Table of Contents

[Introduction 1](#__RefHeading__1271_1077372440)

[Converting the tutorials 1](#__RefHeading__1273_1077372440)

[Spring Gemfire Migration Guidelines 3](#__RefHeading__1275_1077372440)

[Part 2: Create a Cache and Start Peers 3](#__RefHeading__1277_1077372440)

[Single approach to Gemfire developer examples and quickstart 4](#__RefHeading__1279_1077372440)

[Coverage is subset of developer topics that are supported by Spring Data Gemfire 5](#__RefHeading__1281_1077372440)

[Highlight STS and springsource capabilities 6](#__RefHeading__1283_1077372440)

[Issues to resolve 6](#__RefHeading__1285_1077372440)

[Workplan and labor estimate 7](#__RefHeading__1287_1077372440)

Appendix – Edited Tutorial 7

[Using the Tutorial 7](#__RefHeading__1291_1077372440)

[Using the Tutorial 7](#__RefHeading__1293_1077372440)

[Prerequisites 7](#__RefHeading__1295_1077372440)

[Prerequisites 7](#__RefHeading__1297_1077372440)

[Part 1: Start a Locator 7](#__RefHeading__1299_1077372440)

[Part 1: Start a Locator 7](#__RefHeading__1301_1077372440)

[Part 2: Create a Cache and Start Peers 7](#__RefHeading__1303_1077372440)

[Part 2: Create a Cache and Start Peers 7](#__RefHeading__1305_1077372440)

[Fix up points here with spring- gemfire approach. 7](#__RefHeading__1307_1077372440)

[2.Run the Peer application in two terminal sessions. 7](#__RefHeading__1309_1077372440)

[Part 3: Create Replicated Regions 7](#__RefHeading__1311_1077372440)

[Part 3: Configure Replicated Regions 7](#__RefHeading__1313_1077372440)

[</gfe:replicated-region> 7](#__RefHeading__1315_1077372440)

[Part 4: Create Partitioned Regions 7](#__RefHeading__1317_1077372440)

[Part 4: Create Partitioned Regions 7](#__RefHeading__1319_1077372440)

[Part 5: Set Up Client/Server Caching 7](#__RefHeading__1321_1077372440)

# Introduction

Gemfire is a key component of the vFabric solution and is one of the most important products in strategically positioning Gemfire as a leader in the NoSQL solution space. SpringSource is the leading producer in java frameworks and developers who have learned spring expect to feel the spring ambience in every product they use within the SpringSource portfolio. Therefore it is essential that the Gemfire tutorials and examples lead with spring gemfire as the preferred solution approach for java developers.

The goals for converting the vFabric Gemfire tutorials are:

1. Convert the tutorials to use the annotations and configuration options made available through Spring Data Gemfire (SDG).
2. Have a single place where the tutorial and examples are found whereas today there are multiple locations.
3. Have a single preferred format for the published documentation.
4. Evangelize this approach with PSO to ensure that when Spring resources and gemfire resources are brought to bear on a project that they deliver with the same message.

# Converting the tutorials

GemFire is a broad technology and SDG does not encompass the breadth of the solution at this point in time. Therefore the scope of the tutorial migration is to include all of the tutorials that have SDG solutions and open jira issues against SDG where developer functionality is not available.

Currently within the com.gemstone tutorials the API is used directly within the code rather than using the simplifications offered through SDG. The simple HelloWorld will serve as a succint example:

1. Convert directory structure from freeform to maven/gradle structured project. This converts to:
   * src/main/java
   * src/main/resources
   * src/test/java
   * src/test/resources

Figure Figure : src code structure

One approach is to:

* Create new Spring project and point it to the specific tutorial example . [Note: experimented with two approaches; 1) manually created project and moved src code from the gemfire projects to the respective directory. 2) created a roo project. The latter end up being more work simply because the gradle.build and gradle.properties are already working in the spring-date-gemfire example.
* Use the Java Build Path to fixup a consistent src build that corresponds to mvn / gradle structure (e..g see Figure 1: src code structure)
* Copy build.gradle and gradle.properties into the project in the same location at the root of the project.
* Copy the context files into the same location with src/main/resources

1. Convert the project from hardcoded classpath to gradle enabled dependencies.
2. Move the gemfire properties and resources to src/main/resources so that we have app-context.xml, cache-context.xml and cache.properties (gemfire.properties).
3. Spring Gemfire setup. At this point we come to the “Part 1: Start a Locator”. For this we take the approach introduced by Filip Hanik on the AA PoC. A LocatorUtil.launch is available for the project. Simply launch the LocatorUtil. It requires spring-cache-locator.xml in the resources directory.
4. Checkpoint the project to confirm src code structure, classpaths, gradle dependencies, etc are all working correct.

* By now there should be no more compile errors and the code should work as a checkpoint. (May need to do a project->clean).   
    
  + From Run: Run Configurations launch LocatorUtil
  + Launch whatever code is in the project (e.g. examples, tutorial, or quickstart) to ensure they are working.

Follow the exercises. This will verify that the code all works before migration to spring gemfire.

* + Below is the examples from the Gemfire Hello World.

1. Start the code migration to spring gemfire semantics according to the “Spring Gemfire Migration Guidelines”.
2. Update the web page according to the doc template. (See below for docbook example)

# Spring Gemfire Migration Guidelines

## Part 2: Create a Cache and Start Peers

You will store the data on several GemFire peers. The first step to starting up a GemFire peer is to create a com.gemstone.gemfire.cache.Cache. The cache is the central component of GemFire. It manages connections to other GemFire peers. For more information, see [Cache Management](#the_cache) and [Configuring the Gemfire Cache](#bootstrap:cache:server).

Procedure

[editor’s note: In the existing documentation the user is doing more code review than code creation. To convert to Spring Gemfire we need to place configuration in cache-context.xml and leverage annotations to make the code more declarative. We’ll also leverage some additional util beans for avoiding the hardcoding of cache-servers to locators.

1. Look at the cache creation in the initPeer method in GemfireDAO. The first thing this method does is create a cache:

Cache cache = new CacheFactory() .set("locators", "localhost[55221]") .set("mcast-port", "0") .set("log-level", "error") .create();

Spring supports the idea of components. See cache-context.xml for spring configurations:

<!-- GemFire cache bean -->

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

<util:properties id=*"props"* location=*"cache.properties"*/>

cache.properties contains:

log-level=warning

name=Spring GemFire Tutorial

Secondly, it configures the cache: 

* + The GemFire locators property tells the cache which locators to use to discover other GemFire JVMs.
  + The mcast-port property tells GemFire not to use multicast discovery to find peers.
  + The log-level property controls the log level of GemFire's internal logging. Here it is set to "error" to limit the amount of messaging that will show up in the console.

When this code is run, after the call to create finishes, this peer will discover other peers and connect to them. 

* + Run the Peer application in two terminal sessions. You already have one window open where you started the locator. Start another terminal window. In each window, run the Peer application:

$ java com.gemstone.gemfire.tutorial.Peer

The peers start up and connect to each other.

# Single approach to Gemfire developer examples and quickstart

There are known to be four sources for gemfire developer documentation of which the fourth is probably deprecated but currently not completely migrated to the new vFabric Documentation center.

1. Spring-data-gemfire: XML docbook
2. Grails gemfire plugin documentation: do not know what they use for gdoc but its popular and easy to read.
3. Vfabric Gemfire Documentation center – very complete but does not have a spring feel to it.
4. Community.gemfire.com (deprecated ?)

I’ve listed this as an issue because spring users are familiar with the docbook format. Grails users are familiar with another doc tool, which I can not recall the name of. And Gemfire uses a different template approach, which seems to be targeted at unifying the content with all of the vFabric products.

If the docbook format is used the process is:

1. Create a docbook page and move content into the page.
2. Merge into the docbook structure
3. Publish and review the content.

Here’s quick example:

which publishes to:

# Coverage is subset of developer topics that are supported by Spring Data Gemfire

It is recognized that there is not a complete coverage of gemfire from spring. It’s meant to increase developer productivity and simplify configuration. One of the goals of the project is to:

1. Confirm that all of the developer topics covered in QuickStart, examples, and the tutorial are covered within the spring-data-gemfire tutorials and that the functionality is supported.
2. Get a group consensus that spring is a simpler, cleaner approach for ramping developers.
3. Have a clear understanding of when it isn’t appropriate for the task at hand.
4. Identify any gaps that should have been covered by the API but are not.

# Highlight STS and springsource capabilities

(This section is just notes. Will flesh out later but its meant to address the value of the tool, especially in the benchmarking section and integrating with AppInsight).

1. Provide demo versions for STS and cmdline.
2. Gradle enable the demoes for mvn exec:java cmdline.
3. Cover the Getting Started, Quick Start Examples
4. Remove the “Integrating GemFire with Spring IoC Container – it’s the wrong message.

# Issues to resolve

1. What build system are we going to use? vFabric-Gemfire does not use a build system. The tutorial prescribes setting the CLASSPATH variable to include helloworld, tutorial, etc. whereas SDG uses gradle. The suggestion would be to convert to gradle as it is very easy to understand and can run from the command line or within STS.
2. What documentation format should be followed? vFabric-=Gemfire’s documentation as already been productized into the format on vFabric’s website. SDG has also been productized and is the standard format that all spring documentation comes with. The leaning would be to have the vFabric site point to Spring Gemfire’s site.
3. Where will the git location be for this and will it be available to the public?
   * If we put tutorials with spring-data-gemfire the git location is already public.
   * Or we allow for the download from gemfire site.
   * Do we want to track downloads.
4. What about visibility to starting and stopping locators and cacheservers? In the spring environment they are automagically launched whereas in the gemfire tutorials one page is devoted to starting locators and servers.

# Workplan and labor estimate

This is not resource loaded yet to account for two people working part time. Here’s a picture of the tasks. The description of the tasks are in section #2.

# Appendix – Edited Tutorial

|  |  |
| --- | --- |
| Running the Gemfire Tutorial | Running the Spring Gemfire Tutorial |
| Using the Tutorial The GemFire tutorial walks you through several GemFire features. The tutorial includes the following parts:    * [Part 1: Start a Locator](#tutorial_overview__section_53DB74E97CAC4B5F885B2A2FEC279EA6) * [Part 2: Create a Cache and Start Peers](#tutorial_overview__section_DAB47B1B42F445C9BE83B817AA772AD1) * [Part 3: Create Replicated Regions](#tutorial_overview__section_D1EB696CD2FD467CA8A7C8B847A2EBEC) * [Part 4: Create Partitioned Regions](#tutorial_overview__section_F8D85A1CE8074243A6B2F54171C5D98E) * [Part 5: Set Up Client/Server Caching](#tutorial_overview__section_A4C0352B63E2446F8BF73DB21E6B8D88) * [Part 6: Add and Stop Cache Servers (Peers)](#tutorial_overview__section_E728D18E3A194745B0696C430B50C02E) * [Part 7: Configure Persistence](#tutorial_overview__section_7EB34C41E4184DBAA1D4EAF2EFD082B8)   Before you use the tutorial, make sure you have completed the prerequisites listed in the following section. | Using the Tutorial The GemFire tutorial walks you through several GemFire features. The tutorial includes the following parts:    * Part 1: Installing Spring Tool Suite (STS) * Importing skeleton project * [Part 3: Start a Locator](#tutorial_overview__section_53DB74E97CAC4B5F885B2A2FEC279EA6) * [Part 4: Create a Cache and Start Peers](#tutorial_overview__section_DAB47B1B42F445C9BE83B817AA772AD1) * [Part 5: Create Replicated Regions](#tutorial_overview__section_D1EB696CD2FD467CA8A7C8B847A2EBEC) * [Part 6: Create Partitioned Regions](#tutorial_overview__section_F8D85A1CE8074243A6B2F54171C5D98E) * [Part 7: Set Up Client/Server Caching](#tutorial_overview__section_A4C0352B63E2446F8BF73DB21E6B8D88) * [Part 8: Add and Stop Cache Servers (Peers)](#tutorial_overview__section_E728D18E3A194745B0696C430B50C02E) * [Part 9: Configure Persistence](#tutorial_overview__section_7EB34C41E4184DBAA1D4EAF2EFD082B8)   Before you use the tutorial, make sure you have completed the prerequisites listed in the following section. |
| Prerequisites  * Install and configure vFabric GemFire and the product examples. See [Installing vFabric GemFire](#setting_up_gemfire) and [Setting Up the Product Examples](#setting_up_the_product_examples). * Verify that your system has Java 1.6 or later. * Open several terminal sessions. * Use a text editor or your favorite IDE to open the GemfireDAO.java file in the tutorial application code. This file is located $SamplesDirectory/tutorial/src/com/gemstone/gemfire/tutorial/storage directory where $SamplesDirectory corresponds to the location where you unzipped the product examples distribution. | Prerequisites  * Install and configure vFabric GemFire and the product examples. See [Installing vFabric GemFire](#setting_up_gemfire) and [Setting Up the Product Examples](#setting_up_the_product_examples). * Verify that your system has Java 1.6 or later. * Open several terminal sessions. * Import vfgf-tutorial into STS to open the GemfireDAO.java file in the tutorial application code. This file is located $SamplesDirectory/tutorial/src/com/gemstone/gemfire/tutorial/storage directory where $SamplesDirectory corresponds to the location where you unzipped the product examples distribution. |
| Part 1: Start a Locator JVMs running GemFire discover each other through multicast messaging or through a TCP locator service, which is called a locator. Multicasting is a good way to run quick tests, but it is less robust and flexible than the locator service. The locator runs as a separate process to which each new member connects to first discover the list of available peers. Clients connect to the locator to get server connection information. For more information, see [How Member Discovery Works](#how_member_discovery_works). For this example, we use a locator.  *Figure 1. Peer Discovery Using a Locator*   Procedure     1. Start a locator.   $ gemfire start-locator -port=55221  The locator process runs in the background, listening for connections on port 55221. To stop the process, you can use "gemfire stop-locator." But don't stop it yet. | Part 1: Start a Locator JVMs running GemFire discover each other through multicast messaging or through a TCP locator service, which is called a locator. Multicasting is a good way to run quick tests, but it is less robust and flexible than the locator service. The locator runs as a separate process to which each new member connects to first discover the list of available peers. Clients connect to the locator to get server connection information. For more information, see [How Member Discovery Works](#how_member_discovery_works). For this example, we use a locator.   1. Start a locator.   This launches a locator with the following arguments (see Arguments tab) -Ddata.location=${project\_loc:/vfgf-tutorial}/target/data.  The locator utility is configured through the spring-cache-locator and uses the common.properties file for it source of runtime arguments. See  <context:property-placeholder system-properties-mode="OVERRIDE" location="classpath:/common.properties" />  in spring-cache-locator.xml. When you launch the LocatorUtil configuration it creates the Locator class, which instantiates the spring bean factory through this line:  String resource = ("spring-cache-locator.xml");  ClassPathXmlApplicationContext mainContext = **new** ClassPathXmlApplicationContext(**new** String[] {resource}, **false**);  mainContext.setValidating(**true**);  mainContext.refresh();  The locator process runs in the background, listening for connections on port the port configured in common.properties, “locator.port1=10334”. To stop the locator simply switch to the debug view and click the terminate action.  But don’t stop it yet. |
| Part 2: Create a Cache and Start Peers You will store the data on several GemFire peers. The first step to starting up a GemFire peer is to create a com.gemstone.gemfire.cache.Cache. The cache is the central component of GemFire. It manages connections to other GemFire peers. For more information, see [Cache Management](#the_cache).  Procedure     1. Look at the cache creation in the initPeer method in GemfireDAO. The first thing this method does is create a cache:   Cache cache = new CacheFactory() .set("locators", "localhost[55221]") .set("mcast-port", "0") .set("log-level", "error") .create();  Secondly, it configures the cache:    * + The GemFire locators property tells the cache which locators to use to discover other GemFire JVMs.   + The mcast-port property tells GemFire not to use multicast discovery to find peers.   + The log-level property controls the log level of GemFire's internal logging. Here it is set to "error" to limit the amount of messaging that will show up in the console.   When this code is run, after the call to create finishes, this peer will discover other peers and connect to them.    * + Run the Peer application in two terminal sessions. You already have one window open where you started the locator. Start another terminal window. In each window, run the Peer application:   $ java com.gemstone.gemfire.tutorial.Peer  The peers start up and connect to each other. | Part 2: Create a Cache and Start Peers You will store the data on several GemFire peers. The first step to starting up a GemFire peer is to create a com.gemstone.gemfire.cache.Cache. The cache is the central component of GemFire. It manages connections to other GemFire peers. For more information, [2.2 Configuring the GemFire Cache](http://static.springsource.org/spring-gemfire/docs/current/reference/bootstrap.html). This is stored within src/main/resources, the standard location for the location configurations, in the file cache-context.xml.  Procedure   1. Look at the cache creation in cache-context.xml. This cache is created when the app-context is initialized by the Spring Bean Factory Lifecycle process.   <!-- GemFire cache bean -->  <gfe:cache properties-ref=*"props"* />  <util:properties id=*"props"* location=*"cache.properties"*/>  The cache.properties contain the configuration of the log-level and name for the application:  log-level=warning  name=Spring GemFire Tutorial Fix up points here with spring- gemfire approach.Run the Peer application in two terminal sessions. The peers connect to each other. …. |
| Part 3: Create Replicated Regions The GemFire com.gemstone.gemfire.cache.Region interface defines a key-value collection. Region extends the java.util.concurrent.ConcurrentMap interface. The simplest type of region is a replicated region. Every peer that hosts a replicated region stores a copy of the entire region locally. Changes to the replicated region are sent synchronously to all peers that host the region. For more information on regions, see [Data Regions](#data_regions).  This procedure walks you through the creation of a replicated region called People.      1. Look at the initPeer method in GemfireDAO.java to see where it creates the people region.   To create a com.gemstone.gemfire.cache.Region, you use a com.gemstone.gemfire.cache.RegionFactory class. This is the code in initPeer method that creates the people region:  people = cache.<String, Profile>createRegionFactory(REPLICATE) .addCacheListener(listener) .create("people");  The people region is constructed with com.gemstone.gemfire.cache.RegionShortcut.REPLICATE, which tells the factory to start with the configuration for a replicated region. This method adds a cache listener to the region. You can use a cache listener to receive notifications when the data in the region changes. This sample application includes a LoggingCacheListener class, which prints changes to the region to System.out and lets you see how entries are distributed.   1. Look at the addPerson method. It adds the entry to the region by calling the put method of Region.   public void addPerson(String name, Profile profile) { people.put(name, profile); }  Calling put on the people region distributes the person to all other peers that host that region. After this call completes, each peer will have a copy of this person.   1. Add people. In one of your terminal windows, type:   person Isabella person Ethan  You will see the users show up in the other window:  In region people created key Isabella value Profile [friends=[]] In region people created key Ethan value Profile [friends=[]] | Part 3: Configure Replicated Regions <!-- people region -->  <gfe:replicated-region id=*"people"*>  <gfe:cache-listener>  <bean class=*"com.gemstone.gemfire.tutorial.storage.LoggingCacheListener"*/>  </gfe:cache-listener> </gfe:replicated-region> the use of Spring annotations. See  For more information on regions, see [2.3.2 Replicated Region](http://static.springsource.org/spring-gemfire/docs/current/reference/bootstrap.html).   1. Look at the GemfireDao.java instance variable declarations. We inject the regions into the class through spring annotations:   @Resource(name = "people")  **private** Region<String, Profile> people;  name is resolved through the id=”people” in the configuration file. Notice that the cache listener was also configured and is automatically injected through spring’s bean lifecycle management.   1. Look at the addPerson method. It adds the entry to the region by calling the put method of Region.   public void addPerson(String name, Profile profile) { people.put(name, profile); }  Calling put on the people region distributes the person to all other peers that host that region. After this call each peer will have a copy of this person.   1. Add people. In one of your terminal windows, type:   person Isabella person Ethan  You will see the users show up in the other window:  In region people created key Isabella value Profile [friends=[]] In region people created key Ethan value Profile [friends=[]] |
| Part 4: Create Partitioned Regions You expect to create a lot of posts. Because of that, you do not want to host a copy of the posts on every server. Thus you store them in a partitioned region. The API to use the partitioned region is the same as that for a replicated region, but the data is stored differently. A partitioned region lets you control how many copies of your data will exist in the distributed system. The data is partitioned over all peers that host the partitioned region. GemFire automatically maintains the number of copies of each partition that you request.  Procedure    1. Look at the code that creates the posts region. You can create partitioned regions with the PARTITION\_XXX shortcuts.   To create the posts region, the initPeer method uses thecom.gemstone.gemfire.cache.RegionShortcut.PARTITION\_REDUNDANT shortcut to tell GemFire to create a partitioned region that keeps one primary and one redundant copy of each post on different machines.  posts = cache.<PostID, String>createRegionFactory(PARTITION\_REDUNDANT) .addCacheListener(listener) .create("posts");   1. Start another terminal window and launch the peer application in that window.   You should have three peers running now. Each post you create will be stored in only two of these peers.   1. Add some posts to the posts region.   > post Isabella I like toast > post Ethan Waaa! > post Ethan Hello  You see that the listener in only one of the JVMs is invoked for each post. That's because partitioned regions make one copy of the post the primary copy. By default GemFire only invokes the listener in the peer that holds the primary copy of each post.   1. From any window, list the available posts with the posts command.   You should be able to list all posts, because GemFire fetches them from the peer that hosts each post.  > posts Ethan: Waaa! Isabella: I like toast Ethan: Hello  If you kill one of the JVMs, you should still be able to list all posts. You can bring that member back up and kill another one, and you should still see all of the posts.   1. Kill your peers before moving on to the next section. Type quit in each window. | Part 4: Create Partitioned Regions You expect to create a lot of posts. Because of that, you do not want to host a copy of the posts on every server. Thus you store them in a partitioned region. The API to use the partitioned region is the same as that for a replicated region, but the data is stored differently. A partitioned region lets you control how many copies of your data will exist in the distributed system. The data is partitioned over all peers that host the partitioned region. GemFire automatically maintains the number of copies of each partition that you request.  Procedure    1. Look at the spring configuration that creates the posts region.   <!-- posts region -->  <gfe:partitioned-region id=*"posts"*>  <gfe:cache-listener>  <bean class=*"com.gemstone.gemfire.tutorial.storage.LoggingCacheListener"*/>  </gfe:cache-listener>  </gfe:partitioned-region>  To use the posts region, we annotate an instance variable within GemfireDao in this manner: @Resource(name = "posts")  **private** Region<PostID, String> posts;   1. Start another terminal window and launch the peer application in that window.   You should have three peers running now. Each post you create will be stored in only two of these peers.   1. Add some posts to the posts region.   > post Isabella I like toast > post Ethan Waaa! > post Ethan Hello  You see that the listener in only one of the JVMs is invoked for each post. That's because partitioned regions make one copy of the post the primary copy. By default GemFire only invokes the listener in the peer that holds the primary copy of each post.   1. From any window, list the available posts with the posts command.   You should be able to list all posts, because GemFire fetches them from the peer that hosts each post.  > posts Ethan: Waaa! Isabella: I like toast Ethan: Hello  If you kill one of the JVMs, you should still be able to list all posts. You can bring that member back up and kill another one, and you should still see all of the posts.   1. Kill your peers before moving on to the next section. Type quit in each window. |
| Part 5: Set Up Client/Server Caching You have a fully working system now, but the UI code is running in the same member in which you are storing data. That works well for some use cases, but it may be undesirable for others. For example, if you have a Web server for the UI layer, you may want to increase or decrease the number of Web servers without modifying your data servers. Or you may need to host only 100 GB of data, but have thousands of clients that might access it. For these use cases, it makes more sense to have dedicated GemFire servers and access the data through a GemFire client.  GemFire servers are GemFire peers, but they also listen on a separate port for connections from GemFire clients. GemFire clients connect to a limited number of these cache servers.  Like regular peers, GemFire servers still need to define the regions they will host. You could take the peer code you already have and create a com.gemstone.gemfire.cache.server.CacheServer instance programmatically, but this example uses the cacheserver script that ships with GemFire. The cacheserver script reads the cache configuration from a cache xml file, which is a declarative way to define the regions in the cache.  Procedure     1. Walk through configuring GemFire with XML. All GemFire configuration that you can do in java you can also do in xml. Look at the server.xml file in the xml directory. This file creates the same replicated and partitioned regions as the java code in the GemfireDAO.initPeer method, demonstrated in the preceding procedure.   ?xml version="1.0"?> <!DOCTYPE cache PUBLIC "-//GemStone Systems, Inc.//GemFire Declarative Caching 6.6//EN" "http://www.gemstone.com/dtd/cache6\_6.dtd"> <cache> <region name="people" refid="REPLICATE"> <region-attributes> <cache-listener> <class-name> com.gemstone.gemfire.tutorial.storage.LoggingCacheListener </class-name> </cache-listener> </region-attributes> </region> <region name="posts" refid="PARTITION\_REDUNDANT"> <region-attributes> <cache-listener> <class-name> com.gemstone.gemfire.tutorial.storage.LoggingCacheListener </class-name> </cache-listener> </region-attributes> </region> </cache>   1. From the tutorial directory, start two cache servers.   $ mkdir server1 $ cacheserver start locators=localhost[55221] mcast-port=0 \ cache-xml-file=../xml/server.xml -server-port=0 -dir=server1 $ mkdir server2 $ cacheserver start locators=localhost[55221] mcast-port=0 \ cache-xml-file=../xml/server.xml -server-port=0 -dir=server2  The cacheserver script starts a GemFire peer (cache server) that listens for client connections. The cache servers should now be running. Here is what all those command line parameters mean.  **locators**  List of locator hosts and ports that discover other cache servers.  **mcast-port**  Multicast port that discovers other cache servers. 0 means do not use multicast.  **cache-xml-file**  Where to find the cache xml file to use, relative to the working directory of the server.  **server-port**  Port on which to listen for GemFire clients. 0 means the server will listen on an ephemeral port, which is a temporary port assigned to the process by the OS.  **dir**  The working directory of the server. Logs for the server are written to this directory.   1. Start a client.   Starting a GemFire client is very similar to starting a GemFire peer. In the GemFire client, you create an instance ofcom.gemstone.gemfire.cache.client.ClientCache, which connects to the locator to discover servers. Look at the GemfireDAO.initClient method. The first thing the method does is create a ClientCache:  ClientCache cache = new ClientCacheFactory() .addPoolLocator("localhost", 55221) .setPoolSubscriptionEnabled(true) .setPoolSubscriptionRedundancy(1) .set("log-level", "error") .create();  Once you create a ClientCache, it maintains a pool of connections to the servers similar to a JDBC connection pool. However, withGemFire you do not need to retrieve connections from the pool and return them. That happens automatically as you perform operations on the regions. The pool locator property tells the client how to discover the servers. The client uses the same locator that the peers do to discover cache servers. Setting the subscription enabled and subscription redundancy properties allow the client to subscribe to updates for entries that change in the server regions. You are going to subscribe to notifications about any people that are added. The updates are sent asynchronously to the client. Because the updates are sent asynchronously, they need to be queued on the server side. The subscription redundancy setting controls how many copies of the queue are kept on the server side. Setting the redundancy level to 1 means that you can lose 1 server without missing any updates.   1. Create a proxy region, posts, in the client.   Creating regions in the client is similar to creating regions in a peer. There are two main types of client regions, PROXY regions andCACHING\_PROXY regions. PROXY regions store no data on the client. CACHING\_PROXY regions allow the client to store keys locally on the client. This example uses a lot of posts, so you won't cache any posts on the client. You can create a proxy region with the shortcut PROXY. Look at the GemFireDAO.initPeer method. This method creates the posts region like this:  posts = cache.<PostID, String>createClientRegionFactory(PROXY) .create("posts");   1. Create a caching proxy region, people, in the client.   You do not have many people, so for this sample the client caches all people on the client. First you create a region that has local storage enabled. You can create a region with local storage on the client with ClientRegionShortcut.CACHING\_PROXY. In the initClient method, here's where the people region is created.  people = cache.<String, Profile>createClientRegionFactory(CACHING\_PROXY) .addCacheListener(listener) .create("people");   1. Call the registerInterest method to subscribe to notifications from the server.   By creating a CACHING\_PROXY, you told GemFire to cache any people that you create from this client. However, you can also choose to receive any updates to the people region that occur on other peers or other clients by invoking the registerInterest methods on the client. In this case you want to register interest in all people, so you cache the entire people region on the client. The regular expression ".\*" matches all keys in the people region. Look at the initClient method. The next line calls registerInterestRegex:  people.registerInterestRegex(".\*");  When the registerInterestRegex method is invoked, the client downloads all existing people from the server. When a new person is added on the server it is pushed to the client.   1. Iterate over keys from the client.   Look at the getPeople and getPosts methods in GemFireDAO.  public Set<String> getPeople() { return people.keySet(); } public Set<PostID> getPosts() { if(isClient) { return posts.keySetOnServer(); } else { return posts.keySet(); } }  GemFireDAO.getPeople calls people.keySet(), whereas GemFireDAO.getPosts calls posts.keySetOnServer() for the client. That's because the keySet method returns the keys that are stored locally on the client. For the People region you can use your locally cached copy of the keys, but for the Post region you need to go to the server to get the list of keys.   1. Open a new terminal window and start the client application.   $ java com.gemstone.gemfire.tutorial.Client  You should be able to add people and posts from the client. You can start another client and see that people are sent from one client to the other. |  |