <xsd:documentation
                source=
"org.springframework.data.gemfire.search.lucene.LuceneServiceFactoryBean"><![CDATA[
Defines a GemFire LuceneService bean.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.lucene.LuceneService"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:attribute name="id" type="xsd:string">
                <xsd:annotation>
```

```
                    <xsd:documentation><![CDATA[
Identifier (name) for the LuceneService 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:complexType>
    </xsd:element>
    <!-- JNDI -->
    <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" type="xsd:string" use="required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies the DataSource implementation: ManagedDataSource, PooledDataSource,
SimpleDataSource, or XAPooledDataSource.
            ]]></xsd:documentation>
            </xsd:annotation>
        </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 the data type if other than java.lang.String.
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- Transaction Management (Local Cache) -->
    <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: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="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>
    <!-- Client/Server Topology -->
    <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="org.apache.geode.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 notification queues.
                ]]></xsd:documentation>
                    </xsd:annotation>
                    <xsd:complexType>
                        <xsd:attribute name="eviction-type" type="xsd:string" use=
"optional" default="NONE"/>
                        <xsd:attribute name="capacity" type="xsd:string" use="
```

```
optional" default="1"/>
                        <xsd:attribute name="disk-store" type="xsd:string" use=
"optional"/>
                    </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="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="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:complexType>
    </xsd:element>
    <!-- -->
    <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="org.apache.geode.cache.client.Pool" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:choice>
                <xsd:element name="locator" type="connectionType" minOccurs="0"
maxOccurs="unbounded" />
                <xsd:element name="server" type="connectionType" minOccurs="0"
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="keep-alive" type="xsd:string" use="optional" default
="false"/>
            <xsd:attribute name="locators" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Comma-delimited list of Locator endpoints used by this Pool in the form of:
host1[port1],host2[port2],...,hostN[portN]
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <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="servers" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Comma-delimited list of Server endpoints used by this Pool in the form of:
host1[port1],host2[port2],...,hostN[portN]
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <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: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: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="id" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
identifier of the listener (optional)
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <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=
"org.apache.geode.cache.RegionService" />
                        </tool:annotation>
                    </xsd:appinfo>
```

```
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="pool-name" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
The name of the GemFire Pool used by the container.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="auto-startup" type="xsd:string" use="optional"
default="true"/>
            <xsd:attribute name="error-handler" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
A reference to a Spring ErrorHandler strategy handling any errors that may occur when
the container executes the CQs.
                    ]]></xsd:documentation>
                    <xsd:appinfo>
                        <tool:annotation kind="ref">
                            <tool:expected-type type=
"org.springframework.util.ErrorHandler"/>
                        </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="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 CQ listener invokers.
The 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: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 specified listener method.
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="name" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The name of the resulting GemFire ContinuousQuery (CQ). Useful for monitoring and
statistics-based querying.
            ]]></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 ContinuousQueryListener
interface or provide a method named 'handleEvent'.
            ]]></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:complexType name="interestType" abstract="true">
        <xsd:attribute name="durable" type="xsd:string" use="optional" default="false
">
```

```
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Indicates whether the Registered Interest is durable or not. Default is false.
                ]]></xsd:documentation>
            </xsd:annotation>
        </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:attribute name="result-policy" type="xsd:string" use="optional" default=
"KEYS_VALUES">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
The result policy for this interest. Can be one of 'KEYS', '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:attribute>
    </xsd:complexType>
    <!-- WAN Topology -->
    <xsd:attributeGroup name="commonWANQueueAttributes">
        <xsd:attribute name="batch-conflation-enabled" type="xsd:string" use="
optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Boolean value that determines whether GemFire should conflate messages.  GemFire
default is false.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="batch-size" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Maximum number of messages that a batch can contain.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
```

```
        <xsd:attribute name="batch-time-interval" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Maximum number of milliseconds that can elapse between sending batches.
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="disk-store-ref" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Named DiskStore to use for storing the Queue overflow, or for persisting the Queue. If
you specify a value,
the named DiskStore must exist. If you specify a null value, GemFire uses the Default
DiskStore for overflow
and Queue persistence.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="disk-synchronous" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
For Regions that write to disk, a boolean that specifies whether disk writes are done
synchronously (true)
for the Region or asynchronously (false).
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="dispatcher-threads" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Number of dispatcher threads that are used to process Region Events from a
GatewaySender Queue
or Asynchronous Event Queue.
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="maximum-queue-memory" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Maximum amount of memory in megabytes that the Queue can consume before overflowing to
disk.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="order-policy" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
When the dispatcher-threads attribute is greater than 1, order-policy configures the
way in which multiple
dispatcher threads process Region Events from a serial Gateway Queue or serial
Asynchronous Event Queue.
```

```
This attribute can have one of the following values:

KEY - When distributing Region Events from the local Queue, multiple dispatcher
threads preserve the order of key updates.
THREAD - When distributing Region Events from the local Queue, multiple dispatcher
threads preserve the order
in which a given thread added Region Events to the Queue.
PARTITION - When distributing Region Events from the local Queue, multiple dispatcher
threads preserve the order
in which Region Events were added to the local Queue. For a Partitioned Region, this
means that all Region Events
delivered to a specific partition are delivered in the same order to the remote
GemFire site. For a Distributed Region,
this means that all key updates delivered to the local GatewaySender Queue are
distributed to the remote site
in the same order.

You cannot configure the order-policy for a parallel Event Queue, because parallel
Queues cannot preserve event
ordering for Regions. Only the ordering of events for a given partition (or in a given
Queue of a Distributed Region)
can be preserved.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="parallel" type="xsd:string">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Value of "true" or "false" specifying the type of GatewaySender or AsyncEventQueue
that GemFire creates.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="persistent" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Boolean value that determines whether GemFire persists the Gateway Queue or
AsyncEventQueue.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:attributeGroup>
    <!-- -->
    <xsd:complexType name="baseAsyncEventQueueType">
        <xsd:annotation>
            <xsd:documentation source="org.apache.geode.cache.wan.AsyncEventQueue"
><![CDATA[
An async event queue definition (requires Gemfire 7.0 or later)
           ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
```

```
                    <tool:exports type="org.apache.geode.cache.wan.AsyncEventQueue" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="async-event-listener" minOccurs="1" maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation
                        source="org.apache.geode.cache.wan.AsyncEventListener"
><![CDATA[
An AsyncEventListener bean definition for this AsyncEventQueue. (requires Gemfire 7.0)
                            ]]></xsd:documentation>
                    <xsd:appinfo>
                        <tool:annotation>
                            <tool:exports type=
"org.apache.geode.cache.wan.AsyncEventListener"/>
                        </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 async 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 async event listener bean referred by this declaration. Used as a
convenience method. If no reference exists,
use inner bean declarations.
                                    ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Optionally specifies the GemFire AsyncEventQueue id. By default this value is the bean
id or a generated value
if an inner bean.
                  ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
```

```
        <xsd:attributeGroup ref="commonWANQueueAttributes" />
    </xsd:complexType>
    <!-- -->
    <xsd:element name="async-event-queue">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="baseAsyncEventQueueType">
                    <xsd:attribute name="id" type="xsd:string" use="required">
                        <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:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:complexType name="baseGatewayReceiverType">
        <xsd:annotation>
            <xsd:documentation source="org.apache.geode.cache.wan.GatewayReceiver"
><![CDATA[
A gateway receiver definition (requires Gemfire 7.0 or later)
           ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.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="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: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="hostname-for-senders" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Attribute where you can specify an IP address or hostname for gateway sender
connections.
If you configure hostname-for-senders, locators will use the provided hostname or IP
address
when instructing gateway senders on how to connect to gateway receivers. If you
provide ""
or null as the value, by default the gateway receiver's bind-address will be sent to
clients.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <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="manual-start" type="xsd:string" default="false">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Specifies if the gateway receiver is manually (true) or automatically (false) started.
Default is an automatic start (false).
            ]]></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:complexType>
    <!-- -->
    <xsd:element name="gateway-receiver" type="baseGatewayReceiverType" />
    <!-- -->
    <xsd:complexType name="baseGatewaySenderType">
        <xsd:annotation>
            <xsd:documentation source="org.apache.geode.cache.wan.GatewaySender"
><![CDATA[
A gateway sender gateway definition (requires Gemfire 7.0 or later)
              ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.wan.GatewaySender" />
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="event-filter" type="gatewayEventFilterType" minOccurs=
"0" maxOccurs="1"/>
            <xsd:element name="event-substitution-filter" type=
"gatewayEventSubstitutionFilterType" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="transport-filter" type="gatewayTransportFilterType"
minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Optionally specifies the GemFire GatewaySender id. By default, this value is the bean
id or a generated value
if an inner bean.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="remote-distributed-system-id" type="xsd:string" use=
"required">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Integer that uniquely identifies the remote GemFire cluster to which this
```

```
GatewaySender will send Region Events.
This value corresponds to the distributed-system-id property specified in Locators for
the remote cluster.
This attribute is required.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="alert-threshold" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Maximum number of milliseconds that a Region Event can remain in the GatewaySender
Queue before GemFire logs an alert.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="enable-batch-conflation" type="xsd:string" use="optional
">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Boolean value that determines whether GemFire should conflate messages. GemFire
default is false.
NOTE, this attribute is deprecated in favor of the common WAN Queue attribute, 'batch-
conflation-enabled'.  If both
attributes are specified, then 'batch-conflation-enabled' takes precedence.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="manual-start" type="xsd:string" default="false">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Boolean value that specifies whether you need to manually start the GatewaySender. If
you supply a null value,
the default is "false" and the GatewaySender attempts to start automatically.
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="socket-buffer-size" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Size of the socket buffer that sends messages to remote sites. This size should match
the size of the
socket-buffer-size attribute of remote GatewayReceivers that process Region Events.
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="socket-read-timeout" type="xsd:string" use="optional">
            <xsd:annotation>
                <xsd:documentation><![CDATA[
Amount of time in milliseconds that the GatewaySender will wait to receive an
acknowledgment from a remote site.
By default this is set to 0, which means there is no timeout. If you do set this
```

```
timeout, you must set it to
a minimum of 30000 (milliseconds). Setting it to a lower number will generate an error
message and reset the value
to the default of 0.
            ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attributeGroup ref="commonWANQueueAttributes"/>
    </xsd:complexType>
    <!-- -->
    <xsd:element name="gateway-sender">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="baseGatewaySenderType">
                    <xsd:attribute name="id" type="xsd:string" use="required">
                        <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:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:complexType name="gatewayEventFilterType">
        <xsd:annotation>
            <xsd:documentation source="org.apache.geode.cache.wan.GatewayEventFilter"
><![CDATA[
A Gateway Event Filter for this GatewaySender.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type="org.apache.geode.cache.wan.GatewayEventFilter
"/>
                </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(s) to declare and add GatewayEventFilter(s) to the
GatewaySender.
                    ]]></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 GatewaySender Event Filter bean referred to by this declaration.
Used for convenience. If no reference exists, use inner bean declarations.
                    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
    <!-- -->
    <xsd:complexType name="gatewayEventSubstitutionFilterType">
        <xsd:annotation>
            <xsd:documentation source=
"org.apache.geode.cache.wan.GatewayEventSubstitutionFilter"><![CDATA[
A Gateway Event Substitution Filter for this GatewaySender.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type=
"org.apache.geode.cache.wan.GatewayEventSubstitutionFilter"/>
                </tool:annotation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
Inner bean definition to declare and add a single GatewayEventSubstitutionFilter to
the GatewaySender.
                    ]]></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 GatewaySender Event Sustitution Filter bean referred to by this
declaration.
Used for convenience. If no reference exists, use inner bean declarations.
                    ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
```

```
    <!-- -->
    <xsd:complexType name="gatewayTransportFilterType">
        <xsd:annotation>
            <xsd:documentation source=
"org.apache.geode.cache.wan.GatewayTransportFilter"><![CDATA[
A Gateway Transport Filter for this GatewaySender.
            ]]></xsd:documentation>
            <xsd:appinfo>
                <tool:annotation>
                    <tool:exports type=
"org.apache.geode.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(s) for the Gateway Transport Filter(s) to add to this
GatewaySender.
                    ]]></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 GatewaySender Transport Filter bean referred to by this declaration.
Used for convenience. If no reference exists, use inner bean declarations.
                ]]></xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>
</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:context="http://www.springframework.org/schema/context"
            xmlns:gfe="http://www.springframework.org/schema/gemfire"
            xmlns:repository="http://www.springframework.org/schema/data/repository"
            xmlns:tool="http://www.springframework.org/schema/tool"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.springframework.org/schema/data/gemfire"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            version="2.0">
```

```
    <xsd:import namespace="http://www.springframework.org/schema/beans"/>
    <xsd:import namespace="http://www.springframework.org/schema/context"
                schemaLocation="http://www.springframework.org/schema/context/spring-
context.xsd" />
    <xsd:import namespace="http://www.springframework.org/schema/data/repository"
                schemaLocation=
"http://www.springframework.org/schema/data/repository/spring-repository.xsd"/>
    <xsd:import namespace="http://www.springframework.org/schema/gemfire"
                schemaLocation="http://www.springframework.org/schema/gemfire/spring-
gemfire.xsd"/>
    <xsd:import namespace="http://www.springframework.org/schema/tool"/>
    <!-- -->
    <xsd:annotation>
        <xsd:documentation><![CDATA[
            Namespace support for the Spring Data GemFire Client side data access.
        ]]></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>
```

```
    <!-- Function Executions -->
    <xsd:element name="function-executions">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
   Enables component scanning for annotated function execution interfaces.
             ]]></xsd:documentation>
        </xsd:annotation>

        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="include-filter" type="context:filterType"
minOccurs="0" maxOccurs="unbounded">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
   Controls which eligible types to include for component scanning.
                           ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="exclude-filter" type="context:filterType"
minOccurs="0" maxOccurs="unbounded">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
   Controls which eligible types to exclude for component scanning.
                       ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="base-package" type="xsd:string" use="required">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
                Defines the base package where function execution interfaces will be
tried to be detected.
                ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <!-- DataSource -->
    <xsd:element name="datasource">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
Defines a connection from a Cache client to a set of GemFire Cache Servers.
            ]]></xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:choice minOccurs="1" maxOccurs="1">
                <xsd:element name="locator" type="gfe:connectionType"
                             minOccurs="1" maxOccurs="unbounded" />
                <xsd:element name="server" type="gfe:connectionType"
                             minOccurs="1" maxOccurs="unbounded" />
            </xsd:choice>
```

```
            <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>
    <!-- JSON support -->
    <xsd:element name="json-region-autoproxy">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
Enables A Spring AOP proxy to perform automatic conversion to and from JSON for
appropriate region operations
            ]]></xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:attribute name="region-refs" use="optional" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
```

```
A comma delimited string of region names to include for JSON conversion. By default
all regions are included.
                ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="pretty-print" use="optional" type="xsd:string">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
A boolean value to specify whether returned JSON strings are pretty printed, false by
default.
                ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="convert-returned-collections" use="optional" type=
"xsd:string">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
A boolean value to specify whether Collections returned by Region.getAll(),
Region.values() should be converted from the
native GemFire PdxInstance type. True, by default but will incur significant overhead
for large collections.
                ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <!-- Snapshot Service support -->
    <xsd:element name="snapshot-service">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
Access to GemFire's Snapshot Service for taking snapshots of GemFire Cache and Region
data.
            ]]></xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="snapshot-import" type="snapshotMetadataType"
minOccurs="0" maxOccurs="unbounded">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Specifies meta-data for a snapshot import.
                        ]]></xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="snapshot-export" type="snapshotMetadataType"
minOccurs="0" maxOccurs="unbounded">
                    <xsd:annotation>
                        <xsd:documentation><![CDATA[
Specifies meta-data for a snapshot export.
                        ]]></xsd:documentation>
                    </xsd:annotation>
```

```
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:string" use="required">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
ID of the GemFire [Cache|Region] SnapshotService bean in the Spring context.
                ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="cache-ref" type="xsd:string" use="optional" default=
"gemfireCache">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
(Optional) Name of the GemFire Cache bean from which to extract data and record a
snapshot (by default 'gemfireCache').
                ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="region-ref" type="xsd:string" use="optional">
                <xsd:annotation>
                    <xsd:documentation><![CDATA[
(Optional) Name of the GemFire Region bean from which to extract data and record a
snapshot.
                    ]]></xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="suppress-import-on-init" type="xsd:string" default=
"false">
                <xsd:annotation>
                    <xsd:documentation>
                        Determines whether imports are suppressed on initialization of
the GemFire Snapshot Service.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
    <!-- -->
    <xsd:complexType name="snapshotMetadataType">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
Declares an element type defining snapshot meta-data.
            ]]></xsd:documentation>
        </xsd:annotation>
        <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="location" type="xsd:string" use="required"/>
        <xsd:attribute name="format" type="xsd:string" use="optional" default="
GEMFIRE"/>
        <xsd:attribute name="filter-ref" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:schema>
```