1.0.1

Preface

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

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

Part I. Introduction

This document is the reference guide for Spring GemFire project (SGF). It explains the relationship between Spring framework and GemFire Enterprise Fabric (GEF) 6.0.x/6.5.x, defines the basic concepts and semantics of the integration and how these can be used effectively.

Part II. Reference Documentation

Document structure

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

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

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

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

Chapter 1. Bootstrapping GemFire through the Spring container

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

1.1. Using the Spring GemFire Namespace

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

```
<?xml version="1.0" encoding="UTF-8"?>
<objects xmlns="http://www.springframework.net"
         xmlns:gfe❶="http://www.springframework.net/gemfire❷">

  <gfe:cache ...>❸

</objects>
```

❶ Spring GemFire namespace prefix. Any name can do but through out the reference documentation, the `gfe` will be used.
❷ The namespace URI.
❸ Declaration example for the GemFire namespace. Notice the prefix usage.

For the remainder of this doc, to improve readability, the XML examples will simply refer to the `<gfe>` namespace without the namespace declaration, where possible. To enable Spring.NET to recognize this namespace you need to register it in the applications App.config section.

```
<configuration>

  <configSections>
    <sectionGroup name="spring">
      <!-- other Spring config sections handler like context, typeAliases, etc not shown for brevity -->
      <section name="parsers" type="Spring.Context.Support.NamespaceParsersSectionHandler, Spring.Core"/>
     </sectionGroup>
  </configSections>

  <spring>
    <parsers>
      <parser type="Spring.Data.GemFire.Config.GemfireNamespaceParser, Spring.Data.GemFire" />
    </parsers>
  </spring>

</configuration>
```

1.2. Configuring the GemFire Cache

In order to use the GemFire Fabric, one needs to either create a new Cache or connect to an existing one. As of the current version of GemFire (6.0.x) there can be only one opened cache per application. In most cases the cache is created once and then all other consumers connect to it.

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

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

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

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

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

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

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

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

  <object name="props" type="System.Collections.Specialized.NameValueCollection">
    <property name="['log-level']" value="warning"/>
    <!-- set properties such as durable-client-id, durable-timeout for a durable cache -->
  </object>
```

Other properties that can be set in the namespace are `disconnect-on-close`, `keepalive-on-close`, and `distributed-system-name` as shown below.

```
  <gfe:cache disconnect-on-close="false" keepalive-on-close="true" distributed-system-name="MySystemName"/>
```

The disconnect-on-close property is set to true by default. It should be changed to false in case you encounter a race condition in the 3.0.0.5 client library that results in the disconnect call hanging the application.

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

```
  <object name="cache-with-props" type="Spring.Data.GemFire.CacheFactoryObject, Spring.Data.GemFire">
    <property name="Properties">
      <name-values>
        <add key="log-level" value="warning"/>
      </name-values>
    </property>
  </object>
```

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

```
<object name="default-cache" type="Spring.Data.GemFire.CacheFactoryObject, Spring.Data.GemFire"/>
```

Here, the *default-cache* will try to connect to an existing cache and, in case one does not exist, create a local client cache. Since no additional properties were specified the created cache uses the default cache configuration.

The name of the cache will be the name of the Spring object definition unless you specify a different name using the NameValueCollection property described next. Note, you can specify the name of the underlying DistributedSystem that will be created with the property `DistributedSystemName`

Especially in environments with opened caches, this basic configuration can go a long way. For scenarios where the cache needs to be configured, the user can pass in a reference the GemFire configuration file:

```
<object name="cache-with-xml" type="Spring.Data.GemFire.CacheFactoryObject, Spring.Data.GemFire">
  <property name="CacheXml" value="cache.xml"/>
</object>
```

In this example, if the cache needs to be created, it will use the file named `cache.xml` located in the runtime directory.

In addition to referencing an external configuration file one can specify GemFire settings directly through .NET name value properties. This can be quite handy when just a few settings need to be changed:

```
<object name="cache-with-props" type="Spring.Data.GemFire.CacheFactoryObject, Spring.Data.GemFire">
  <property name="Properties">
    <name-values>
      <add key="log-level" value="warning"/>
    </name-values>
  </property>
</object>
```

The 'name' property is used to set the name of the Cache object. For a complete list of properties refer to the GemFire reference documentation.

Spring object definitions support property replacement through the use of variable replacement. The following configuration allows you to externalize the properties from the Spring configuration file which is a best practice. The Spring configuration file is usually an embedded assembly resource so as to prevent accidental changes in production. There are seven locations supported out of the box in Spring.NET where you can place your externalized configuration data and can be extended to support your own locations. In the following example the configuration flues would come from a name-value configuration section in App/Web.config

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

<object name="props" type="System.Collections.Specialized.NameValueCollection">
  <property name="['log-level']" value="${cache.log-level}"/>
</object>

<object type="Spring.Objects.Factory.Config.VariablePlaceholderConfigurer, Spring.Core">
   <property name="VariableSources">
      <list>
         <object type="Spring.Objects.Factory.Config.ConfigSectionVariableSource, Spring.Core">
            <property name="SectionNames" value="CacheConfiguration" />
         </object>
      </list>
   </property>
</object>
```

The CacheConfiguration section in App.config would then look like the following

```
<configuration>
  <configSections>
    <section name="CacheConfiguration" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>

  <CacheConfiguration>
    <add key="cache.log-level" value="warning"/>
  </CacheConfiguration>

</configuration>
```

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

1.3. Configuring a GemFire Region

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

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

1.3.1. Using an externally configured Region

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

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

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

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

Note

If the region does not exist, an initialization exception will be thrown. For configuring new GemFire regions proceed to the sections below for client region and advanced client region configuration.

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

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

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

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

1.3.2. Client Region

GemFire supports various deployment topologies for managing and distributing data. For .NET the currenlty supported topoloy is client-server. Additional topologies such as peer-to-peer may be supported in future releases of the .NET client as they are supported in the Java client already. To declare *client* regions which connect to a backing cache server, SGF offers dedicated support for such configuration through the `client-region` and `pool` elements. As the name imply, the former defines a client region while the latter connection pools to be used/ shared by the various client regions.

Below is a usual configuration for a client region:

```
<gfe:cache/>

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

```
<object id="c-listener" type="Spring.Data.GemFire.Tests.SimpleCacheListener, Spring.Data.GemFire.Tests"/>

<!-- pool declaration -->
<gfe:pool id="gemfire-pool" subscription-enabled="false">
    <gfe:server host="localhost" port="40404"/>
</gfe:pool>
```

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

The client-region can also reference multiple cache-listeners:

```
<gfe:cache/>

  <gfe:client-region id="complex" pool-name="gemfire-pool">
    <gfe:cache-listener>
      <ref object="c-listener"/>
      <object type="Spring.Data.GemFire.Tests.SimpleCacheListener, Spring.Data.GemFire.Tests"/>
    </gfe:cache-listener>

  </gfe:client-region>

  <object id="c-listener" type="Spring.Data.GemFire.Tests.SimpleCacheListener, Spring.Data.GemFire.Tests"/>

  <!-- pool declaration -->
  <gfe:pool id="gemfire-pool" subscription-enabled="false">
    <gfe:server host="localhost" port="40404"/>
  </gfe:pool>
```

Other options to configure the client-region are to specify a specific cache by name, set the and persistence of the reigon via the persistent attribute, and also to set the various RegionAttributes. RegionAttributes are set using the attributes-ref element:

```
  <gfe:client-region id="simple" pool-name="gemfire-pool" attributes-ref="myCustomRegionAttributes" />

  <!-- some common base cache configuration that can be used later in parent object referencts -->
  <object id="cachingProxy" type="Spring.Data.GemFire.RegionAttributesFactoryObject, Spring.Data.GemFire">
    <property name="ClientNotification" value="true"/>
  </object>

  <object id="myCustomRegionAttributes"
          parent="cachingProxy"
          type="Spring.Data.GemFire.RegionAttributesFactoryObject, Spring.Data.GemFire">
    <property name="LruEntriesLimit" value="1234"/>
  </object>
```

The use of Spring's parent referent allows you set setup commonly used configurations and then extend them by overriding or adding additional properties.

For a full list of options to set on the client and especially on the pool, please refer to the SGF schema (Appendix A, *Spring.NET for Gemfire's spring-gemfire-1.0.xsd*) and the GemFire documentation.

1.3.2.1. Client Interests

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

```
  <gfe:client-region id="complex" pool-name="gemfire-pool">

    <gfe:all-keys-interest durable="true" result-policy="Keys"/>

    <gfe:key-interest result-policy="KeysAndValues" key-ref="cacheableString"/>
    <gfe:regex-interest pattern=".*"/>

  </gfe:client-region>

  <object id="cacheableString" type="GemStone.GemFire.Cache.CacheableString, GemStone.GemFire.Cache">
    <constructor-arg index="0" value="hello"/>
  </object>
```

1.4. Advanced Region Creation

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

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

```
<object name="basic" type="Spring.Data.GemFire.ClientRegionFactoryObject, Spring.Data.GemFire">
  <property Name="Cache">
    <object type="Spring.Data.GemFire.CacheFactoryObject, Spring.Data.GemFire"/>
  </property>
</object>
```

By default the region name is the name of the object definition unless explicitly specified using the `Name` property. Notice how the GemFire cache definition has been nested into the declaring region definition. Let's add more regions and make the cache a top level object.

Since the region object definition name is usually the same with that of the cache, the `name` property can be omitted (the object name will be used automatically).

```
<!-- shared cache across regions -->
<object id="cache" type="Spring.Data.GemFire.CacheFactoryObject, Spring.Data.GemFire"/>

<!-- region named 'basic' -->
<object name="basic" type="Spring.Data.GemFire.ClientRegionFactoryObject, Spring.Data.GemFire">
  <property name="Endpoints" value="localhost:40404"/>
  <property name="Cache" ref="Cache"/>
</object>

<!-- region with a name different then the object definition -->
<object name="basic" type="Spring.Data.GemFire.ClientRegionFactoryObject, Spring.Data.GemFire">
  <property name="Endpoints" value="localhost:40404"/>
  <property name="Cache" ref="Cache"/>
  <property name="Name" value="default-region"/>
</object>
```

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

```
  <object name="baseRegion" abstract="true">
    <property name="Endpoints" value="localhost:40404"/>
    <property name="Cache" ref="Cache"/>   <!-- definition not shown here -->
  </object>

  <object name="listeners" type="Spring.Data.GemFire.ClientRegionFactoryObject, Spring.Data.GemFire"
          parent="baseRegion">
    <property name="CacheListener">
      <object type="Spring.Data.GemFire.Tests.SimpleCacheListener, Spring.Data.GemFire.Tests">
        <!-- set properties or constructor arguments -->
      </object>
    </property>
    <property name="CacheLoader">
      <object type="Spring.Data.GemFire.Tests.SimpleCacheLoader, Spring.Data.GemFire.Tests"/>
    </property>
    <property name="CacheWriter">
      <object type="Spring.Data.GemFire.Tests.SimpleCacheWriter, Spring.Data.GemFire.Tests"/>
    </property>
  </object>
```

1.4.1. Configuring update interest for *client* Region

For scenarios where a *CacheServer* is used and *clients* need to be configured and the namespace is not an option, SGF offers a dedicated configuration class named: ClientRegionFactoryObject. This allows client *interests* to be registered in both key and regex form through AllKeysInterest, KeyInterest, and RegexInterest classes in the `Spring.Data.GemFire` namespace. Here is an example of how to configure the AllKeys interest in the region.

```
  <object name="baseRegion" abstract="true">
    <property name="Endpoints" value="localhost:40404"/>
    <property name="Cache" ref="Cache"/>   <!-- Cache definition not shown here -->
    <property name="Attributes">
      <object type="Spring.Data.GemFire.RegionAttributesFactoryObject, Spring.Data.GemFire">
        <property name="ClientNotification" value="true"/>
      </object>
    </property>
  </object>

  <object name="region-all-keys-interest" type="Spring.Data.GemFire.ClientRegionFactoryObject, Spring.Data.GemFire"
          parent="baseRegion">
    <property name="interests">
      <list>
        <object type="Spring.Data.GemFire.AllKeysInterest"/>
      </list>
    </property>
  </object>
```

This example makes use of Spring's object configuration inheritance allowing you to place common configuration in the baseRegion object definition and then refer to it later via the `parent` attribute.

Users that need fine control over a region, can configure it in Spring by using the `Attributes` property. To ease declarative configuration in Spring, SGF provides an RegionAttributesFactoryObject. The previous examples shows configuring the baseRegion by embedding the use of the RegionAttributesFactoryObject:

To register interest for a set of key, use the KeyInterest class, as shown below from the sample application

```
  <object name="region" type="Spring.Data.GemFire.ClientRegionFactoryObject, Spring.Data.GemFire"
                 parent="baseRegion">  <!-- baseRegion definition not shown here -->
    <property name="Interests">
      <list>
        <object type="Spring.Data.GemFire.KeyInterest">
          <property name="Keys">
            <list>
              <object type="GemStone.GemFire.Cache.CacheableString" factory-method="Create">
                <constructor-arg value="Key-123"/>
              </object>
            </list>
          </property>
        </object>
      </list>
    </property>
  </object>
```

To register interest based on a regular expression, use the following configuration

```
  <object name="Region" type="Spring.Data.GemFire.ClientRegionFactoryObject, Spring.Data.GemFire"
          parent="baseRegion">  <!-- baseRegion definition not shown here -->
    <property name="Interests">
      <list>
        <object type="Spring.Data.GemFire.RegexInterest">
```

```
          <property name="Regex" value="Key-.*"/>
        </object>
      </list>
    </property>
  </object>
```

This would only register interest in keys of the type 'Key-123' or 'Key-4abc'.

See below an example of configuring a region through Spring XML embedding the use of the RegionAttributesFactoryObject, no parent object definition is used.

```
  <object name="Region" type="Spring.Data.GemFire.ClientRegionFactoryObject, Spring.Data.GemFire"
    <property name="Endpoints" value="localhost:40404"/>
    <property name="Cache" ref="Cache"/>
    <property name="Name" value="exampleregion"/>
    <property name="Attributes">
      <object type="Spring.Data.GemFire.RegionAttributesFactoryObject, Spring.Data.GemFire">
        <property name="ClientNotification" value="true"/>
      </object>
    </property>
    <property name="Interests">
      <list>
        <object type="Spring.Data.GemFire.RegexInterest">
          <property name="Regex" value="Key-.*"/>
        </object>
      </list>
    </property>
  </object>
```

Please refer to the API documentation for more information on the various IInterest subclasses.

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

With SGF, GemFire regions, pools and cache can be configured either through Spring or directly inside GemFire, native, `cache.xml` file. While both are valid approaches, it's worth pointing out that Spring's powerful DI container and AOP functionality makes it very easy to wire GemFire into an application. This lalso lets you leverage features such as replacement of varaiable placeholders, e..g. ${port}. Also configuring a region cache loader, listener and writer through the Spring container is preferred since the same instances can be reused across multiple regions. Spring object definition inheritance also allows you to create base region and region attribute object definitions that can later be referred to using Spring's 'parent' attribute.

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

Chapter 2. Working with the GemFire APIs

Once the GemFire cache and regions have been configured they can injected and used inside application objects. This chapter describes the integration with Spring's DaoException hierarchy.

2.1. Exception translation

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

Chapter 3. Using Caching Advice with GemFire

An implementation of Spring.NET's ICache interface is provided. You can read about the features of Spring.NET Caching advice here [http://www.springframework.net/doc-latest/reference/html/aop-aspect-library.html#caching-aspect]. In this documentation we will show a basic configuration of the caching advice.

3.1. Caching Advice

The caching advice allows you to use string values as the keys and a GemFire .NET serialized objects as the values. To apply the Gemfire Caching advice you need to create an object definition for Spring.Data.GemFire.Caching.GemFireCache and pass a reference to a GemFire region in the constructor. In this example, the caching advice is being applied to a data access repository for Inventors. The inventors The repository is identified as a candidate to apply caching advice through the use of `[Repository]` attribute.

```
<objects xmlns="http://www.springframework.net"
         xmlns:gfe="http://www.springframework.net/gemfire">

  <object id="inventorstore" type="Spring.Data.GemFire.Tests.InventorRepository, Spring.Data.GemFire.Tests"/>

  <!-- the ID matches the name used in the [Cache] attribute -->
  <object id="inventors" type="Spring.Data.GemFire.Caching.GemFireCache, Spring.Data.GemFire">
    <constructor-arg ref="exampleregion"/>
  </object>

  <!-- Defines multiple Advisors for Caching - the CacheResultAdvisor, CacheParameterAdvisor, and InvalidateCacheAdvisor-->
  <object id="CacheAspect" type="Spring.Aspects.Cache.CacheAspect, Spring.Aop"/>


  <!-- The AutoProxy based on attributes to apply the advisors defined in the CacheAspect -->
  <object type="Spring.Aop.Framework.AutoProxy.AttributeAutoProxyCreator, Spring.Aop">
    <!-- AttributeTypes selects the classes that have the RepositoryAttribute at the class level-->
    <property name="AttributeTypes" value="Spring.Stereotype.RepositoryAttribute"/>
    <!-- Interceptor names can be either of the type IAdvice, IAdvisor, or IAdvisors -->
    <!-- The CacheAspect is of the type IAdvisors -->
    <property name="InterceptorNames" value="CacheAspect"/>
  </object>


  <!-- default name is gemfire-cache -->
  <gfe:cache/>

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

  <gfe:client-region id="exampleregion" pool-name="gemfire-pool"  />

</objects>
```

Spring's CacheAspect class implements the IAdvisors interface, so that it can define multiple IAdvisors, one for each of the attributes used in the caching advice: `[CacheResult]`, `[CacheParameter]` and `[InvalidateCache]`. The InventorRepository class is shown below, it is backed by a simple dictionary, but in a real world scenario it would be backed by a database.

```
    [Serializable]
    [Repository]
    public sealed class InventorRepository : IInventorRepository
    {
        private IDictionary inventors = new ListDictionary();
        public InventorRepository()
        {
            Inventor tesla = new Inventor("Nikola Tesla", new DateTime(1856, 7, 9), null);
            Inventor pupin = new Inventor("Mihajlo Pupin", new DateTime(1854, 10, 9), null);
            inventors.Add("Nikola Tesla", tesla);
            inventors.Add("Mihajlo Pupin", pupin);
        }

        [CacheResultItems("inventors", "Name")]
        public IList GetAll()
        {
            return new ArrayList(inventors.Values);
        }


        [CacheResult(CacheName = "inventors")]
        public IList GetAllNoCacheKey()
        {
            return new ArrayList(inventors.Values);
        }

        [CacheResult("inventors", "#name")]
        public Inventor Load(string name)
        {
            return (Inventor)inventors[name];
        }

        public void Save([CacheParameter("inventors", "Name")] Inventor inventor)
        {
            inventor.Nationality = "Serbian";
        }

        [InvalidateCache("inventors", Keys = "#inventor.Name")]
        public void Delete(Inventor inventor)
        {
        }

        [InvalidateCache("inventors")]
        public void DeleteAll()
        {
        }
    }
```

The IInventorRepository interface is shown below. Inventors have the `[Serializable]` attribute applies to them.

```
    public interface IInventorRepository
    {
        IList GetAll();
        IList GetAllNoCacheKey();
        Inventor Load(string name);
        void Save(Inventor inventor);
        void Delete(Inventor inventor);
        void DeleteAll();
    }
```

Chapter 4. Sample Applications

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

The sample is bundled with the distribution and is included in the main solution file. You can also run it from the command line once it is built in visual studio.

4.1. Prerequisites

1. You will need to download, install, and obtain a license for

- GemFire Enterprise 6.5
- GemFire Enterprise Native Client 3.5.0

The GemFire Enterprise licence file, gemfireLicense.zip should reside in the root of your GemFire Enterprise install directory. The GemFire Enterprise Native Client licence file, gfCpplicense.zip, should reside in the 'bin' sub-directory of the Native Client install directory.

The .NET clients run in a client-server architecture. This is shown below from the perspective of a single 'native (C++ or .NET) client and a single cache server process.

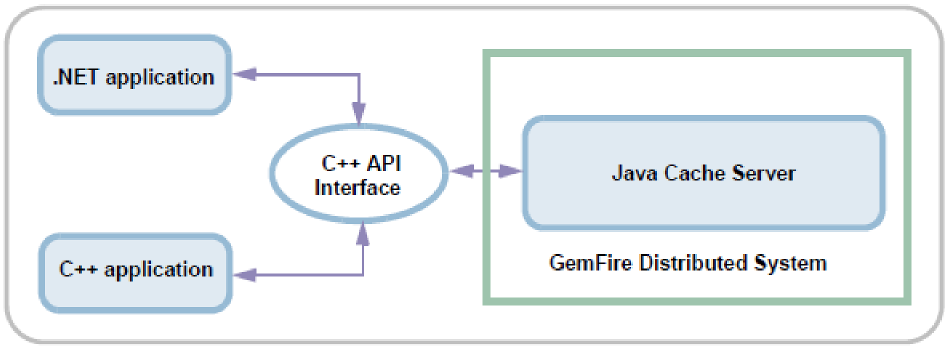

Figure 4.1. Client-Server Configuration for native (C++ or .NET) clients

While not shown in this picture, there can be multiple .NET/C++ client applications and also multiple other processes that act in a peer-to-peer like manner as part of the GemFire Distributed System.

When running in a client-server architecture there needs to be configuration of the client side, the .NET/C++ Application, and the data grid side, the Java Cache Server. In this example the configuration of the client side is done entirely by the FactoryObject's declared in the Spring XML file. The data grid side is configured with a standalone configuration file.

2. The Java Cache Server requires a configuration file in order to run. This file is provided in the Spring GemFire distribution and in named `cache.xml`. It is located in the 'example\Spring.Data.Gemfire.HelloWorld' directory. It needs to be into the GemFire Enterprise 6.0 'bin' directory where the cacheserver.bat file is located.

3. Run the cacheserver.bat file located in the 'bin' of the GemFire Enterprise 6.0 product. By default it will load a configuration file named '`cache.xml`'.

The Spring GemFire project ships with the essential GemFire client side libraries to run the sample application. Note that because the .NET Client API is a wrapper around the C++ API, it needs to be referenced in either the App.config file or installed into the GAC. The HelloWorld example program is configured to load the libraries from the 'lib\GemFire\net\2.0' directory.

Despite providing the client However, it is recommended that you download the GemFire Enterprise Native Client 3.0.0.9 from the download site to have access to reference documentation. You may also find the GemFire .NET API Tour of interest to read.

4.2. Hello World

The Hello World sample demonstrates the basic functionality of the Spring GemFire project and is also useful to understand how GemFire clients work. The application bootstraps GemFire, configures it, allows for the execution of several commands against the data grid, and gracefully shuts down when the application exits. Multiple instances can be started at the same time as they will work with each other sharing data without any user intervention.

4.2.1. Compiling, starting and stopping the sample

Hello World is designed as a stand-alone application. The main class is in the file `Program.cs` and generates an executable named `HelloWorld.exe`. To start the example follow the steps

To compile the example, load the solution `Spring.Data.GemFire.sln` in the root of the Spring GemFire project directory.

1. Load the Visual Studio 2008 solution Spring.Data.GemFire.sln located in the root of the Spring GemFire project directory. Compile the solution.

2. Ensure that the Java Cache Server is running as described in the previous section.

3. Set the startup project to be the Spring.Data.Gemfire.HelloWorld project **or** cd to the '`example\Spring.Data.Gemfire.HelloWorld\bin\Debug`' directory and run `HelloWorld.exe`.

   Note

   You can pass as a command line argument the name that will appear before you enter commands in the shell. This is useful for distinguishing different members of the distributed system

4. You can now execute commands in the shell, which will be described in the next section. Exit the shell by typing '`exit`'.

4.2.2. Using the sample

Once started, the sample will create a client side cache that is replicated with the server cache contained in the Java Cache Server. For example, the command line '`HelloWorld.exe client-1`' will result in the following greeting

```
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
size - returns the size of the grid
clear - removes all mapping in the grid
```

```
keys - returns the keys contained by the grid
values - returns the values contained by the grid
containsKey <key> - indicates if the given key is contained by the grid
containsValue <value> - indicates if the given value is contained by the grid
map - returns a list of the key-value pairs in the grid

query <query> - executes a query on the grid

help - this info
exit - this node exists
client-1>
```

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

```
client-1>put 1 unu
null
client-1>put 1 one
old value = [unu]
client-1>size
1
client-1>put 2 two
null
client-1>size
2
client-1>
```

Multiple instances can be created at the same time. Once started, the new clients automatically see the existing region and its information. Start a second client with the command line '`HelloWorld.exe client-2`'

```
Hello World!
...

client-2>size
2
client-2>map
[2=two][1=one]
client-2>
```

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, the Java Cache Servier needs to be running at all times.

Part III. Other Resources

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

Chapter 5. Useful Links

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

Part IV. Appendices

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

Appendix A. Spring.NET for Gemfire's `spring-gemfire-1.0.xsd`

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns="http://www.springframework.net/gemfire"
           xmlns:objects="http://www.springframework.net"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:vs="http://schemas.microsoft.com/Visual-Studio-Intellisense"
           targetNamespace="http://www.springframework.net/gemfire"
           elementFormDefault="qualified" attributeFormDefault="unqualified"
           vs:friendlyname="Spring.NET Gemfire Framework Configuration" vs:ishtmlschema="false" vs:iscasesensitive="true" vs

    <xs:import namespace="http://www.springframework.net"/>

    <xs:annotation>
        <xs:documentation>
            Namespace support for the Spring GemFire project.
        </xs:documentation>
    </xs:annotation>

    <xs:element name="cache">
  <xs:annotation>
   <xs:documentation>
Defines a GemFire Cache instance used for creating or retrieving 'regions'.
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:attribute name="id" type="xs:ID" use="optional">
    <xs:annotation>
     <xs:documentation>
The name of the cache definition (by default "gemfire-cache").</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="cache-xml-location" type="xs:string" use="optional">
    <xs:annotation>
     <xs:documentation>
The location of the GemFire cache xml file.
       </xs:documentation>
    </xs:annotation>
   </xs:attribute>
      <xs:attribute name="disconnect-on-close" type="xs:boolean" default="true" use="optional">
        <xs:annotation>
          <xs:documentation>
Indicates whether to call DistributedSystem.Disconnect when this cache is disposed.
There is a bug in the 3.0.0.9 client that may hang calls to close.  The default is
true, set to false if you experience a hang in the application.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="keepalive-on-close" type="xs:boolean" default="true" use="optional">
        <xs:annotation>
          <xs:documentation>
Indicates whether to whether to keep a durable client's queue alive when this cache is closed.
The default is true if the cache has been configured as a durable client.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="distributed-system-name" type="xs:string" use="optional">
        <xs:annotation>
          <xs:documentation>
The name of the GemFire Distributed System.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="properties-ref" type="xs:string" use="optional">
    <xs:annotation>
     <xs:documentation>
The object name of a NameValuesCollection 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.
```

```
        </xs:documentation>
     </xs:annotation>
    </xs:attribute>
   </xs:complexType>
  </xs:element>

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

  <xs:complexType name="basicRegionType">
   <xs:attribute name="id" type="xs:string" use="required">
    <xs:annotation>
     <xs:documentation>
The id of the region object definition.
     </xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="name" type="xs:string" use="optional">
    <xs:annotation>
     <xs:documentation>
The name of the region definition. If no specified, it will have the value of the id attribute (that is, the object name).
     </xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="cache-ref" type="xs:string" default="gemfire-cache" use="optional">
    <xs:annotation>
     <xs:documentation>
The name of the object defining the GemFire cache (by default 'gemfire-cache').
     </xs:documentation>
    </xs:annotation>
   </xs:attribute>
     <xs:attribute name="attributes-ref" type="xs:string" use="optional">
       <xs:annotation>
         <xs:documentation>
The object name of a RegionAttributesFactoryObject that will be used to configure detailed region properties.
         </xs:documentation>
       </xs:annotation>
     </xs:attribute>
  </xs:complexType>


  <xs:complexType name="readOnlyRegionType" abstract="true">
   <xs:complexContent>
    <xs:extension base="basicRegionType">
     <xs:sequence>
      <xs:element name="cache-listener" minOccurs="0" maxOccurs="1">
       <xs:annotation>

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.
        </xs:documentation>
       </xs:annotation>
       <xs:complexType>
        <xs:sequence>
         <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="skip">
             <xs:annotation>
                  <xs:documentation>
Inner object definition of the cache listener.
          </xs:documentation>
         </xs:annotation>
               </xs:any>
        </xs:sequence>
        <xs:attribute name="ref" type="xs:string" use="optional">
         <xs:annotation>
         <xs:documentation>
The name of the cache listener object referred by this declaration. Used as a convenience method. If no reference exists,
use inner object declarations.
         </xs:documentation>
         </xs:annotation>
        </xs:attribute>
       </xs:complexType>
      </xs:element>
      <xs:element name="disk-store" type="diskStoreType" minOccurs="0" maxOccurs="1">
       <xs:annotation>
        <xs:documentation>
Disk storage configuration for the defined region.
        </xs:documentation>
       </xs:annotation>
      </xs:element>
     </xs:sequence>
     <xs:attribute name="persistent" type="xs:boolean" default="false">
      <xs:annotation>
       <xs:documentation>
Indicates whether the defined region is persistent or not. 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.
       </xs:documentation>
      </xs:annotation>
     </xs:attribute>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="regionType">
   <xs:complexContent>
    <xs:extension base="readOnlyRegionType">
     <xs:sequence minOccurs="0" maxOccurs="1">
      <xs:element name="cache-loader" minOccurs="0"  maxOccurs="1" type="beanDeclarationType">
       <xs:annotation>
        <xs:documentation source="com.gemstone.gemfire.cache.CacheLoader">
The cache loader definition for this region. A cache loader allows data to be placed into a region.
        </xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="cache-writer" minOccurs="0" maxOccurs="1" type="beanDeclarationType">
       <xs:annotation>
        <xs:documentation source="com.gemstone.gemfire.cache.CacheWriter">
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.
        </xs:documentation>
       </xs:annotation>
      </xs:element>
     </xs:sequence>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

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

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

  <xs:complexType name="evictionType">
   <xs:sequence minOccurs="0" maxOccurs="1">
    <xs:element name="object-sizer" type="beanDeclarationType">
     <xs:annotation>
      <xs:documentation>
Entity computing sizes for objects stored into the grid.
      </xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="type" default="ENTRY_COUNT">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="ENTRY_COUNT">
       <xs:annotation>
        <xs:documentation>
Considers the number of entries in the region before performing an eviction.
        </xs:documentation>
       </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="MEMORY_SIZE">
       <xs:annotation>
        <xs:documentation>
Considers the amount of memory consumed by the region before performing an eviction.
        </xs:documentation>
       </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="HEAP_PERCENTAGE">
       <xs:annotation>
        <xs:documentation>
Considers the amount of heap used (through the GemFire resource manager) before performing an eviction.
        </xs:documentation>
       </xs:annotation>
      </xs:enumeration>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="threshold" type="xs:long" use="required">
    <xs:annotation>
     <xs:documentation>
The threshold (or limit) against which the eviction algorithm runs. Once the threashold is reached, eviction is
performed.
     </xs:documentation>
    </xs:annotation>
   </xs:attribute>
  </xs:complexType>

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

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

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

Note: the directory must already exist.
        </xs:documentation>
       </xs:annotation>
      </xs:attribute>
      <xs:attribute name="max-size" type="xs:int" default="10240">
       <xs:annotation>
        <xs:documentation>
The maximum size (in megabytes) of data stored in each directory. Default is 10240 MB (10 gigabytes).
        </xs:documentation>
       </xs:annotation>
      </xs:attribute>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="synchronous-write" type="xs:boolean" default="false">
    <xs:annotation>
     <xs:documentation>
Indicates whether the writing to the disk si synchronous or not. Default is false, meaning asynchronous writing.

     </xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="auto-compact" type="xs:boolean" default="true">
    <xs:annotation>
     <xs:documentation>
Indicates whether or not the operation logs are automatically compacted or not. Default is true.

     </xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <!--
   <xs:attribute name="compaction-threshold" default="50">
    <xs:annotation>
     <xs:documentation>
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 this garbage will be cleaned up freeing up disk space. Garbage is created
by entry destroys, entry updates, and region destroys.

     </xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:short">
      <xs:minExclusive value="0"/>
      <xs:maxExclusive value="100"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
    -->
   <xs:attribute name="max-oplog-size" type="xs:long" default="1024">
    <xs:annotation>
     <xs:documentation>
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.

     </xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="time-interval" type="xs:long" default="1">
    <xs:annotation>
     <xs:documentation>
Sets the number of milliseconds that can elapse before unwritten data is written to disk.
It is considered only for asynchronous writing.
```

```
    </xs:documentation>
   </xs:annotation>
  </xs:attribute>
  <xs:attribute name="queue-size" type="xs:long" default="0">
   <xs:annotation>
    <xs:documentation>
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.

    </xs:documentation>
   </xs:annotation>
  </xs:attribute>
 </xs:complexType>

 <xs:element name="client-region">
  <xs:annotation>
   <xs:documentation source="org.springframework.data.gemfire.client.ClientRegionFactoryObject">
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.
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="readOnlyRegionType">
     <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element name="key-interest">
        <xs:annotation>
         <xs:documentation>
Key based interest. If the key is a List, then all the keys in the List will be registered.
         </xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:complexContent>
          <xs:extension base="interestType">
           <xs:sequence minOccurs="0" maxOccurs="1">
            <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="skip">
                <xs:annotation>
                     <xs:documentation>
Inner object definition of the client key interest.
              </xs:documentation>
             </xs:annotation>
                   </xs:any>
           </xs:sequence>
           <xs:attribute name="key-ref" type="xs:string" use="optional">
            <xs:annotation>
             <xs:documentation>
The name of the client key interest object referred by this declaration. Used as a convenience method. If no reference exist
use the inner object declaration.
             </xs:documentation>
            </xs:annotation>
           </xs:attribute>
          </xs:extension>
         </xs:complexContent>
        </xs:complexType>
       </xs:element>
              <xs:element name="all-keys-interest">
                <xs:annotation>
                  <xs:documentation>

Register interest for all the keys of the region to get updates from the server. Valid only for a Native Client region
when client notification is true.

                  </xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:complexContent>
                    <xs:extension base="interestType">
                    </xs:extension>
                  </xs:complexContent>
                </xs:complexType>
              </xs:element>
       <xs:element name="regex-interest">
        <xs:annotation>
```

```
         <xs:documentation>
Regular expression based interest. If the pattern is '.*' then all keys of any type will be pushed to the client.
         </xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:complexContent>
          <xs:extension base="interestType">
           <xs:attribute name="pattern" type="xs:string"/>
          </xs:extension>
         </xs:complexContent>
        </xs:complexType>
       </xs:element>
      </xs:choice>
      <xs:element name="eviction" minOccurs="0" maxOccurs="1">
       <xs:annotation>
        <xs:documentation>
Eviction policy for the partitioned region.
        </xs:documentation>
       </xs:annotation>
       <xs:complexType>
        <xs:complexContent>
         <xs:extension base="evictionType">
          <xs:attribute name="action" type="evictionActionType" default="LOCAL_DESTROY">
           <xs:annotation>
            <xs:documentation>
The action to take when performing eviction.
            </xs:documentation>
           </xs:annotation>
          </xs:attribute>
         </xs:extension>
        </xs:complexContent>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
     <xs:attribute name="pool-name" use="required" type="xs:string">
      <xs:annotation>
       <xs:documentation>
The name of the pool used by this client.
        </xs:documentation>
      </xs:annotation>
     </xs:attribute>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

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

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

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

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