ORACLE
FUSION MIDDLEWARE
COHERENCE

An Oracle White Paper
February 2014

Oracle Coherence GoldenGate HotCache Tutorial

Introduction

Third-party updates to the database can cause Oracle Coherence applications to work with data which could be stale and out-of-date. Oracle Coherence GoldenGate HotCache solves this problem by monitoring the database and pushing any changes into the Coherence cache.

HotCache employs an efficient push model which processes only stale data. Low latency is assured because the data is pushed when the change occurs in the database.

HotCache can be added to any Coherence application. Standard JPA is used to capture the mappings from database data to Java objects. The configuration can be captured in XML exclusively, or in XML with annotations.

You can view this tutorial as an aggregation of Oracle GoldenGate and Coherence documentation which specifically deals with GoldenGate setup and HotCache setup.

This tutorial provides a step-by-step guide to the new HotCache feature introduced in Coherence 12*c*. The supported GoldenGate releases are 11GR1 and 11gR2.

This document assumes that you have:

- The full installation of Oracle WebLogic Server 12c (which includes Coherence).
- Downloaded and installed GoldenGate 11gR1 or 11gR2 (just unzip).
- Downloaded and installed GoldenGate Java adapter (just unzip).
- An IDE, such as the Eclipse IDE (OEPE), or you can use your preferred IDE.
- A working knowledge of GoldenGate.
- A working knowledge of Coherence.
- Familiarity with Eclipse.
- Familiarity with JPA.

Note:

- For convenience, an IDE is used in this tutorial.

1. Understanding GoldenGate

Before you setup GoldenGate, and just in case you have no prior experience with GoldenGate, here is a brief introduction to its architecture as it pertains to HotCache.

GoldenGate consists of several processes, each of which is responsible for performing a distinct function:

- The Manager process: this is the umbrella process that must be up and running for any other GoldenGate process to run. The Manager can be configured to automatically restart a failed process. The Manager can be configured to automatically purge a trail file once all dependent processes are finished with it.
- The Primary Extract process: this process captures real time, committed transactions from the transaction log, and in the case of Oracle Database, from the online Redo log. It can also operate in bulk mode where it will perform a `SELECT *` from a table for an initial load of a target table, although this is not the configuration used for HotCache. The Primary Extract can push data directly to a target/remote trail file, but in the case of configuring HotCache, you setup the Primary Extract to write to a local trail file.
- The trail file: a binary file which stores the committed transactions in a canonical format which can be read by GoldenGate processes.
- The User Exit Extract process: reads from the trail file, calls the Java Adapter, implements the attributes in the property files, produces an XML representation of the database change, and passes it to the Java adapter (HotCache in this case).

Note:

- As stated earlier, the GoldenGate Manager process controls all GoldenGate processes mentioned earlier and it can be configured to start other needed GodenGate processes..

Other than replicating data from one database to another, GoldenGate offers out-of-the-box "handlers" such as JMS handlers (or User Exits) where any changes to the database table are sent as a JMS message to a pre-defined JMS provider or written to a file.

HotCache takes advantage of the GoldenGate event handlers.

These custom handlers operate as "User Exits". User Exits are called by the User Exit Extract process. Every-time the GoldenGate Extract process detects a change, it formulates the change into an XML document that the "User Exit" can use to further process the change as needed.

For example, assume you have a table called "Accounts". When an insert operation is detected, the Extract process generates the following XML representing the newly added record:

```
<operation table="ACCOUNTS" type="INSERT" numCols="5">
 <col name="CUSTOMERID" index="0">
   <before missing="true"/>
   <after>3</after>
 </col>
 <col name="ACCOUTNID" index="1">
   <before missing="true"/>
   <after>3</after>
 </col>
 <col name="NAME" index="2">
   <before missing="true"/>
   <after>Bassam Hijazi</after>
 </col>
  <col name="BALANCE" index="4">
   <before missing="true"/>
   <after>4800</after>
 </col>
</operation>
```

Figure 1. XML for the Accounts Record

As you can see from the XML representation, you can easily find the table name, the operation type (insert, update, delete) and number of columns. Also, if this was an "update" operation and GoldenGate was configured to send the "before" values, you would get the "before" and "after" values.

The above XML format is the default format of GoldenGate (at the time of this writing) and it uses the "Velocity Macro" for its XML formatting. It does, however, allow you to develop your own custom formatter which the extractor process can use to represent the database operation in XML.

As you can see, GoldenGate is quite flexible in that it offers a lot of options to customize the operation of the Extract process from having custom Java handlers and custom formatters, to having its own set of out-of-the-box handlers.

This completes the review of GoldenGate architecture. Let's move on to setting it up.

2. GoldenGate Initial Setup

After GoldenGate is installed, you must perform some initial setup before it is ready to run. This document will refer to the directory where GoldenGate was installed as `$GG_HOME`.

GoldenGate offers a command line tool called "GoldenGate Software Command Interpreter" or `ggsci`, which is available under `$GG_HOME`.

The initial setup involves creating all of the directories that GoldenGate manager needs along with defining the TCP port GoldenGate manager will use. So, you first create the needed directories, then you define the TCP port that the manager will be using, and finally start GoldenGate Manager.

Step 1. Start GoldenGate

To start the initial setup of GoldenGate, navigate to the `$GG_HOME` and invoke `ggsci`:

```
 ./ggsci
Oracle GoldenGate Command Interpreter for Oracle
Version 11.2.1.0.1 OGGCORE_11.2.1.0.1_PLATFORMS_120423.0230_FBO
Linux, x64, 64bit (optimized), Oracle 11g on Apr 23 2012 08:32:14
Copyright (C) 1995, 2012, Oracle and/or its affiliates. All rights
reserved.
```

Step 2. Create the Directory Structure

The following command creates all of the subdirectories under `$GG_HOME` that GoldenGate manager and other processes require:

```
GGSCI (bhijazi-linux) 1>CREATE SUBDIRS
```

Step 3. Specify the TCP Port

The following command opens an editor ("vi" if you are using Unix or Linux) where you can define the TCP port that OGG manager will use.

```
GGSCI (bhijazi-linux) 1>EDIT PARAMS MGR
```

Simply enter `PORT` *tcp_port_number* in the file. For example:

```
PORT 7809
```

Save the file using the Save command of the editor you are using, which will bring you back to the command prompt.

Step 4. Start the Manager Process

Enter the `start mgr` command to start the manager:

```
GGSCI (bhijazi-linux) 1> START MGR
```

Step 5. Check the Status of the Manager Process

To check the status of the manager, enter the `status mgr` command:

```
GGSCI (bhijazi-linux) 2> status mgr
Manager is running (IP port bhijazi-linux.7809)
```

Exit the GoldenGate command line tool with the `exit` command:

```
GGSCI (bhijazi-linux) 4>exit
```

Now that you have finished GoldenGate initial setup, you are ready to setup the database. This document assumes that you have already installed a version of Oracle Database which is supported by GoldenGate.

3. Preparing Oracle Database for GoldenGate

Before GoldenGate can start capturing changes to an Oracle Database table or tables, you must prepare the database for GoldenGate. You might need the SYS user password to perform all of the steps listed below. Use SQLPlus or whatever tool you prefer to access the database. Login to the database as the SYS user.

Step 1. Enable Archive Mode

Check to see if your database is in archive mode. When the database is in archive mode, it will create copies of the redo log files. Although the Extraction process reads changes from the transaction logs, there are circumstances (such as a gap in the redo) where GoldenGate will get the transactions from the archived log files.

To check, issue the following statement:

```
SQL>select log_mode from v$database;
```

If archive mode is enabled, you should get something like this:

```
LOG_MODE
------------
ARCHIVELOG
```

If the command states that your database is not in archive mode, then enable it by issuing the following SQL statements:

```
SQL>shutdown immediate;
SQL>startup mount;
SQL>Alter database archivelog;
SQL>Alter database open;
```

Step 2. Enable Supplemental Logging

Ensure that your database supports supplemental logging. You need supplemental logging because Oracle Database usually uses the Redo logs to store the ROWIDs of the row(s) that have been changed along with the new values for the column(s).

When dealing with two databases, the ROWIDs will be different. Therefore, implementing supplemental logging instructs Oracle Database to include the primary key, unique keys, or all columns of the tables (some exceptions apply) in the redo logs as well.

To check if you are using supplemental logging, issue the following SQL statement:

```
SQL>select supplemental_log_data_min from  v$database;
```

If the result is Yes, then you are ready to move to the next step. If it is not Yes, then issue the following statement to enable supplemental logging:

```
SQL>alter database add supplemental log data;
```

Now that you have the database prerequisite for GoldenGate configured, it is time to create a special user that GoldenGate will use to manage the extraction of transactional data.

Step 3. Create a User and Grant Privileges

Create a user called `ggowner` and password `ggowner`. After creating the user, you must grant the user the following set of privileges:

```
SQL>create user ggowner identified by ggowner;
SQL>grant connect to ggowner;
SQL>grant dba to ggowner;
SQL>grant create session to ggowner;
SQL>grant alter session to ggowner;
SQL>grant select any dictionary to ggowner;
SQL>grant select any table to ggowner;
SQL>grant select any transaction to ggowner;
SQL>grant unlimited tablespace to ggowner;
```

Note:

- You may feel that granting "Any" to the GoldenGate user is too aggressive, but you can always replace the "Any" with a more specific grant. For example, you can replace "`grant select any table to ggowner`" with "`grant select scott.employee to ggowner`"

Step 4. Login to the Database from GoldenGate

Logout of your SQL session. Now, try to login to the database using the user you just created, `ggowner`, using the GoldenGate command line:

```
$GG_HOME/./ggsci
GGSCI (bhijazi-linux) 1>dblogin userid ggowner, password ggowner
Successfully logged into database
Stop GG Manager and exit the shell:
GGSCI (bhijazi-linux) 3> stop mgr
GGSCI (bhijazi-linux) 4>exit
```

Step 5. Create a Database Table

For the purpose of this tutorial, create a sample table called `Customers` with the following columns:

- `CustID integer`
- `Firstname varchar2`

- Lastname varchar2
- Telephone varchar2

The Customers table defines CustID as the primary key.

Use your preferred SQL editor to create the table:

```
SQL> CREATE TABLE "SCOTT"."CUSTOMERS" ("CUSTID" INTEGER NOT NULL,
"FIRSTNAME" VARCHAR2(20) NOT NULL, "LASTNAME" VARCHAR2(20) NOT NULL,
"TELEPHONE" VARCHAR2(15), PRIMARY KEY ("CUSTID") VALIDATE);
```

Enable supplemental logging to the table(s) from which you want to capture committed transactions. To do this, use the GoldenGate command line interface:

```
$GG_HOME/./ggsci
GGSCI (bhijazi-linux) 1>dblogin userid ggowner, password ggowner
GGSCI (bhijazi-linux) 2>add trandata SCOTT.CUSTOMERS
```

Note:

- If you are monitoring all the tables under one schema, then you can issue:
  GGSCI (bhijazi-linux) 2>add trandata *schema_name*.*

Step 6. Create Table Metadata for the Extract Process

The Extract process requires metadata describing the trail data. This metadata can come from a database or a Source Definition file which describes the table being monitored. The metadata used in this example defines the column names and data types in the trail being read. To create the file, enter the following command:

```
GGSCI (bhijazi-linux) 3>edit params defgen
```

This command will open a file editor. Enter the following text in the file:

```
USERID ggowner, password ggowner
defsfile dirdef/customerdef.def
TABLE SCOTT.CUSTOMERS;
```

Save the file. This will create a file called defgen.prm in the $GG_HOME/dirprm directory.

The text in the file instructs GoldenGate to create a definition file named customerdef.def for the table scott.customers in the dirdef directory.

```
GGSCI (bhijazi-linux) 4>exit
```

Step 7. Generate the Table Definition File

Next, invoke the defgen command, which is available under the $GG_HOME, as follows:

```
./defgen paramfile dirprm/defgen.prm
```

The command will read the file created in the previous step and create a table definition file which you must specify in the Java Extract file in the `sourceDefs` parameter. Open and examine the file created by the `defgen` command.

Since HotCache relies on JPA, in the next section you will setup a simple JPA project to create all of the artifacts that JPA requires, such as the `persistence.xml` file and the POJO representation of the table(s) you will be monitoring.

4. Setting Up JPA

JPA can operate either through the use of JPA annotations or by using an XML mapping file to represent a table. In section 4.2 you will learn how to use JPA annotations to define a JPA entity. In section 4.3, you will learn how to use a pure XML mapping to define a JPA entity.

4.1 Creating a JPA Project in Eclipse

This tutorial uses the Eclipse IDE to create a new JPA project.

Note:

- This section is optional as it deals only with JPA/TopLink. If you can create the `persistence.xml` file and the POJOs on your own or if you already have them, then you can skip this section because HotCache needs only the mandatory JPA artifacts.

Step 1. Setup a JPA Project in the Eclipse IDE

Open the Eclipse IDE. Create a new workspace named `HotCache`. In the Eclipse IDE, create a new JPA project by clicking **File**, then **New**, then **Others**, then **JPA**, then **JPA Project**, then **Next**. Name the new project `CoherenceHotCache`.

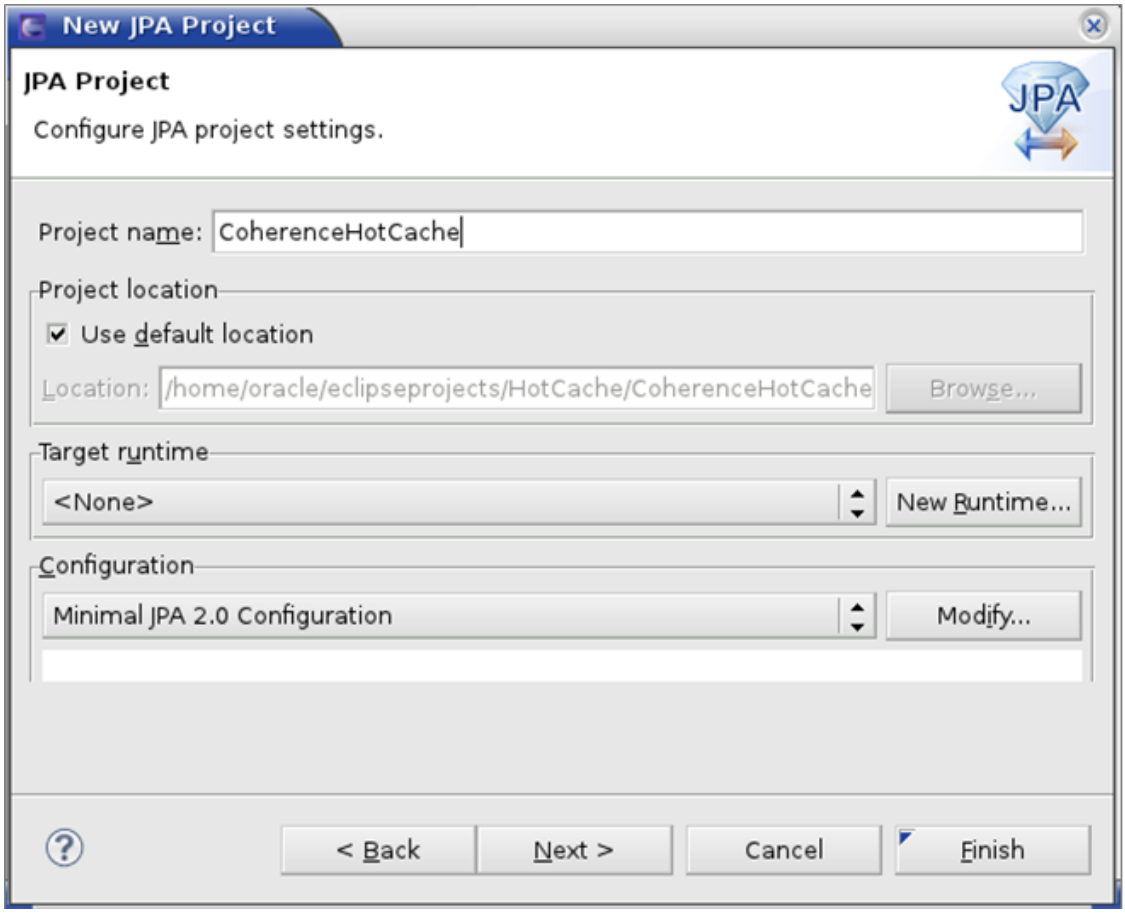

Figure 2. Configuring a JPA Project in Eclipse

Click **Finish**. Expand the project in the Eclipse Project Explorer. Under the `src` directory, create a new folder named `mypojos`.

Note:

- Under the `META-INF` directory, you should see that a `persistence.xml` file has been created for you by the JPA plugin.

Step 2. Edit the persistence.xml File

Double-click the `persistence.xml` file. The screen presented will allow you to customize the file. For now, simply define the connection to the database.

Click the **Connection** tab and select **Resource Local** from the **Transaction type** dropdown list. In the **JDBC connections properties** section, enter your database information. An example of the configuration is shown in the following illustration.

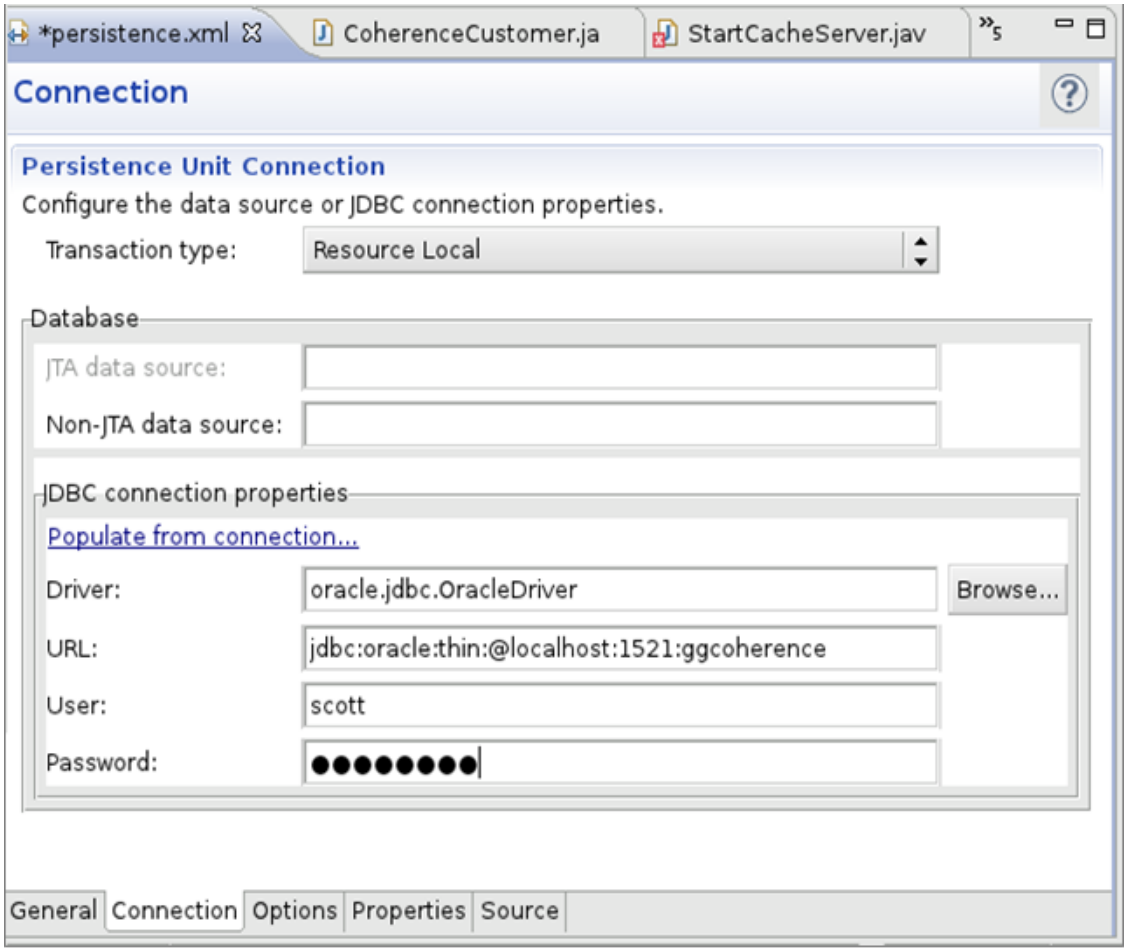

Figure 3. Configuring the Database Connection Screen in the Eclipse IDE

Ensure you add the `ojdbc6.jar` file to your project classpath.

4.2 Defining JPA Using Annotations

As stated earlier, JPA can be specified with either JPA annotations or XML mappings. If you would rather work with XML mappings instead of JPA annotations, skip to the next section.

The Eclipse JPA plugin allows you to create a JPA entity automatically by pointing it to a database table. To do this, right-click the directory you created (`mypojos`), and choose **New**, then **Entities from Tables**.

You will then be presented with a screen where you might need to create a database connection if one does not already exist.

Click the "+" symbol in the far right-hand corner. From the list presented, select your database and click **Next**. Enter all of the required information for the database, then test the connection. Click **Finish** if the connection is successful.

From the **'Schema'** drop-down list, select your schema and then select the table you created earlier. Click **Finish**.

Two things were accomplished with this procedure:

- The JPA plugin creates a POJO representation of your table.
- The JPA plugin updates the `persistence.xml` file to reflect that this class will be managed by JPA. The newly created class should be called `Customers` and it should appear under the `src/mypojos` directory.

In the `persistence.xml` file, the JPA plugin adds an entry representing the POJO that it created. Examine the `persistence.xml` file to confirm this.

Note:

- If you look at the `persistence.xml` file, you will see that the top level is `persistence-unit`. Make a note of the value of the name attribute as you will need it later.

4.3 Defining JPA Using XML mapping

To use an XML mapping to setup JPA, you must create these items:

- A POJO representing the table you are persisting to.
- An XML mapping file (the default name of the file is `orm.xml`).

Earlier in this tutorial, you created a table named `Customers`. Now create a POJO to represent the `Customers` table.

Step 1. Create a Java Class to Represent the Table

Create a new Java class named `Customers.java` in the `mypojos` folder. The following figure illustrates a sample `Customers.java` file.

```
import java.io.Serializable;

public class Customers implements Serializable {
    private static final long serialVersionUID = 1L;
    private long custid;
    private String firstname;
    private String lastname;
    private String telephone;

    public Customers() {}

    Getter/Setter methods
}
```

Figure 4. Listing of the Customer.java File

Step 2. Create a JPA Mapping File

Create a JPA mappings file to represent the `Customer.java` class you just created. Right-click the `META-INF` directory, select **New**, then **Mapping File**. The following figure illustrates the Mapping File screen.

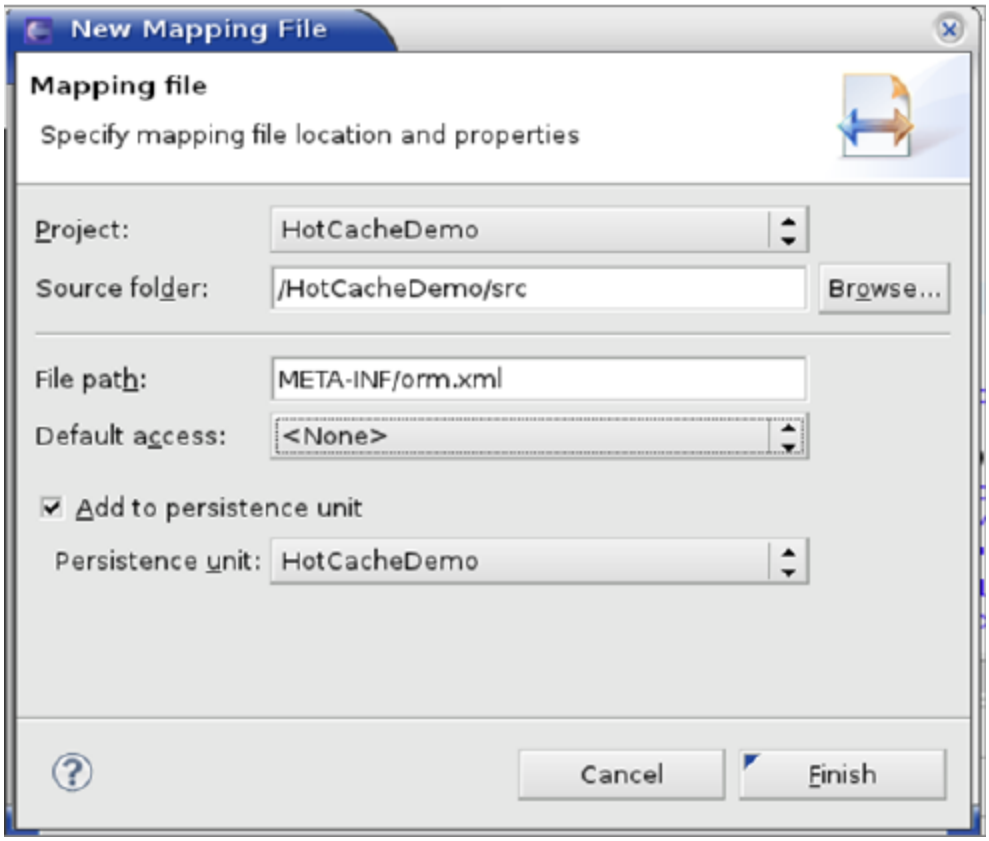

Figure 5. Configuring the New Mapping File Screen in the Eclipse IDE

Make sure that the **Add to Persistence unit** checkbox is selected and click **Finish**. This will create a file named `orm.xml` in the `META-INF` directory. The `orm.xml` file defines the JPA XML mappings. Update the `orm.xml` file to reflect the `Customers` table. The following figure illustrates the contents of the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings version="2.0" xmlns="http://java.
    <package>mypojos</package>
    <entity class = "Customers" name="Customers">
       <table name="CUSTOMERS"/>
         <attributes>
            <id name="custid">
            </id>
            <basic name="firstname"/>
            <basic name="lastname"/>
            <basic name="lastname"/>
        </attributes>  </entity>
  </entity-mappings>
```

Figure 6. Customer Table Mapping in the orm.xml File

A discussion of JPA XML mapping is outside the scope of this tutorial. A good starting point to learn about XML mapping is at the following URL:

http://en.wikibooks.org/wiki/Java_Persistence/Mapping

See also "Pro JPA 2" by Mike Keith:

http://www.amazon.com/Pro-JPA-2-Mike-Keith/dp/1430249269

4.4 Testing JPA Persistence Using JPA APIs

This section is not directly concerned with HotCache, but it is presented to ensure that your JPA/TopLink configuration is correct. Test your JPA/TopLink configuration by inserting new records into the database and selecting them back.

Step 1. Create a Java Class to Test the JPA Configuration

In the `mypojos` directory, create a Java class named `NewCustomers` which uses the JPA APIs to insert a new record into the database table. The Java class should look similar to the following code:

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;


public class NewCustomers {

    private static EntityManagerFactory emf;
    private static EntityManager em;
```

```
    public static void main(String[] args)
    {
        Customer customer = new Customer();
        customer.setCustid(10);
        customer.setFirstname("John");
        customer.setLastname("Smith");
        customer.setTelephone("1-416-702-0771");

        //Persist using JPA APIs
        //CoherenceHotCache is the persistence unit in
persistence.xml

        emf =
Persistence.createEntityManagerFactory("CoherenceHotCache");
        em = emf.createEntityManager();
            em.getTransaction().begin();
            em.persist(customer);
            em.getTransaction().commit();
            em.close();
    }
}
```

Step 2. Run the Test Class

Run the class. If all goes well, a new record is inserted into the database. Check the table using your choice of SQL viewer:

```
SQL> select * from customers;

   CUSTID FIRSTNAME          LASTNAME           TELEPHONE
--------- ------------------ ------------------ --------------
       10 John               Smith              1-416-702-0771
```

Figure 7. New Customer Record Returned from the Database

Step 3. Test the JPA Configuration Using Named Queries

To further test your JPA/TopLink connection, add a couple of Named Queries to the `Customer.java` class that was created by the JPA plugin. Under the `@Table` annotation, add the following annotations:

```
@NamedQueries({
@NamedQuery(name = "CountCustomers", query = "select count(c) from
Customer c"),
@NamedQuery(name = "GetCustomer", query = "select c from Customer c
where c.custid=:id")})
```

Step 4. Create a Java Class to Get a Customer

Create a Java class called `GetCustomer.java`. The code should look similar to the following:

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class GetCustomer {

    private static EntityManagerFactory emf;
    private static EntityManager em;

    public static void main(String[] args)
    {
        //First let get the number of records in our table
        emf =
Persistence.createEntityManagerFactory("CoherenceHotCache");
        em = emf.createEntityManager();

        Query q1 = em.createNamedQuery("CountCustomers");
        System.out.println("Number of Records :" +
q1.getSingleResult());

        //Now let's get the record
        Query q2 = em.createNamedQuery("GetCustomer");
        q2.setParameter("id", 10);
        Customer cust = (Customer)q2.getSingleResult();
        System.out.println("ID : " + cust.getCustid() + "," +
cust.getFirstname());

    }
}
```

Now that you have `persistence.xml` and the POJO representing your database table, you are ready to setup Coherence.

Step 5. Export the Eclipse Project

Export the Eclipse project you just created because some of the artifacts it contains will be needed by the HotCache adapter and must be in its classpath. To do this, Click **File**, then **Export**, then **Java**, then **JarFile,** and specify the destination where the file will be created. For example, you can name your JAR file `customerJPA.jar`.

4.5 Testing Persistence using Coherence APIs

The previous section discussed testing database persistence using JPA APIs. Another way to test your JPA setup is to have Coherence perform the database read/write for you.

Coherence supports automatic read/write caching of any data source, including databases, web services, packaged applications, and file systems. However, databases are the most common use case.

Effective caches must support both intensive read-only and read/write operations, and in the case of read/write operations, the cache and database must be kept fully synchronized. To do this, Coherence supports Read-Through, Write-Through, Refresh-Ahead, and Write-Behind caching.

Once Coherence is setup to read/write to a database you can use the same Coherence APIs that you use to interact with Coherence. This is because Coherence is configuration-based, which makes this feature especially powerful as it does not require any code changes or learning a new set of APIs.

For example, if you have a very simple client which `puts` a `Person` POJO using Coherence, the code would look similar to this:

```
NamedCache myCache = CacheFactory.getCache("Person");

Person person = new Person();
person.setId(2);
person.setName("Ian");
person.setAddress("Mississauga, Canada");

myCache.put(person.getId(), person);
```

Figure 8. Java Listing to Put a Person Object In the Cache

If Coherence is running in the default configuration, then it will simply save the entry in memory. If, however, Coherence is configured to read/write to the database, then it will insert/update the database with the new/updated data.

The same concept applies to `getting` data from Coherence. Consider the following example for getting the entry that was `put` into Coherence by the previous code example:

```
NamedCache myCache = CacheFactory.getCache("Person");
Person person = (Person)myCache.get(2);
```

Figure 9. Getting the Person Entry from the Coherence Cache

In the case of getting an entry from the cache, Coherence will check to see if it has the entry. If it does not, then it will fetch the entry from the database and save it in memory.

Note:

- As stated earlier, Coherence supports Read-Through, Write-Through, Refresh-Ahead and Write-Behind caching. To learn about the different read/write approaches, see "Caching Data Sources" in *Developing Applications with Oracle Coherence* at: http://docs.oracle.com/middleware/1212/coherence/COHDG/cache_rtwtwbra.htm#CFHEJHCI

Coherence offers database read/write by using a `CacheStore`. A `CacheStore` is an application-specific adapter used to connect a cache to an underlying data source. The `CacheStore`

implementation accesses the data source by using a data access mechanism, such as Hibernate, JPA (the standards-based approach), or TopLink.

The `CacheStore` implementation understands how to build a Java object using data retrieved from the data source, map and write an object to the data source, and erase an object from the data source.

Defining a Coherence `CacheStore` is beyond the scope of this tutorial, but please check the links given above for a detailed description of how to setup out-of-the-box Coherence `CacheStore` implementation(s).

Coherence offers three out-of-the-box `CacheStore` implementations based on Hibernate, JPA, or TopLink. You can also develop a custom `CacheStore` to suit your needs.

5. Setting Up Coherence Extend

Using Coherence*Extend is the recommended way for HotCache to connect to Coherence, but you can also connect to Coherence using the default protocol (TCMP).

If you are just testing, then you can skip this section. When you need to start Coherence, you can just run `$COHERENCE_HOME/bin/cache-server.sh`.

Coherence*Extend consists of two basic components: a client running outside the cluster, and a proxy service running inside the cluster. The client API includes implementations of both the `CacheService` and `InvocationService` interfaces which route all requests to a proxy running within the Coherence cluster. The proxy service in turn responds to client requests by delegating to an actual Coherence clustered service (for example, a Partitioned or Replicated cache service).

In this section you will create a cache server with Coherence*Extend enabled. Enabling Coherence*Extend is very easy. All you need to do is modify the Coherence cache configuration file and enable the proxy service.

Clients can connect using Extend in the following ways:

- To a cache server process whose cache configuration file enables a proxy-scheme; or
- To a dedicated proxy server process not acting as a cache server.

Since a proxy server will be handling all client connections, it is advisable for it to be storage-disabled.

To simplify Coherence setup, this tutorial will use the first approach.

Note :

- Regardless of which option you use to connect to Coherence through extend, a cache name must be specified on both the client-side and the server-side cache configuration files.

5.1 Configuring Coherence*Extend for the Cache Server

Follow these steps to configure Coherence*Extend for the cache server:

Step 1. Configure the Coherence Cache Configuration File

If you do not have a `coherence-cache-config.xml` file, you can extract it from the `coherence.jar` file available in the `$COHERENCE_HOME/lib` directory.

Enable Coherence*Extend. Open the `coherence-cache-config.xml` file and search for the proxy service section which should look similar to the following:

```
<proxy-scheme>
     <scheme-name>example-proxy</scheme-name>
     <service-name>TcpProxyService</service-name>
     <acceptor-config>
        <tcp-acceptor>
         <local-address>
           <address system-
property="tangosol.coherence.extend.address">localhost</address>
           <port system-
property="tangosol.coherence.extend.port">9099</port>
        </local-address>
        </tcp-acceptor>
    </acceptor-config>
    <autostart system-
property="tangosol.coherence.extend.enabled">false</autostart>
 </proxy-scheme>
```

The important values here are in bold. These values specify the host where Coherence*Extend will be running and the TCP port it will be listening to. The value of the last property, `tangosol.coherence.extend.enabled`, should be changed to `true`.

Step 2. Test Coherence*Extend

To test, make a copy of the `cache-server.sh` or `cache-server.cmd` file available under `$COHERENCE_HOME/bin` and name it `cache-server-extend.sh` or `cache-server-extend.cmd`, for example. Open the newly created file and add the following `-D` flag:

`-Dtangosol.coherence.cacheconfig=` *path_to_the_server_cache_configuration_file*

Now run the cache server. If you look closely at the output of the running server, the last section should display a list of enabled Coherence Services. The last line (`ProxyService`) indicates that the proxy service that you enabled is running.

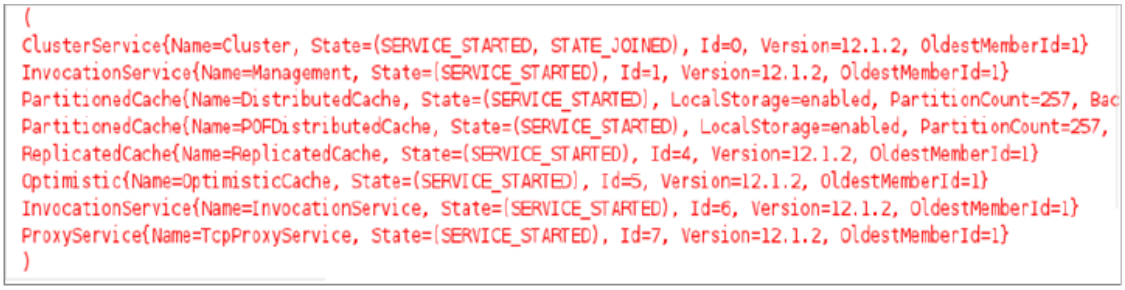

Figure 10. Services Enabled in the Cache Server Output

5.2 Configure Coherence Extend for the Cache Client

Create a new Coherence cache configuration for the client side. Name the file `extend-client-config.xml`. You can copy and paste the following XML code into the file:

```
<?xml version="1.0"?>
<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-
cache-config coherence-cache-config.xsd">
   <caching-scheme-mapping>
       <cache-mapping>
         <cache-name>*</cache-name>
         <scheme-name>remote</scheme-name>
      </cache-mapping>
   </caching-scheme-mapping>
   <caching-schemes>
     <remote-cache-scheme>
        <scheme-name>remote</scheme-name>
        <service-name>ExtendTcpCacheService</service-name>
        <initiator-config>
           <tcp-initiator>
              <remote-addresses>
                 <socket-address>
                    <address>localhost</address>
                    <port>9099</port>
                 </socket-address>
             </remote-addresses>
             <connect-timeout>10s</connect-timeout>
          </tcp-initiator>
          <outgoing-message-handler>
           <request-timeout>5s</request-timeout>
          </outgoing-message-handler>
        </initiator-config>
     </remote-cache-scheme>
   </caching-schemes>
</cache-config>
```

The important elements are marked in bold. The `cache-name` element indicates that any cache name is accepted. For this to work, a wildcard value `*` for the `cache-name` element must also exist in the

Coherence cache configuration file on the server side. If you look back to the server-side file you created in the previous section, you will see that this file also has a wildcard value `*` for the `cache-name` element specified by default.

The second `address` and third `port` elements must match the values you defined in the Coherence cache configuration file on the server side.

Note:

- You can define multiple addresses for Extend Servers.

5.3 Using POF-Enabled Objects with HotCache

To use POF-enabled POJOs with HotCache, add the `oracle.eclipselink.coherence.integrated.cache.TopLinkGridPortableObject` and `oracle.eclipselink.coherence.integrated.cache.TopLinkGridSerializer` classes to your POF configuration file, as illustrated in the following figure:

```
<user-type>
  <type-id>9001</type-id>
    <class-name>oracle.eclipselink.coherence.integrated.cache.TopLinkGridPortableObject</class-name>
    <serializer>
      <class-name>oracle.eclipselink.coherence.integrated.cache.TopLinkGridSerializer</class-name>
    </serializer>
</user-type>
```

Figure 11. POF Configuration for HotCache

Note:

- Make sure to add these Coherence POF-enabled flags to the Java User Exit properties file:
  `-Dtangosol.pof.enabled=true`
  `-Dtangosol.pof.config=`*path to your POF configuration file*

5.4 Testing Coherence

To test that your client configuration is working, make a copy of `$Coherence_Home/bin/coherence.sh` cache server file and save it as `extend-coherence.sh`, for example. Use the cache configuration system property `-Dtangosol.coherence.cacheconfig=` to specify the cache configuration file you just created.

Note:

- The `coherence.sh` (or `coherence.cmd` on Windows platform) file is a command line provided by Coherence which allows you to view/put entries in a Coherence cache, among other things.
- Make sure that `toplink-grid.jar` and `eclipselink.jar` are in the classpath of the Coherence nodes.

Run the `extend-coherence.sh` file. This returns a `Map(?)` prompt. Enter `cache testing`, which creates a cache named `testing`, if one does not already exist. Then put a new entry into the cache:

```
Map (?): cache testing
Map (testing): put 1 test
```

Start another instance of `extend-coherence.sh`. First, get the cache you created in the first session:

```
Map (?): cache testing
```

Try to get the entry you just added in the first session:

```
Map (testing): get 1
test
```

If the correct entry is returned, then your Coherence Extend configuration is working. To exit the prompt, enter `bye`.

6. Configuring HotCache

HotCache uses a custom Java API provided by GoldenGate. You must define the following items:

- A Primary Extract parameter file which reads the transaction log and writes changes to a trail file.
- A Java Extract parameter file which reads the trail file created by the Primary Extract, formulates the change into an XML document, and passes it to the Java User Exit.
- A Java User Exit properties file where all HotCache configuration and properties are defined.

Note :

- Once Extract files are created, register the Extract parameter files with the GoldenGate Manager process and associate them with the trail file(s) it will be writing to or consuming from. Before you start setting up the Extracts, set the LD_LIBRARY_PATH variable which should include $GG_HOME. Depending on your hardware, (64-bit or 32-bit) you must also add `$JAVA_HOME/jre/lib/amd64/server` for 64-bit , or `$JAVA_HOME/jre/lib/i386/server` for 32-bit platforms.
- On Windows platforms, be sure to add `$JAVA_HOME/jre/lib/i386/server` to your `$PATH` variable.

6.1 Creating Primary Extract File

The Primary Extract file is used by GoldenGate to extract the updated records from the redo logs into a trail file.

Create the Primary Extract file and name it `custext.prm`. The filename is limited to eight characters and make sure that it is created under the `$GG_HOME/dirprm` directory.

Following are the contents of this file:

```
EXTRACT custext
USERID ggowner, PASSWORD ggowner
```

```
EXTTRAIL /home/oracle/ggtrails/customers/cu
getUpdateBefores
TABLE SCOTT.CUSTOMERS;
```

Each line has the following meanings:

- `EXTRACT` defines this parameter file as an Extract parameter file for the Extract process named in this line. The name of the Extract must match the filename for this Extract parameter file.
- The `USERID/PASSOWRD` will be used by the Extract process to login to the database.
- `EXTTRAIL` points GoldenGate to where the trail file (discussed in "1. Understanding GoldenGate") should be created. Keep in mind that the prefix (`cu`) will be added to the file name generated by GoldenGate automatically and should only consist of two characters.
- `GetUpdateBefores` will send the "before" view of the data that changed.
- The last line is the name of the table to monitor. You can specify a schema to monitor by using this format:
  `TABLE scott.*`

6.2 Creating a Java Extract File

The Java Extract file is used by HotCache to read from the trail file created by the Primary Extract file defined in the previous section.

Create the Java Extract file and name it `hcjavaue.prm`. Following is the content of the file:

```
EXTRACT hcjavaue
USERID ggowner, PASSWORD ggowner
setEnv (GGS_USEREXIT_CONF =
"/home/oracle/oracle_products/Middleware/gg11gr2/dirprm/hotcache.prop
erties")
GetEnv (JAVA_HOME)

GetEnv (PATH)

GetEnv (LD_LIBRARY_PATH)
CUserExit
/home/oracle/oracle_products/Middleware/goldengate/libggjava_ue.so
CUSEREXIT PASSTHRU INCLUDEUPDATEBEFORES
sourceDefs
/home/oracle/oracle_products/Middleware/gg11gr2/dirdef/customerdef.de
f
getUpdateBefores
NoTcpSourceTimer
TABLE SCOTT.CUSTOMERS;
```

Each line has the following meanings:

- `EXTRACT` defines the `extractor` name.

- The `USERID/PASSWORD` will be used by the Extract process to login to the database.
- The `SetEnv` points to the Java User Exit properties file you will create in the next section.
- `CuserEXit` points to the required native library (`libggjava_ue.so` for Linux/UNIX, `ggjava_ue.dll` for Windows).
- The `sourceDefs` is the table definition file you created earlier.
- `getUpdateBefores` includes the "before" data in the capture.
- The name of the table to monitor.

6.3 Creating Java User Exit Properties File

Create the User Exit properties file. This file indicates to the Extract process which Java class to invoke, along with all of the parameters this class requires.

Use a text editor to create the file and call it for example `hotcache.properties` and save it in the `$GG_HOME/dirprm` directory. The file should have the following entries:

```
#=============================================================
# List of active event handlers. Handlers not in the list are
ignored.
gg.handlerlist=cgga
#=============================================================
# Coherence cache updater
gg.handler.cgga.type=oracle.toplink.goldengate.CoherenceAdapter
#=============================================================
gg.brokentrail=true
goldengate.log.logname=cuserexit

goldengate.log.level=DEBUG

goldengate.log.tofile=true
#=============================================
# Native JNI library properties
goldengate.userexit.nochkpt=true
goldengate.userexit.writers=jvm
#=============================================
jvm.bootoptions=-Xmx32M -Xms32M -Dtoplink.goldengate.persistence-
unit=CoherenceHotCache -Dlog4j.configuration=
/home/oracle/oracle_products/Middleware/gg11gr2/ggjava/resources/debu
g-log4j.properties
-Dtangosol.coherence.cacheconfig=
/home/oracle/Documents/goldenGate/hotcache/client-extend-config.xml -
Dlog4j.debug=true -Djava.class.path=
/home/oracle/oracle_products/Middleware/gg11gr2/dirprm:/home/oracle/o
racle_products/dbxe/product/11.2.0/dbhome_1/jdbc/lib/ojdbc6.jar:/home
/oracle/oracle_products/Middleware/gg11gr2/ggjava/ggjava.jar:/home/or
acle/oracle_products/Middleware122c/coherence/lib/coherence.jar:/home
/oracle/oracle_products/Middleware122c/oracle_common/modules/javax.pe
rsistence_2.0.0.0_2-
```

```
0.jar:/home/oracle/oracle_products/Middleware122c/oracle_common/modul
es/oracle.toplink_12.1.2/eclipselink.jar:/home/oracle/oracle_products
/Middleware122c/oracle_common/modules/oracle.toplink_12.1.2/toplink-
grid.jar:/home/oracle/Documents/goldenGate/hotcache/customerJPA.jar
```

Let's walk through the entries in this file:

- In the `gg.handlerlist` property, the value is `cgga`. The value can be any name; as long as you enter the same name in the handler type which is named `gg.handler.cgga.type`. The `oracle.toplink.goldengate.CoherenceAdapter` class is the custom Java class which uses GoldenGate custom Java APIs. This class is HotCache.
- The `goldengate.userexit.nochkpt` property is used to disable the User Exit checkpoint file. This example defines the User Exit checkpoint file to be disabled.
- The `goldengate.userexit.writers` property specifies the name of the writer. The value must be `jvm` to enable calling out to the GoldenGate Java Adapter.
- The last property, `jvm.bootoptions`, should include all the JAR files and `-D` flags that are needed for the HotCache adapter to be successfully invoked.

Examine the entries in the file above and change the path according to your directory structure.

Note:

- Make sure you include the packaged JPA application in the classpath in the Java User Exit properties file. The value of the system property `-Dtoplink.goldengate.persistence-unit` should be the persistence unit name in your `persistence.xml` file.
- The entries in bold will help for debugging.

Keep in mind that when the Java Extract process starts, it will create two log files that will be very helpful in analyzing any issues. Both files are created under `$GG_HOME`. The first file is prefixed with `cuserexit` as defined for the parameter `goldengate.log.logname` in the Java properties file.

The second file is the Log4J `log` file, also available under `$GG_HOME`. To get this file, make sure that you have Log4J `-D` flags defined in the Java properties file. Define the following:

- `-Dlog4j.configuration` property points to the Log4j configuration file. The GoldenGate Java adapter installation provides one under `$GG_JAVA_ADAPTER home/resources/classes`. Note that the `$GG_JAVA_ADAPTER` refers to the directory where the GoldenGate Java adapter was unzipped. If you use the Log4j configuration file packaged with the GoldenGate Java adapter, then the file name created will be `debug-ggue-11.1.1.0.0.006-log4j.log`.
- `-Dlog4j.debug=true`
- `-Dlog4j.logger.oracle.toplink.goldengate=DEBUG, stdout, rolling`

In addition, the following Log4j property appears in the properties file:

- `log4j.logger.oracle.toplink.goldengate=WARN, stdout, rolling`

6.4 Register the Extract Files

Now that you have created the Java Extract file and the Java User Exit properties file, you must register them with the GoldenGate Manager process.

Step 1. Begin Extract File Registration

Start with a clean workspace. Stop the Manager process, restart it, then check the status:

```
GGSCI (bhijazi-linux) 1> stop *
GGSCI (bhijazi-linux) 2> stop mgr
GGSCI (bhijazi-linux) 3> start mgr
Check that the manager is running :
GGSCI (bhijazi-linux) 4> status mgr
Manager is running (IP port bhijazi-linux.7809)
```

Step 2. Login to the Database

Use the following command to login to the database:

```
GGSCI (bhijazi-linux) 5> dblogin userid ggowner, password ggowner
```

Successfully logged in to the database.

Step 3. Register the Primary Extract File

Register the Primary Extract `custext` with the `add extract` command:

```
add extract custext, tranlog, begin now
```

This command instructs the GoldenGate Manager process to register the Extract called `custext` that you created earlier. The `tranlog` parameter means that the Extract process will be capturing changes from the transaction log(s). The last parameter, `begin now`, creates a checkpoint to direct the Extract to begin reading from the log at that point, once it is started.

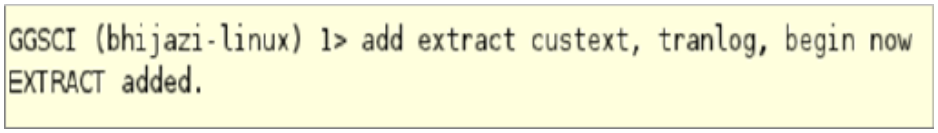

Figure 12. Registering the Primary Extract

Step 4. Register the Java Extract File

Register the Java Extract parameter file `hcjavaue` with the `add extract` command:

```
add extract hcjavaue, exttrailsource /home/oracle/ggtrails/customers/cu
```

This command instructs GoldenGate to add the Java Extract you created earlier. The `exttrailsource` parameter also indicates that the source of the records to be read by this Extract process will be the file that the Primary Extract created.

```
GGSCI (bhijazi-linux) 2> add extract hcjavaue, exttrailsource /home/oracle/ggtrails/customers/cu
EXTRACT added.
```

Figure 13. Registering the Java Extract File

Note:
It is very important that the file specified `/home/oracle/ggtrails/customers/cu` in the `add extract` command above matches exactly the one defined in the Primary Extract file (`custext.prm`) in the `EXTTRAIL` parameter.

6.5 Associating Primary Extract File with the Trail File

In the Primary Extract parameter file `custext.prm` that you created earlier, you defined an `EXTTRAIL` parameter. You must associate the `EXTRAIL` trail file with the `custext` Extract.

Step 1. Associate the Primary Extract File with the Trail File

Use the `add extract` command to associate the trail file with the Primary Extract file:

```
add exttrail /home/oracle/ggtrails/customers/cu, extract custext,megabytes 50
```

```
GGSCI (bhijazi-linux) 3> add exttrail /home/oracle/ggtrails/customers/cu, extract custext,megabytes 50
EXTTRAIL added.
```

Figure 14. Associating the EXTTRAIL Trail File with the custext Extract

Step 2. Start a Storage-enabled Coherence Node

You must start a storage-enabled Coherence node before you can start the Primary Extract and the Java Extract processes.

Step 3. Start the Primary Extract Process

Enter the `start extract custext` command to start the `custext` Primary Extract:

```
GGSCI (bhijazi-linux) 8> start extract custext
```

```
GGSCI (bhijazi-linux) 4> start extract custext

Sending START request to MANAGER ...
EXTRACT CUSTEXT starting
```

Figure 15. Starting the custext Extract

Step 4. Test the Primary Extract Process

To check that the Extract process has started, enter the following command:

```
GGSCI (bhijazi-linux) 9> info extract custext
```

```
GGSCI (bhijazi-linux) 26>  info extract custext

EXTRACT    CUSTEXT   Last Started 2013-09-18 14:51   Status RUNNING
Checkpoint Lag       00:00:00 (updated 00:00:03 ago)
Log Read Checkpoint  Oracle Redo Logs
                     2013-09-18 14:51:04  Seqno 13, RBA 22495232
                     SCN 0.1242961 (1242961)
```

Figure 16. Obtaining Information about the Extract

The important point (other than the status should be running), is the value of `Log Read Checkpoint.` This indicates that the Extract is capturing changes from `Oracle Redo Log`.

The `SCN` output (last line) displays the system change number. This is Oracle Database's clock. Every time you commit, the clock increments. The `SCN` value marks a consistent point in time in the database.

Insert a new record into the database and see how the `SCN` increments. Using an SQL editor, insert a new record into the `customers` table you created earlier. Then, commit the record. The following commands will create one new row in the database.

```
insert into customers (custid,firstname,lastname,telephone) values
(2,'John','Smith','222-222-2222');

commit;
```

```
SQL> insert into customers (custid,firstname,lastname,telephone) values (2,'John
','Smith','222-222-2222');

1 row created.

SQL> commit;
```

Figure 17. Adding a Record to the Database

Enter the `info extract custext` command again and examine the `SCN` number:

```
GGSCI (bhijazi-linux) 10> info extract custext
```

```
GGSCI (bhijazi-linux) 27>  info extract custext

EXTRACT    CUSTEXT   Last Started 2013-09-18 14:51   Status RUNNING
Checkpoint Lag       00:00:00 (updated 00:00:03 ago)
Log Read Checkpoint  Oracle Redo Logs
                     2013-09-18 14:52:53  Seqno 13, RBA 22661120
                     SCN 0.1243126 (1243126)
```

Figure 18. Checking the new SCN Number

As you can see, the SCN has incremented.

Another thing that happens when you first commit, is that the Primary Extract process will create the trail file and will write the database change it just captured to the trail file.

Try this very useful GoldenGate command which tells the number of operations of each type that an Extract process has captured:

```
GGSCI (bhijazi-linux) 9>stats extract custext total
```

```
GGSCI (bhijazi-linux) 19> stats extract custext total

Sending STATS request to EXTRACT CUSTEXT ...

Start of Statistics at 2013-09-18 14:47:28.

Output to /home/oracle/ggtrails/customers/cu:

Extracting from SCOTT.CUSTOMERS to SCOTT.CUSTOMERS:

*** Total statistics since 2013-09-18 14:47:07 ***
        Total inserts                                       1.00
        Total updates                                       0.00
        Total deletes                                       0.00
        Total discards                                      0.00
        Total operations                                    1.00

End of Statistics.
```

Figure 19. Running Statistics for an Extract

As you can see from the illustration, one insert has been captured and output to the `/home/oracle/ggtrails/customers/c` trail file.

Troubleshooting Problems with the Primary Extract Process

If the output of the `info extract custext` command displays a `STOPPED` or `ABENDED` message such as the following, this indicates that there was a problem starting the Extract.

```
EXTRACT    JAVAUE    Initialized   2013-09-04 17:07   Status STOPPED
Checkpoint Lag       00:00:00 (updated 00:16:31 ago)
Log Read Checkpoint  Oracle Redo Logs
                     2013-09-04 17:07:27  Seqno 0, RBA 0
                     SCN 0.0 (0)
```

To view a report about a process regardless if it succeeded or failed, use the following command to check its status:

```
GGSCI (bhijazi-linux) 10 > view report custext
```

This command displays a log file. Navigate through the output and see what is causing the error. Usually, the error messages are pretty good so you should be able to figure out the exact reason why the process failed to start.

Step 5. Start the Java Extract Process

Start the Java Extract process `hcjavaue` with the `start extract` command:

```
GGSCI (bhijazi-linux) 11> start extract hcjavaue
```

If the Java Extract starts successfully, you can check the status of GoldenGate by invoking the `status all` command:

```
GGSCI (bhijazi-linux) 1>status all
```

```
GGSCI (bhijazi-linux) 17> status all

Program     Status      Group       Lag at Chkpt  Time Since Chkpt

MANAGER     RUNNING
EXTRACT     RUNNING     CUSTEXT     00:00:00      00:00:08
EXTRACT     RUNNING     HCJAVAUE    03:58:06      00:00:39
```

Figure 20. Running Status for All Extracts

7. Testing HotCache

To test HotCache, all you need to do is insert/update a record into the table being monitored by GoldenGate and then check that an entry was created in Coherence. To test Coherence, follow the steps outlined in Section 5.4 "Testing Coherence".

Note:

- The name of the cache that the HotCache adapter creates, by default, is the unqualified class name of JPA Entity representing your table. In this case, the cache name is `Customer`.

8. Using Relationship Definitions with HotCache

At the time of this writing, HotCache does not support the following JPA annotations:

- `@OneToOne`
- `@OneToMany`
- `@ManyToMany`

To implement relationships with HotCache, use the `@Embeddable` and `@ElementCollection` annotations to represent relationships.

8.1 Implementing OneToOne Relationships

To illustrate this scenario, assume that you have two tables, `Resident` and `ResidentInfo`, with a OneToOne relationship.

To map this relationship with JPA (with HotCache in mind), the easiest way is to create a Java class (for example, `Resident.java`) where the columns from both tables are represented as attributes. For example:

Table `ResidentInfo` has the following fields:

- `ID` – `Number` // Primary key
- `Address` – `Varchar2`

Table `Resident` has the following fields:

- `ID` – `Number` // Primary key and is referenced as a foreign key in the `ResidentInfo` table
- `Name` – `String`

To split an entity into two tables, use the `@SecondaryTable` annotation which points to the `ResidentInfo` table.

The `Resident` class should look similar to the following:

```
@Entity
@Table(name="RESIDENT")
@SecondaryTable(name = "RESIDENTINFO",pkJoinColumns = @PrimaryKeyJoinColumn(name = "ID",referencedColumnName = "ID"))

public class Resident implements Serializable
{
   @Id
    private long id; //belongs to the Resident table
    private String name; //belongs to the Resident table

    //Below attributes represent columns of the Secondary table (RESIDENTINFO)
    @Column(table="RESIDENTINFO")
    private String street;
    @Column(table="RESIDENTINFO")
    private String city;
    @Column(table="RESIDENTINFO")
    private String state;
    @Column(table="RESIDENTINFO")
    private String country;

     public Resident(){}

     Getter/Setter methods
}
```

Figure 21. Java Listing for the Resident Class

As stated earlier, all of the columns from both tables are represented as attributes of this class. As you can see, the `@SecondaryTable` annotation is being used to point to `ResidentInfo` table and the `ID` column from both tables is being used to define the join. In addition, the `@Column` annotation is used specifically for attributes representing the fields of the `ResidentInfo` table.

8.2 Implementing OneToMany Relationships

Implementing the OneToMany relationship with HotCache in mind requires the use of the `@ElementCollection` and `@Embeddable` JPA annotations.

- `@ElementCollection` can be used to define a one-to-many relationship to an Embeddable object, or a Basic value (such as a collection of Strings). It is similar to using the `@OneToMany` annotation. The `ElementCollection` values are always stored in a separate table. The table is defined by using the `@CollectionTable` annotation or the `<collection-table>` element. The `@CollectionTable` annotation defines the table's name and `@JoinColumn` or `@JoinColumns` if a composite primary key.
- `@Embedded` defines a class whose instances are stored as an intrinsic part of an owning entity and share the identity of the entity. Each of the persistent properties or fields of the embedded object is mapped to the database table for the entity.

With this in mind, assume that you have two tables, `Orders` and `LineItems` where an order can have multiple `LineItems`.

Table `Order`:

- `OrderID` – `Number` // Primary key
- `OrderTotal` – `Float`

Table `LineItems`:

- `ItemID` – `Number` // Primary key
- `OrderID` – `Number` // Primary key
- `Name` – `String`
- `Price` – `Float`
- `Qty` – `Number`

Note:

- The primary key for this table is a composite key of `ItemID` and `OrderID`. In addition, the `OrderID` of the `Order` table is mapped as foreign key in the `LineItems` table.

Create two Java classes to represent the tables.

The `LineItems.java` class should look similar to the following:

```
@Embeddable
public class LineItems
{
    private long itemid;
    private String name;
    private double price;
    private int qty;

    public LineItems() {}

    Getter/Setter methods
}
```

Figure 22. Java Listing for the LineItems.java Class

Note the following features of the `LineItems` class:

- The `@Embeddale` annotation indicates that this class is a "dependent" class that has no identity without the presence of its parent, namely the `Order` class.
- Because `LineItems` does not exist without an order and therefore has no identity, an `@Id` annotation is not present.

Although the `LineItems` table has a field called `orderid`, there is no reason to include it in the `LineItems` class because it will be inserted automatically by the parent class since the parent class has an `orderid` defined.

The `Orders.java` class should look similar to the following:

```
@Entity
@Table(name="ORDERS")
public class Orders implements Serializable
{
  @Id
  private long orderid;
  private double ordertotal;

  @ElementCollection(targetClass=Lineitem.class,fetch=FetchType.EAGER)
  @CollectionTable(name="LineItems", joinColumns=@JoinColumn(name="orderid"))
  private Set<Lineitem> lineitems;

  public Orders() {}

  Getter/Setter methods
}
```

Figure 23. Java Listing for the Orders.java Class

The thing to note here is the `@ElementCollection` annotation which indicates that you have a OneToMany relationship where the target dependent class is the `LineItem` class. The `@CollectionTable` annotation also indicates that `LineItems` should be persisted to a table called `LineItems`.

Note:

- If the `@CollectionTable` annotation is omitted, then it is assumed that the `LineItem` class should be persisted to the `ORDER_LINEITEM` table.

9. Troubleshooting GoldenGate and HotCache

This section describes how to diagnose and fix problems in this GoldenGate and HotCache project.

9.1 Checking GoldenGate Extract Processes

If your Extract processes are running and nothing is happening, then you can get detailed information about your Extracts.

Using the GoldenGate command line interface, enter the following command to get detailed information about the `custext` Extract:

```
info extract custext
```

```
EXTRACT    CUSTEXT   Last Started 2013-09-18 15:01   Status RUNNING
Checkpoint Lag       00:00:00 (updated 00:00:04 ago)
Log Read Checkpoint  Oracle Redo Logs
                     2013-09-18 15:04:14  Seqno 13, RBA 25128448
                     SCN 0.1244215 (1244215)
```

Figure 24. Checking the Redo Logs

Make sure that the value of the Primary Extract `Log Read Checkpoint` field is `Oracle Redo Logs.`

Now invoke the same command, but for the Java Extract `hcjavaue`:

```
info extract hcjavaue
```

```
GGSCI (bhijazi-linux) 9> info extract hcjavaue

EXTRACT    HCJAVAUE  Last Started 2013-09-18 15:06   Status RUNNING
Checkpoint Lag       00:00:00 (updated 00:00:09 ago)
Log Read Checkpoint  File /home/oracle/ggtrails/customers/cu000000
                     First Record  RBA 0
```

Figure 25. Checking the Log for the Java Extract

The `Log Read Checkpoint` line indicates which trail file is being consumed by the Java Extract file.

Next, check if the file **`/home/oracle/ggtrails/customers/cu000000`** exists in your filesystem. If it is there, then insert a new record into the table and see if the file is updated.

It is crucial that the file in bold above matches the file defined in the Primary Extract parameter file.

9.2 Checking the HotCache Properties File

- It will help to enable logging. You should get three log files. It seems that when HotCache encounters an issue on the Windows platform, the screen just disappears. Adding the Coherence log file `-Dtangosol.coherence.log` system property to the HotCache properties file should help.

  The second log file is the User Exit log file. To generate this file, add the following parameters to the HotCache properties file:

  ```
  goldengate.log.logname=any name
  goldengate.log.level=DEBUG
  goldengate.log.tofile=true
  ```

  The third log file is the Log4j file. The GoldenGate Java adapter includes `debug-log4j.properties` which is very useful. The file it generates is available in the `$GG_HOME/ggjava/resources/classes` directory. To generate this file, add the system properties `-Dlog4j.configuration=`*name* and `-Dlog4j.debug=true`. The Log4j default name generated is `debug-ggue-11.1.1.0.0.006-log4j.log`, but you can change it in the Log4j configuration file.

- Make sure that both the storage-enabled Coherence node and the HotCache Adapter are using the same Coherence version.
- If you encounter the `No Persistence Provider for persistence-unit=`*xxx* issue, the most likely cause is that the adapter cannot find the `persistence.xml` file. Make sure it is in the adapter's classpath along with all of the POJOs.
- If the Java Extract fails with a `java.lang.ClassNotFoundException: oracle.toplink.goldengate.coherence.internal.CacheLoaderTypeProcessor` exception, then you must add the full `toplink-grid.jar` and `eclipselink.jar` files to the Coherence storage-enabled node classpath (make sure you have the full path).
- As stated earlier, the name of the cache that the adapter creates, by default, is the unqualified class name of the JPA Entity mapped to the database table. Make sure you either have the cache name specified in the Coherence configuration file or you have used the wildcard `*` for cache name.
- If you want to use a different cache name, use the `@Property` annotation or equivalent XML. For example:

  ```
  @Property(name = "eclipselink.coherence.cache.name", value =
  "CartProductListCache")
  ```

- If you get an error indicating that the `ggjava_ue.dll` (for Windows) or `libggjava_ue.so` (for Linux/UNIX) module could not be located, then it indicates that the `LD_LIBRARY_PATH` environment variable is not set. `$LD_LIBRARY_PATH` should include `$GG_HOME`, and depending on your hardware (32-bit or 64-bit) you must add `$JAVA_HOME/jre/lib/amd64/server` (for 64-bit), or `$JAVA_HOME/jre/lib/i386/`server (for 32-bit). On the Windows platform, make sure you add `$JAVA_HOME/jre/lib/i386/server` or `$JAVA_HOME/jre/lib/amd64/server` to your `$PATH` variable, as well.

White Paper Title
February 2014
Author: Bassam Hijazi
Contributing Authors: Valarie Bedard, David Felcey, Randy Stafford
Editor: Thomas Pfaeffle

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment

Hardware and Software, Engineered to Work Together