Gemfire for Developers

Native Client Lab Instructions

Your guide to completing the hands-on labs

V8.2a - April 2016

Pivotal

Copyright Notice

2016

Lab 1: Installing the Gemfire Native Client Labs Environment

Introduction

Welcome to the GemFire-Developer Native Client course. This is a supplementary course to the main Gemfire for Developers training. We have tried to make the lab setup as simple as possible. All you need to fulfill the Labs is to have set up a compatible system to build the source code provided alongside with these instructions.

The current course version includes C++ native client examples and supports Linux OS only. You would need to have a running Linux VM (e.g. CentOS or RHEL 6.x) and have g++ compiler installed and available for your user in the OS.

Please ask your instructor in case you are encountering issues with the courseware.

What you will learn:

- Setting up Gemfire C++ Native Client build environment
- Compile and link native Linux executable with the Gemfire native C++ client library

Prerequisites:

- Linux virtual machine (CentOS or RHEL 6.x+ is recommended)
- GNU gcc compiler version 4.4.7 or higher installed or network connection for automatic installation
- OS root access rights for the

Estimated completion time: 30 minutes

Instructions

Instructions for this lab are divided into specific sections. Each section describes the steps to perform specific tasks.

1.1 Installing gcc prerequisites

a. Using your OS package management system install the support libraries in standard system locations. For Debian-based systems, including Ubuntu, you should install the packages `libgmp-dev`, `libmpfr-dev` and `libmpc-dev`. For RPM-based systems, including Fedora and SUSE, you should install `gmp-devel`, `mpfr-devel` and `libmpc-devel` (or `mpc-devel` on SUSE) packages. The packages will install the libraries and headers in standard system directories so they can be found automatically when building GCC. We will be using RedHat RPM package system in our examples. Install the gcc and g++ compilers with the following command

```
sudo yum install gcc gcc-g++
```

b. Check that the g++ compiler is installed, available and has correct version

```
gcc --version

     gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-16)
     Copyright (C) 2010 Free Software Foundation, Inc.
     This is free software; see the source for copying conditions.
     There is NO
     warranty; not even for MERCHANTABILITY or FITNESS FOR A
     PARTICULAR PURPOSE.

g++ --version

     g++ (GCC) 4.4.7 20120313 (Red Hat 4.4.7-16)
     Copyright (C) 2010 Free Software Foundation, Inc.
     This is free software; see the source for copying conditions.
     There is NO
     warranty; not even for MERCHANTABILITY or FITNESS FOR A
     PARTICULAR PURPOSE.
```

1.2 Setting up labs sources

The sources for this course are distributed as a simple gzip archive `gemfire-dev-labs-native-0.1.tar.gz` It contains this manual and the lab source code in the subfolder `gemfire-dev-labs-native`. This folder has a subfolder for each lab described in this document with the source code that you need to complete to fulfill the lab and a second

folder with the name ending with `-solution` that contains a working lab solution for your reference.

a. Copy the `gemfire-dev-labs-native` folder to the Linux VM that you will be using for your labs. For example if the host OS file system is mounted in the Linux VM using VMware Fusion HGFS feature

```
cd ~
cp /mnt/hgfs/sshcherbakov/gemfire-dev-labs-native-0.1.tar.gz ~
```

b. Extract archive to the current folder

```
tar xzf gemfire-dev-labs-native-0.1.tar.gz
cd gemfire-dev-labs-native
```

1.3 Setting up Gemfire Native Client libraries

In order to build the native Gemfire client applications as described in this document, we need install the Gemfire Native Client in the Linux VM as described in the “Installing Native Client” section of the “Pivotal Gemfire User’s Guide” document. This document is included to the “Gemfire Developer Native Client Labs” package as `pivotal-gemfire-ug.pdf`. The most actual version of the “Pivotal Gemfire User’s Guide” can also be accessed online on the official Pivotal Gemfire reference documentation page.

a. Follow instructions in the “Installing Native Client” in the `pivotal-gemfire-ug.pdf` on how to download the correct version of the Pivotal Gemfire Native Client package to the Linux VM
b. Set the environment parameters needed to build the labs sources in the Linux VM. In this instructions we will assume that the Pivotal Gemfire Native Client is installed to the `~/NativeClient_Linux_64bit_8202_b3768` subfolder in the Linux VM. In the command line shell that we’ll be using for the Gemfire Native Client Tests enter the following commands

```
export GFCPP=~/NativeClient_Linux_64bit_8202_b3768
export PATH=$GFCPP/bin:$PATH
export LD_LIBRARY_PATH=$GFCPP/lib:$LD_LIBRARY_PATH
```

1.4 Setting up the test Gemfire cluster

We will be reusing the Gemfire installation and environment that you have set up in the main “Gemfire Developer Course”. Please refer to the section 1 of the “Gemfire Developer Course” handout document for details of this installation.

In this course we assume that the Linux VM where we will be building native C++ executables has at least network access to the Gemfire cluster set up in the main Gemfire Developer Course. This can be easy achieved if you run the Linux VM inside of the Windows host OS where you were completing the main Gemfire Developer Course.

Please make sure that no firewall blocks access between Linux VM and the host OS.

Alternatively, you can install and start Gemfire cluster as is described here in the Linux VM itself.

After finishing this section you should be prepared now to continue with the following coding tasks.

Lab 2: Basic Gemfire API operations

Introduction

In this lab you will build the first native C++ application that can access data stored in the Gemfire cluster using basic Gemfire Native Client API.

What you will learn:

- Compile and link native Linux executable with the Pivotal Gemfire native C++ client library
- Configure client cache with a cache XML file
- Store and access the data in the remote Gemfire cluster using basic Gemfire C++ client API

Estimated completion time: 30 minutes

Instructions

2.1 Storing and accessing Gemfire data from a client application

a. In the Linux VM shell set in the previous Lab switch the lab folder

```
cd ~/gemfire-dev-labs-native/lab2-basic
```

b. In you favorite editor open and investigate the `Basic.cpp` and `clientCache.xml` files located in the folder. The `Basic.cpp` contains incomplete code of a simple command line C++ application that will be accessing remote Gemfire cluster running in the host Windows OS. The application will access Gemfire cluster using native Gemfire client cache configuration from the `clientCache.xml` file in the same XML format that is used by the Java Gemfire client applications. E.g. you can view the files directly in the command line

```
less Basic.cpp
less clientCache.xml
```

Each file will contain comments starting with “TODO”. These comments indicate places, where you need you modify code to complete the lab.

c. `(TODO-0)` In the `clientCache.xml` modify the IP address of the Gemfire locator that will be used by the native client cache. Replace the IP address with the IP address of the machine where your test Gemfire cluster is running, e.g. your host Windows OS

d. `(TODO-1)` In the `Basic.cpp` include the `<gfcpp/GemfireCppCache.hpp>` file to enable access to the Gemfire API in the code

e. `(TODO-2)` The Pivotal Gemfire native API calls are part of the `gemfire` C++ namespace. In order not to enter this namespace in each call to the Gemfire API function add `gemfire` namespace to the visibility scope in the `Basic.cpp`

f. `(TODO-3)` Similarly to the Java client, the first thing we have to do to start working with the Gemfire cache is obtaining the instance of the `CacheFactoryPtr` object. Use the static `createCacheFactory()` method of the `gemfire::CacheFactory` class for that. Note, that you do not need to explicitly specify the `gemfire` namespace here after you have executed the step **e)**

g. `(TODO-4)` Use the `CacheFactoryPtr` object to configure it to use the external client cache configuration from the `clientCache.xml` file and create a client cache represented by the `CachePtr` object using `create()` method

h. `(TODO-5)` You can now get the instance of the cache `RegionPtr` object, which you can use to access the `"Customer"` Gemfire region. Use `getRegion(…)` method of the `CachePtr` class

i. `(TODO-6)` Use the `RegionPtr` object now to put simple string key and value to the `"Customer"` region. Use the shortcut `put()` method and pass C++ string literal to it directly

j. `(TODO-7)` You will create the second entry in the `"Customer"` region with the integer key and value types. Use the `CacheableInt32` class to and its static `create(…)` method to create `CacheableKeyPtr` and `CacheablePtr` objects for the entry key and value that are then passed to the `regionPtr->put()` method

k. `(TODO-8)` Use the `regionPtr->get(…)` to retrieve from the Gemfire cache the value of the keys stored in the "Customer" region at the steps **i)** and **j)**

l. `(TODO-9)` At the end of the program close the Gemfire client cache gracefully using `close()` method of the `CachePtr` class

2.2 Building the client application sources

a. In the Linux VM shell that was set in the previous Lab switch to the lab folder

```
cd ~/gemfire-dev-labs-native/lab2-basic
```

b. Build the `Basic.cpp` into a native Linux executable. If you set the Linux VM environment as described in the Lab1, the following command should execute without errors and generate executable file `Basic` in the current folder

```
g++ -D_REENTRANT -O3 -Wall -m64 -I${GFCPP}/include -L${GFCPP}/lib -Wl,\
     -rpath,${GFCPP}/lib -lgfcppcache Basic.cpp -o Basic
```

The `-D-REENTRANT -O3 -Wall -m64 -Wl,-rpath` command line switches are not necessarily required to perform the build but are recommended in the Pivotal Gemfire User's Guide.

Note

If you encounter multiple compilation errors at this step verify whether the GFCPP environment variable is set to the `~/NativeClient_Linux_64bit_8202_b3768` folder in the shell:

```
echo ${GFCPP}
```

2.3 Starting up the Gemfire server cluster

Follow the next steps if you don't have yet Gemfire cluster in the host Windows OS up and running.

c. In the Windows host VM either open a command windows or use the open the one that you already have. This will create a fresh folder to store runtime data for the test Gemfire cluster processes

```
mkdir lab2-basic
cd lab2-basic
```

d. Similar to the main "Gemfire Developer Course" labs we will be using gfsh console to start the Gemfire cluster, let's start it

```
gfsh
```

e. In the `gfsh` console start a Gemfire locator and two server processes. The locator will run by default on port 10334. Servers will get ports assigned automatically.

```
start locator --name=locator1 --initial-heap=50m --max-heap=50m
start server --name=server1 --server-port=0 --initial-heap=50m --max-
      heap=50m
start server --name=server2 --server-port=0 --initial-heap=50m --max-
      heap=50m
```

f. Create a server-side region that our native client application will connect to

```
create region --name=Customer --type=REPLICATE
```

g. Verify that the region has been successfully created

```
list regions
describe region --name=/Customer
```

We should see the following output

```
gfsh>describe region --name=/Customer
..........................................................
Name            : Customer
Data Policy     : replicate
Hosting Members : server2
                  server1

Non-Default Attributes Shared By Hosting Members

 Type   | Name | Value
------ | ---- | -----
Region | size | 0
```

2.4 Running the lab example

a. In the Linux VM shell start the executable that had been built on step 2.1 d)

```
./Basic
```

In the console you should see the Gemfire client library log output as well as the log output from our example application. Last lines should look similar to the following.
As can be seen here the client application successfully starts, connects to the Gemfire cluster and puts sample entries to the server cache.

```
[config 2016/04/16 22:49:58.939887 CEST vfabric-0:2664 139997397157664] Starting the
GemFire Native Client
[info 2016/04/16 22:49:58.940714 CEST vfabric-0:2664 139997397157664] Using
GFNative_aT9YsXM4da2664 as random data for ClientProxyMembershipID
[info 2016/04/16 22:49:58.963455 CEST vfabric-0:2664 139997397157664] Xml file parsed
successfully
ACE_INET_Addr::ACE_INET_Addr[info 2016/04/16 22:49:58.979873 CEST vfabric-0:2664
139997397157664] Creating region Customer attached to pool examplePool
[info 2016/04/16 22:49:58.979944 CEST vfabric-0:2664 139997235623680]
ClientMetadataService started for pool examplePool
[info 2016/04/16 22:49:58.980451 CEST vfabric-0:2664 139997397157664] Declarative
configuration of cache completed successfully
[info 2016/04/16 22:49:58.980476 CEST vfabric-0:2664 139997397157664] Created the GemFire
Cache
[info 2016/04/16 22:49:58.980483 CEST vfabric-0:2664 139997397157664] Created Region
[info 2016/04/16 22:49:58.985439 CEST vfabric-0:2664 139997235623680] Using socket send
buffer size of 64240.
[info 2016/04/16 22:49:58.985460 CEST vfabric-0:2664 139997235623680] Using socket receive
buffer size of 64240.
[info 2016/04/16 22:49:59.127930 CEST vfabric-0:2664 139997397157664] Put the first Entry
into the Region
[info 2016/04/16 22:49:59.132689 CEST vfabric-0:2664 139997397157664] Put the second Entry
into the Region
[info 2016/04/16 22:49:59.133795 CEST vfabric-0:2664 139997397157664] Obtained the first
Entry from the Region
[info 2016/04/16 22:49:59.134829 CEST vfabric-0:2664 139997397157664] Obtained the second
Entry from the Region
[info 2016/04/16 22:49:59.135397 CEST vfabric-0:2664 139997235623680]
ClientMetadataService stopped for pool examplePool
[config 2016/04/16 22:49:59.196926 CEST vfabric-0:2664 139997397157664] Stopped the
GemFire Native Client
[info 2016/04/16 22:49:59.196956 CEST vfabric-0:2664 139997397157664] Closed the GemFire
Cache
```

b. Let's check in the Gemfire gfsh console whether the two sample entries created by the sample app are really stored in the Gemfire server cache now. In the Windows host OS gfsh console where we started Gemfire cluster at step 2.2 enter the following commands

```
query --query="select * from /Customer"
```

```
gfsh>query --query="select * from /Customer"

Result     : true
startCount : 0
endCount   : 20
Rows       : 2

Result
------
Value1
123

NEXT_STEP_NAME : END
```

As you can see here, there are two entries in the `/Customer` region and they have values as created by our native client application.
Let's check the type of those entries:

```
get --region=/Customer --key="Key1"
```

```
gfsh>get --region=/Customer --key="Key1"
Result      : true
Key Class   : java.lang.String
Key         : Key1
Value Class : java.lang.String
Value       : Value1
```

The C++ string literals are being stored as Java Strings in the cache. If we query for the second key “123” we will find no value available in the cache.

```
get --region=/Customer --key="123"
```

```
gfsh>get --region=/Customer --key="123"
Result      : false
Key Class   : java.lang.String
Key         : 123
Value Class : java.lang.String
Value       : <NULL>
```

This is because the type of the key that we have used when storing the “123” entry was `CacheableInt32` integer. We need to specify the key type in the gfsh to obtain the key’s value:

```
get --region=/Customer --key="123" --key-class=java.lang.Integer
```

```
gfsh>get --region=/Customer --key="123" --key-class=java.lang.Integer
Result      : true
Key Class   : java.lang.Integer
Key         : 123
Value Class : java.lang.Integer
Value       : 123
```

The key and value specified as `CacheableInt32` integer in the C++ client have been stored as Java primitive integer type `java.lang.Integer`.

Congratulations! You have completed this lab.

Lab 3: Cross-platform communication using PDX

Introduction

In this lab you will build a native C++ application that can access data stored in the Gemfire cluster by a Java application. Your application will connect to the same Gemfire cluster that you were using in the main "Gemfire Developer Course". You will also need to complete the "Data Serialization" lab in the main "Gemfire Developer Course" since we will rely on the data stored in the Gemfire in this lab.

What you will learn:

- Use PDX serialization mechanism in native C++ client applications
- Working with multiple versions of domain objects in PDX
- Using PdxInstance to get a single field

Estimated completion time: 40 minutes

Instructions

3.1 Using Gemfire PDX serialization

a. In the Linux VM shell switch the lab folder

```
cd ~/gemfire-dev-labs-native/lab3-cross-platform
```

b. In your favorite text editor open and investigate the `PdxInstance.cpp` and `clientPdxInstance.xml` files. The `PdxInstance.cpp` is a simple console application that acts as Gemfire client and stores PDX serialized data in the remote Gemfire distributed system. The `clientPdxInstance.xml` is a Gemfire cache XML file used by the PdxInstance.cpp application to configure Gemfire client cache instance. You will need to fill-in the places marked with `"TODO-"` in the both files to complete this lab. Please refer to the `~/gemfire-dev-labs-native/lab3-cross-platform-solution` for the full working version of the lab application.

c. `(TODO-0)` In the `clientPdxInstance.xml` modify the IP address of the Gemfire locator that will be used by the native client cache. Replace the IP address with

the IP address of the machine where your test Gemfire cluster is running, e.g. your host Windows OS

d. `(TODO-1)` In the `clientPdxInstance.xml` add the `<pdx>` element and turn on the `read-serializable` flag. This is tell the native client cache to obtain the entries as PdxInstance objects and not try to deserialize them automatically when returning to the client application code

e. `(TODO-2)` Create an instance of the Person class to be stored in the Gemfire cache using PDX serialization. It will be used as a source of the PdxInstance object fields and can be placed directly on the `main()` procedure stack to simplify memory management. Please note that the Person class is defined and implemented directly at the top of the main `PdxInstance.cpp` application source file.

f. `(TODO-3)` Create `PdxInstanceFactoryPtr` object for the `"Person"` class. You should use for that the `createPdxInstanceFactory()` method of the already initialized `CachePtr` object and pass the fully qualified name of the `PdxInstance` object when becomes deserialized. It is not important in this part of the exercise; therefore you can simply provide `"Person"` as an argument. You will use the `PdxInstanceFactoryPtr` object to serialize Person's object fields one by one.

g. `(TODO-4)` Wrtie Person object's field `m_name` to the `PdxInstanceFactoryPtr` as String using the `writeString()` method.

h. `(TODO-5)` Wrtie Person object's field `m_id` to the `PdxInstanceFactoryPtr` as Integer using the `writeInt()` method.

i. `(TODO-6)` You will need to mark at least one `Person` class field as identity field for the Gemfire PDX mechanism to differentiate objects correctly. You should pick the field that uniquely identifies the object. Mark the `m_id` field of the `Person` class as PDX instance identity field using the `markIdentityField()` method.

j. `(TODO-7)` Now you are ready to create the `PdxInstancePtr` object from the `PdxInstanceFactoryPtr`. Use its instance method `create()` to instantiate the `PdxInstance` object that is now ready to be stored in the Gemfire.

k. `(TODO-8)` You can use the normal Gemfire API calls to put the `PdxInstancePtr` object to the "`Customer`" region under key `Key3` as is. The Gemfire will automatically recognize the object type and will store it on the server side in the PDX form ready for clients on other platforms to access its data.

l. `(TODO-9)` We will now read the stored PdxInstance object back from the Gemfire server and compare retrieved values with the ones that we have used at object creation. Add the code to read the value stored under the key `Key3` from the

`Customer` region as `PdxInstancePtr` object. Use the already defined `regionPtr` object for that.

m. `(TODO-10)` After you have retrieved the `PdxInstancePtr` corresponding to the value that was stored at previous step **k)** read the `m_id`, `m_age` and `m_name` fields from the `PdxInstancePtr` object into the `id`, `age` and `name` variables defined at this place. The following code performs conditional check to verify that the retrieved field values are equal to the test values that were used during initial `PdxInstance` object creation. It must output success message to the application's standard out log stream when you build and run the test application at the step **3.3 c)**

3.2 Store PDX data from the Java client application

a. You are now going to access values in the Gemfire cache that were stored by an application running on another platform. That will be a sample Java application from the "Data Serialization" lab of the main "Gemfire Developer Course" labs (Chapter 11). We assume that you have successfully set up the main "Gemfire Developer Course" environment on your laptop and have successfully executed the `com.gopivotal.bookshop.buslogic.NewCustomerClient` command line Java application from the `data-serialization` or `data-serilization-solution` project. When run without any parameters this application creates under the key "9999" in the "Customer" region a PDX serialized value and stores an instance of the `com.gopivotal.bookshop.domain.Customer` Java class in the Gemfire cache.

b. Check that the Java object has been successfully stored in the "Customer" region by executing the gfsh command

```
get --region=/Customer --key="9999" --key-clas=java.lang.Integer
```

You should see the similar output in the console:

```
gfsh>get --region=/Customer --key="9999" --key-class=java.lang.Integer
Result      : true
Key Class   : java.lang.Integer
Key         : 9999
Value Class : com.gemstone.gemfire.pdx.internal.PdxInstanceImpl

customerNumber | firstName | lastName | telephoneNumber |
primaryAddress                        | myBookOrders
-------------- | --------- | -------- | --------------- | --------------------------
---------------------------- | -------------------------
9999           | Modified  | Customer | 5035551212      | class
com.gemstone.gemfire.pdx.internal.PdxInstanceImpl | class java.util.ArrayList
```

Note

Please note that the above command can only show the contents of the stored PDX object in case when 2 pre-requisites are fulfilled in course of “Data Serialization” lab execution:

a. Gemfire server processes are started with the data-serialization lab jar file added to the classpath

b. Appropriate PDX serializer has been added to the Gemfire server cache XML on startup, e.g.

```
<?xml version="1.0" encoding="UTF-8"?>
<cache
        xmlns="http://schema.pivotal.io/gemfire/cache"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://schema.pivotal.io/gemfire/cache
http://schema.pivotal.io/gemfire/cache/cache-8.1.xsd"
        version="8.1">
    <pdx read-serialized="true" >
        <pdx-serializer>
            <class-
name>com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer</class-name>
            <parameter name="classes">
                <string>com.gopivotal.bookshop.domain.*</string>
            </parameter>
        </pdx-serializer>
    </pdx>
        <region name="Customer">
         <region-attributes refid="REPLICATE">
        </region-attributes>
       </region>

</cache>
```

The sample Gemfire server start-up command can look like the following in this case:

```
start server --name=server1 --server-port=0 --initial-heap=50m --max-heap=50m --
    classpath=/path-to/data-serialization-solution/target/data-serialization-
    solution-1.0.0.CI-SNAPSHOT.jar --cache-xml-file=/ path-to /data-serialization-
```

3.3 Reading PDX values stored by Java application

a. `(TODO-11)` You will now continue implementation of the `PdxInstance.cpp` application by reading the `PdxInstancePtr` object stored under key 9999 by the Java application from the main Gemfire Developer Course "Data Serialization" lab. Get that object using the available `regionPtr` object.

b. (TODO-12) In the similar way as you did in the step **3.1 m)** read the `PdxInstance` object fields `customerNumber`, `firstName`, `lastName`, `telephoneNumber` into the pre-defined local variables

c. You are now ready to build the lab test C++ Gemfire client application. Build it in the `~/gemfire-dev-labs-native/lab3-cross-platform` folder similar to the `Basic` application in the Lab2:

```
g++ -D_REENTRANT -O3 -Wall -m64 -I${GFCPP}/include -L${GFCPP}/lib -Wl,\
    -rpath,${GFCPP}/lib -lgfcppcache PdxInstance.cpp -o PdxInstance
```

d. The build should create a native executable file `PdxInstance` in the current folder and generate no warnings or errors.

e. (TODO-13) Now run the `PdxInstance` application

```
./PdxInstance
```

And observe its console output. You should see no errors and all log output messages that were added to the application code.
Please note that the values of the Customer object with the key “9999” stored by the test Java application are getting also shown correctly and correspond to the values set in the Java application.
The console output should look similar to the following

```
[config 2016/04/18 07:25:36.199293 CEST vfabric-0:3446 139964125525792] Starting the GemFire Native Client
[info 2016/04/18 07:25:36.199874 CEST vfabric-0:3446 139964125525792] Using GFNative_sDB4W4BZom3446 as random
data for ClientProxyMembershipID
[info 2016/04/18 07:25:36.201300 CEST vfabric-0:3446 139964125525792] Xml file parsed successfully
ACE_INET_Addr::ACE_INET_Addr[info 2016/04/18 07:25:37.209358 CEST vfabric-0:3446 139964125525792] Creating
region Customer attached to pool examplePool
[info 2016/04/18 07:25:37.209522 CEST vfabric-0:3446 139963960116992] ClientMetadataService started for pool
examplePool
[info 2016/04/18 07:25:37.209812 CEST vfabric-0:3446 139964125525792] Declarative configuration of cache
completed successfully
[info 2016/04/18 07:25:37.209850 CEST vfabric-0:3446 139964125525792] Created the GemFire Cache
[info 2016/04/18 07:25:37.209860 CEST vfabric-0:3446 139964125525792] Obtained the Region from the Cache
[info 2016/04/18 07:25:37.209866 CEST vfabric-0:3446 139964125525792] ****************** WORKING WITH PDX
******************
[info 2016/04/18 07:25:37.209879 CEST vfabric-0:3446 139964125525792] Created PdxInstanceFactory for Person
class
[info 2016/04/18 07:25:37.209973 CEST vfabric-0:3446 139963960116992] Using socket send buffer size of 64240.
[info 2016/04/18 07:25:37.209992 CEST vfabric-0:3446 139963960116992] Using socket receive buffer size of 64240.
[info 2016/04/18 07:25:37.232473 CEST vfabric-0:3446 139964125525792] Created PdxInstance for Person class
[info 2016/04/18 07:25:37.236264 CEST vfabric-0:3446 139964125525792] Populated PdxInstance Object
[info 2016/04/18 07:25:37.237857 CEST vfabric-0:3446 139964125525792] Got PdxInstance Object
[info 2016/04/18 07:25:37.237886 CEST vfabric-0:3446 139964125525792] PdxInstance returns all fields value
expected
[info 2016/04/18 07:25:37.237892 CEST vfabric-0:3446 139964125525792] ****************** READING JAVA VALUES
******************
[info 2016/04/18 07:25:37.239259 CEST vfabric-0:3446 139964125525792] Read Customer: customerNumber=956301351;
firstName=Modified; lastName=Customer; telephoneNumber=5035551212
[info 2016/04/18 07:25:37.239939 CEST vfabric-0:3446 139963960116992] ClientMetadataService stopped for pool
examplePool
[config 2016/04/18 07:25:37.363919 CEST vfabric-0:3446 139964125525792] Stopped the GemFire Native Client
[info 2016/04/18 07:25:37.363961 CEST vfabric-0:3446 139964125525792] Closed the GemFire Cache
```

Congratulations! You have now successfully completed this lab.

Appendix A: Notes

The Pivotal Gemfire Native Client package included into these lab materials contains a `quickstart/SampleCode/cpp` subfolder with more C++ client application code examples that illustrate Gemfire API usage from a native C++ application. Please refer to those code examples for more details.

The latest native client package version is available on the Pivotal product download site:

https://network.pivotal.io/products/pivotal-gemfire

You can find the native C++ client API reference in the official Pivotal Gemfire online reference documentation:

http://data-docs-samples.cfapps.io/docs-gemfire/latest/cpp_api/cppdocs/index.html

More information on building the C++ Gemfire Native Client application you can find in the official Gemfire product documentation:

http://gemfire.docs.pivotal.io/docs-gemfire/gemfire_nativeclient/cpp-caching-api/cpp-caching-api.html