# BEA AquaLogic Commerce Services

## Developer Guide

Version 5.1.2
July 2007

# Copyright



# Restricted Rights Legend



# Trademarks and Service Marks

# Contents

# Overview

This document is intended to help developers build sophisticated online stores by providing detailed technical explanations of all areas of the AquaLogic Commerce Services product.

Sections include:

- **Setting up your development environment**
    - recommendations for setting up an efficient development environment to work with AquaLogic Commerce Services applications
- **Programming with AquaLogic Commerce Services**
    - helps a developer get familiar with the **core concepts** needed to build e-commerce applications on the AquaLogic Commerce Services platform
- **Architecture reference**
    - explains the underlying architectural aspects of AquaLogic Commerce Services that apply to a wide range of features
- **Building AquaLogic Commerce Services Applications**
    - This section lists a few of the more commonly used ANT tasks. A full listing of Ant tasks is available in each build.xml file.

# 1 - Setting up your development environment

Here we provide our recommendations for setting up an efficient development environment to work with AquaLogic Commerce Services applications. This configuration is used by our own product development team and is therefore our recommended setup, but you are of course free to use a different IDE, application server and database server for development.

# Creating a database in MySQL 4.0

To create a MYSQL database, please see the Creating a Database section in the Deployment guide.

# Setting up Eclipse and BEA Workshop

## *Downloading and installing Eclipse*

If you are using Eclipse as your development environment:

1. Download and unzip the Eclipse SDK at: http://www.eclipse.org/callisto/java.php. The current version as of the AquaLogic Commerce Services 5.0 release is 3.2.1; the following instructions are based on this version.

2. Start Eclipse

> **Tip**
> You may want to run AquaLogic Commerce Services applications from within Eclipse for faster code deployment, in which case you should allocate some minimum and maximum memory amounts by adding something like 'eclipse.exe -Xms128m -Xmx768m' to the shortcut.

3. Create or Select a workspace.

## Installing Eclipse Callisto plugins

> **Check your plugin versions**
>
> If you are using BEA Workshop, install plugins for Eclipse 3.1 rather than 3.2.

1. After the startup of Eclipse open the Install / Update dialog via the menu entry "Help > Software Updates > Find and Install ...".
2. Select "Search for new features to install".
3. Check the box for "Callisto Discovery Site", then click the "Finish" button.
4. After picking your mirror(s), you will have the opportunity to select the plugins you wish to install. We recommend installing, at a minimum:
    - Charting and Reporting - > Eclipse BIRT Report Designer Framework
    - Graphical Editors and Frameworks - > Graphical Editing Framework
    - Java Development - > J2EE Standard Tools (JST) Project
    - Testing and Performance - > (ALL) (Note: Please see TPTP for more information)
    - Web and J2EE Development - > Web Standard Tools, J2EE Standard Tools
5. Then click the "Select Required" button to resolve any dependencies, and click "Next" to continue with the installation process.

## Installing third-party Eclipse plugins

Once the Callisto plugins have been installed, also install the following plugins:

| Plugin Name | Plugin summary and installation instructions | Plugin website |
|---|---|---|
| Spring IDE | Spring (update site) | http://www.springide.org/ |

| Plugin Name | Plugin summary and installation instructions | Plugin website |
|---|---|---|
| PMD | PMD (update site) | http://pmd.sourceforge.net/integrations.html#eclipse |
| CheckStyle | Checkstyle (update site) | http://eclipse-cs.sourceforge.net/ |
| Velocity | Velocity (update site) | http://propsorter.sourceforge.net/veloeclipse/ |
| Hibernate Tools | Hibernate Tools (manual install) | http://www.hibernate.org/255.html |
| Amateras Eclipse HTML Editor | XML Editor (manual install) | http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=EclipseHTMLEditor |
| JSEclipse | JSEclipse (Optional commercial plugin - manual install) | http://www.interaktonline.com/Products/Eclipse/JSEclipse/Overview/ |

- Optional: Install Agent Controller remote TPTP profiling
  NOTE: You will have to restart your Eclipse workbench after unzipping the manually-installed plugins.

## *Setting up your environment*

**JAI**

If you do not start Eclipse with the JDK into which you installed SUN Java Advanced Imaging, or if it is not the default in Eclipse, then when you import your projects some files will show errors stating that Eclipse is unable to find JAI classes. Use the -vm argument when starting Eclipse to ensure that you're starting up with the appropriate JDK/bin directory. Alternatively, go to "Window > Preferences > Java > Installed JREs" and make sure that your default is the JRE or JDK with JAI libraries installed.

## Setting up the Eclipse WTP plugin for easy deployment and testing (Part 1)

The Eclipse WTP plugin allows a developer to configure their environment for automatic deployment and synchronization with Java projects open in Eclipse, but it can be tricky to set up. If you are using BEA Workshop, the you will already have tight integration with Weblogic and will not need to install the Eclipse WTP plugins.

1. From the File menu, select "New... Other". In the resulting dialog box, select "Server" from the "Server" folder, and hit "Next >".
2. Select the WebLogic Server and hit "Next >".
3. Select your JRE and fill in the text boxes. Hit "Next >".
4. Ignore the Available Projects for now, and hit "Finish".

## Setting up the Eclipse WTP plugin for easy deployment and testing (Part 2)

Now that your Eclipse projects are set up, you will want to be able to deploy them within Eclipse.

1. Expand the Servers folder in the Package Explorer, and edit the Server.xml file to tell the server where to find the class files to deploy when it starts up, and to tell it the path under which to deploy the application. A sample configuration is as follows:

```
<Context docBase="C:/ep/com.elasticpath.sf" path="/ep5sf"
reloadable="true">
     <Resource auth="Container"
driverClassName="com.mysql.jdbc.Driver"
maxActive="100" maxIdle="30" maxWait="10000" name="jdbc/epjndi"
password="mysql"
scope="Shareable" type="javax.sql.DataSource"
url="jdbc:mysql://localhost:3306/EP5_DEFAULT?useUnicode=true&amp;characterEncoding=UTF8"
username="root"/>
</Context>
<Context docBase="C:/ep/com.elasticpath.cm" path="/alcsm"
reloadable="true">
     <Resource auth="Container"
driverClassName="com.mysql.jdbc.Driver"
maxActive="100" maxIdle="30" maxWait="10000" name="jdbc/epjndi"
password="mysql"
scope="Shareable" type="javax.sql.DataSource"
url="jdbc:mysql://localhost:3306/EP5_DEFAULT?useUnicode=true&amp;characterEncoding=UTF8"
username="root"/>
   </Context>
```

The server.xml file is a powerful tool. You can use multiple server configurations and install the StoreFront on one server and the CommerceManager on a different server. Or, you can specify a docbase equal to a WAR file and have WTP/WebLogic Server deploy the war file (automatically expanding it) instead.
However, a word of caution if you decide to try deploying a WAR file directly: WebLogic Server does not automatically clean out the expanded WAR file, so if you change the WAR and/or replace it with a different one, then you should manually clean out the expanded directory before you restart your server. The expanded directory location can be found in the server's launch configuration (see below).

2. Tell the WTP/Eclipse where to find the source code for the projects you just configured in server.xml.
    - The easiest way to do this is to select Window > Show View > Servers to see the "Servers" tab.
    - Double-click the server on that tab to open the "Server Overview" tab.
    - Make sure that the checkbox "Run modules directly from the workspace" is checked.
    - Click "Open launch configuration", and on the "Source" tab click "Add", then add your three projects.
    - Note that on the Arguments tab you can see what base directory the plugin is using to run your project. Don't forget to manually clean out any expanded WAR files if you choose to use the plugin to deploy WAR files for quick testing.
3. You should now be able to start your server by right-clicking it in the Server tab and selecting "Start" or "Debug".

## *BEA Workshop and Eclipse Plugins*

This section describes several plugins for BEA Workshop and Eclipse that can be useful when customizing AquaLogic Commerce Services. When installing plugins for BEA Workshop, select the Eclipse 3.1 version.

### Checkstyle plugin for Eclipse

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard.

(Above content obtained from: http://checkstyle.sourceforge.net/)

### Installation

From the checkstyle website:

1. Within Eclipse go to Help > Software Updates > Find and Install.
2. Choose Search for new features to install and press Next.
3. Create a New Remote Site.
4. Input a name to your liking (for instance Checkstyle Plug-in) and input the following URL: http://eclipse-cs.sourceforge.net/update
5. Click your way through the following pages to install the plug-in.

## Hibernate Tools Eclipse plugins

The following features are available in the Hibernate Tools Eclipse plugins:

Mapping Editor: An editor for Hibernate XML mapping files, supporting auto-completion and syntax highlighting. It also supports semantic auto-completion for class names and property/field names, making it much more versatile than a normal XML editor.

Hibernate Console: The console is a new perspetive in Eclipse. It provides an overview of your Hibernate Console configurations, were also get an interactive view of your persistent classes and their relationships. The console allows you to execute HQL queries against your database and browse the result directly in Eclipse.

Configuration Wizards and Code generation: A set of wizards are provided with the Hibernate Eclipse tools; you can use a wizard to quickly generate common Hibernate configuration (cfg.xml) files, and from these you can code generate a series of various artifacts, there is even support for completely reverse engineer an existing database schema and use the code generation to generate POJO source files and Hibernate mapping files.

(Above content obtained from: http://www.hibernate.org/hib_docs/tools/reference/en/html/)

### Installation

To install Hibernate Tools Eclipse plugin, download it (http://www.hibernate.org/6.html) and extract it to your Eclipse directory and restart Eclipse.

Detailed instructions can be found here: http://www.hibernate.org/255.html

## JavaScript Editor plugin for Eclipse

For information on JSEclipse, go to:
http://www.interaktonline.com/Products/Eclipse/JSEclipse/Overview/

### Installation

1. Close Eclipse.

2. Go to http://www.interaktonline.com/Products/Eclipse/JSEclipse/Try-Download/.
3. Download the latest version of JSEclipse.
4. Unzip the zip file into the directory that holds your plugins directory for Eclipse.
5. Start Eclipse.

Please note that this is a commercial product.

## PMD plugin for Eclipse

PMD scans Java source code and looks for potential problems like:

- Possible bugs - empty try/catch/finally/switch statements
- Dead code - unused local variables, parameters and private methods
- Suboptimal code - wasteful String/StringBuffer usage
- Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
- Duplicate code - copied/pasted code means copied/pasted bugs

(Above content obtained from: http://pmd.sourceforge.net/)

### Installation

1. Start Eclipse.
2. Select *Help* > *Software Updates* > *Find and Install...*
3. Click *Next* >.
4. Click *New Remote Site...*.
5. Enter the name and URL:

Name: **PMD**

URL: **http://pmd.sf.net/eclipse**

6. Click *Finish*.
7. Select the latest version of *PMD for Eclipse*.
8. Click *Next* >.
9. Click *I accept the terms in the license agreements*.
10. Click *Next* >.
11. Click *Finish*.
12. Feature verification (PMD for Eclipse): click *Install*.

### Usage

The PMD plugin allows detect things such as unused variables, empty catch blocks and much more bad coding practices. Each detection is put onto the Todo list, with markers in the code to let you know it should be done.

To run PMD, right-click on a project node, select "PMD" and select one of the task to run.

To run the duplicate code detector, right-click on a project node and select PMD > Find suspect cut and paste. The report will be placed in a "reports" directory in a file called "cpd-report.txt".

## Spring plugin for Eclipse

For a good introduction to the Spring Framework, please read this article:
http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework

### Installation

1. Start Eclipse.
2. Select Help > Software Updates > Find and Install...
3. Click Next.
4. Click New Remote Site....
5. Enter the name and URL:

| | |
|---|---|
| Name: | **Spring** |
| URL: | **http://springide.org/updatesite** |

6. Click Finish.
7. Select latest version of Spring Framework.
8. Select latest version of Spring IDE, Beans Configuration Support.
9. Click Next >.
10. Click I accept the terms in the license agreements.
11. Click Next >.
12. Click Finish.
13. Feature verification (Spring Framework): click Install.
14. Feature verification (Spring IDE - Beans Configuration Support): click Install.
15. Install/Update: click Yes.

## TPTP plugin for Eclipse

The Eclipse Test & Performance Tools Platform (TPTP) Project is an open source Top Level Project of the Eclipse Foundation, and is installed as part of the Callisto plugin install process. TPTP is divided into four projects:

- TPTP Platform: the TPTP Platform Project encompasses a large amount of common infrastructure and capability which the other TPTP projects expand and specialize. It provides common user interface, standard data models, data collection and communications control, as well as remote execution environments
- Monitoring Tools: it addresses the monitoring and logging phases of the application lifecycle.
- Testing Tools: it addresses the testing phase of the application lifecycle.
- Tracing and Profiling Tools: it addresses the tracing and profiling phases of the application lifecycle.

(Above content obtained from: http://www.eclipse.org/tptp/)

### Agent Controller

The standalone Agent Controller is necessary for using the TPTP profiling tool to handle remote profiling.
As of the Callisto Eclipse bundle, TPTP has an integrated agent controller that will work for most people when profiling on a localhost.
However, if you want to install that standalone Agent Controller, you still can.

### Installation

- Download Agent Controller
  Visit the TPTP web site at http://www.eclipse.org/tptp/.
  Find, download, and unzip the agent contoller .zip file into a directory.
- At the bottom of the download page there are links to documentation, including install docs.

## Velocity plugin for Eclipse

Velocity is a Java-based template engine. It permits anyone to use a simple yet powerful template language to reference objects defined in Java code.

When Velocity is used for web development, Web designers can work in parallel with Java programmers to develop web sites according to the Model-View-Controller (MVC) model, meaning that web page designers can focus solely on creating a site that looks good, and programmers can focus solely on writing top-notch code. Velocity separates Java code from the web pages, making the web site more maintainable over its lifespan and providing a viable alternative to Java Server Pages (JSPs) or PHP.

(Above content obtained from: http://jakarta.apache.org/velocity/)

### Installation

1. Start Eclipse.
2. Select Help > Software Updates > Find and Install...
3. Click Next >.
4. Click New Remote Site....
5. Enter the name and URL:

| Name: | **Veloeclipse** |
|---|---|
| URL: | http://propsorter.sourceforge.net/veloeclipse/ |

6. Click Finish.
7. Select the latest version
8. Click Next >.
9. Click I accept the terms in the license agreements.
10. Click Next >.
11. Click Finish.
12. Feature verification: click Install.

### Usage

Any *.vm* file will be opened in the Velocity Editor and allow you to use (limited) code completion. See the website for more details.

## WTP plugin for Eclipse

The Eclipse Web Tools Platform (WTP) project extends the Eclipse platform with tools for developing J2EE Web applications, and is part of the Eclipse Callisto distribution The WTP project includes the following tools: source editors for HTML, JavaScript, CSS, JSP, SQL, XML, DTD, XSD, and WSDL; graphical editors for XSD and WSDL; J2EE project natures, builders, and models and a J2EE navigator; a Web service wizard and explorer, and WS-I Test Tools; and database access and query tools and models.

(Above content obtained from: http://www.eclipse.org/webtools/)

## Amateras XML Editor plugin for Eclipse

Eclipse HTML Editor is an Eclipse plugin for HTML/JSP/XML Editing. It works on Eclipse 3.0 (or higher), JDT and GEF. It has following features.

- HTML/JSP/XML/CSS/DTD/JavaScript Hilighting
- HTML/JSP Preview
- JSP/XML Validation
- Contents Assist (HTML Tags/Attributes, XML based on DTD and JSP taglib and more)
- Wizards for creating HTML/JSP/XML files
- Outline View
- Editor Preferences

- Editor Folding
- Web Browser (It works as an Eclipse's editor)
- Image Viewer
- Tag Palette
- CSS code completion and outline
- DTD code completion, outline and validation
- JavaScript code completion, outline and validation

(Above content obtained from:
http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=EclipseHTMLEditor)

### Installation

1. Close Eclipse.
2. Go to https://sourceforge.jp/projects/amateras/files/.
3. Download the latest version of EclipseHTMLEditor.
4. Unzip the zip file into the directory that holds your plugins directory for Eclipse.
5. Start Eclipse.

### Usage

If you make sure you add a DTD to the XML file then the Amateras XML editor (you may have to use *Open With* to start using it) will create an *Outline* view and enable you to use code completion.

## *BEA Workshop and Eclipse Tips*

- Put your cursor on the interface name or method name, press "CTRL+T", it will show a drop down of all implementations. You can select one you want to go. This is very helpful with all the Spring interfaces going around.

# Other Development Tools

This section describes other tools that may be helpful for developing AquaLogic Commerce Services applications.

## *Web browser tools*

- Web developer extension for FireFox/Mozilla. This plug-in provides a variety of handy features for developing web pages, including W3C compliance checking. http://chrispederick.com/work/webdeveloper/
- Live HTTP Headers. Shows real-time HTTP header data http://livehttpheaders.mozdev.org/

# Creating Eclipse Projects from the Source Distribution

This section explains how to configure your Eclipse or BEA Workshop IDE to prepare for developing customizations to the Commerce Services application.

## *Creating the Eclipse Project*

Take the following steps to create your project in Eclipse.

1. Select File > New > Project.
2. Select "Java Project" and click "Next"
3. Enter a name for the project, e.g. CommerceServices
4. Select "Create project from existing source"
5. Click "Browse" and navigate to this location: "<BEA_HOME>\weblogic92\samples\commerce\commerceApp\commerceServices"
6. Click "Next"
7. In the "Source" tab, right-click WEB-INF/src/test/java under the project you created and select "Remove from build path"
8. Check "Allow output folders for source folders"
9. Click "Finish"

## *Development tips*

- You will likely wish to disable the product cache re-loader for faster server start times. Navigate to <BEA_HOME>\weblogic92\samples\commerce\commerceApp\commerceServices\WEB-INF and open the commerce-config.xml file. Search for "productcache.preload" and set the value to false as shown below.

```
<property name="productcache.preload" value="false"/>
```

- When your development environment is configured as in the "Creating the Eclipse Project" section above, you are developing on the live deployment. If you start the application, you will be able to modify files and observe the effect on the running server immediately.

# 2 - Programming with AquaLogic Commerce Services

This section is intended to help a developer get familiar with the **core concepts** needed to build e-commerce applications on the AquaLogic Commerce Services platform.

The explanations and examples provided in this section are deliberately **brief and concise** because the aim is **to get you productive as quickly as possible** without you having to learn all the details of what's going on under the hood. For more details on the underlying architecture, refer to the Architecture Reference.

# Tutorial 1 - Adding an address (storefront)

In this tutorial we will see how to implement a simple "Add Address" feature to the AquaLogic Commerce Services Storefront. This feature allows customers to create a shipping address from the customer self-service area of the store. We will describe the major steps involved in implementing this feature at each layer of the AquaLogic Commerce Services architecture as well as several key design considerations. This tutorial assumes that there is no address feature in the system.

## *Adding customer addresses to the domain model*

The first step in extending AquaLogic Commerce Services's functionality is to extend the domain model to support the new feature. We will need to add an "Address" Java class to represent a customer's address in the system. The domain object model in AquaLogic Commerce Services is shared by both the Commerce Manager and Storefront web applications. Therefore, the model can be found in the com.elasticpath.core project that both web applications depend on. Domain objects are implemented in the com.elasticpath.domain package. All domain objects in AquaLogic Commerce Services implement an interface and we will add this to the system first. Because addresses are related to customers, we will add a new Address interface to the com.elasticpath.domain.customer package. The address interface will extend the Persistence interface because classes that implement it are intended to be saved to persistent storage. Note that an address could also be considered a ValueObject because it represents a value that does not have its own identity (It must belong to a Customer with an identity). In this case the Address interface would implement ValueObject. However, in this tutorial we will create Addresses as entities as this is the more common domain object type. In this interface we will define accessor methods for each of the data fields of an address such as addressLine1 and phoneNumber. A code snippet from this interface is shown below. Note the javadocs including @param and @return tags. These javadocs are required if you are using the Checkstyle tool to verify your coding style.

```
package com.elasticpath.domain.customer;

import com.elasticpath.domain.Persistence;

/**
```

```
 * <code>Address</code> represents a North American street address.
 */
public interface Address extends Persistence {


        /**
         * Gets the first name associated with this
<code>Address</code>.
         *
         * @return the first name
         */
        String getFirstName();


        /**
         * Sets the first name associated with this
<code>Address</code>.
         *
         * @param firstName the first name.
         */
        void setFirstName(final String firstName);


. . .
```

We can now create the implementation class for the address interface in the corresponding ".impl" package. In our case, this package is com.elasticpath.domain.customer.impl and we will name the class AddressImpl. The "Impl" suffix differentiates the implementation class from the interface. This class extends from AbstractPersistenceImpl, which declares the uidPk database identifier field.

```
package com.elasticpath.domain.customer.impl;


import com.elasticpath.domain.customer.Address;
import com.elasticpath.domain.impl.AbstractPersistenceImpl ;


/**
 * Implementation of a North American street address.
 */
public class AddressImpl extends AbstractPersistenceImpl implements
Address {
        private String firstName;
        private String lastName;
        private String city;
         . . .
```

In AquaLogic Commerce Services, custom non-singleton domain model objects can be created simply by using the `new()` operator. All domain objects will require that the ElasticPath instance is set after instantiation by calling the setElasticPath() method. Alternatively, new objects that are not frequently instantiated can be declared in Spring configuration files and accessed using getElasticPath().getBean("<bean id from Spring config file>").

Built-in domain model objects can be retrieved by invoking the getBean method on ElasticPath and passing in the constant name of the bean as in the following example.

```
 (Address) getElasticPath().getBean(ContextIdNames.ADDRESS);
```

Note that creating instances of built-in domain model objects in this way will not incur the overhead of Spring object creation although there are some exceptions where the object returned is configured in and retrieved from the Spring context.

**Note**

It's important to note that singleton instances of service classes are also retrieved using the getElasticPath().getBean() syntax. However, for these services, the beans are retrieved from the Spring context and the definition appears in a Spring configuration file such as service.xml. The getElaticPath().getBean() method will automatically retrieve beans from the Spring context if they are not pre-defined in ElasticPathImpl.

Since customers will have collections of addresses that they create, we need to extend the CustomerImpl class to declare the collection. The new collection in TutorialCustomerImpl is shown below.

```
public class TutorialCustomerImpl extends CustomerImpl implements
Customer {

  private List addresses;

  /**
   * Gets the <code>CustomerAddress</code>es associated
   * with this <code>Customer</code>.
   *
   * @return the list of addresses.
   */
  public List getAddresses() {
    return addresses;
  }

  /**
   * Sets the <code>CustomerAddress</code>es associated
```

```
   * with this <code>Customer</code>.
   *
   * @param addresses the new list of addresses.
   * @throws EpDomainException if the given <code>List</code>
   * is <code>null</code>.
   */
  public void setAddresses(final List addresses) throws
EpDomainException {
    this.addresses = addresses;
  }


        . . .
```

When adding collections to a domain model, it is often also necessary to update the setDefaultValues() method. We initialize our collection to a new empty ArrayList in this method which can prevent exceptions when unit testing. In general, AquaLogic Commerce Services domain objects initialize collections in this setDefaultValues() method or in an accessor rather than in a constructor or initializer. This limits unnecessary memory overhead when collections are not used. Therefore, any domain model classes you extend should override setDefaultValues(), call super.setDefaultValues(), and then initialize any new collections in the subclass.

## *Unit testing domain model classes*

Domain model classes such as an Address implementation are well covered by JUnit tests. Our new AddressImpl class should have a JUnit test defined in the same package. However, instead of cluttering the implementation package with JUnit tests, unit tests are placed in a parallel package and folder hierarchy that is rooted at the "test" folder instead of the "java" folder. This allows separation of the JUnit tests while still declaring the tests in the same package as the classes being tested so that members with "protected" visibility can be accessed by the tests. All JUnit tests in ElasticPath extend from ElasticPathTestCase. ElasticPathTestCase provides access to pre-configured objects that are useful for testing such as a populated ElasticPath instance. ElasticPathTestCase extends MockObjectTestCase to provide object mocking features.

In addition to adding a new test case for the AddressImpl class, we will also need to update the CustomerImplTest JUnit test case to cover the new code added to CustomerImpl.

## *Persisting address objects*

Now that we've created and tested the domain model, we can connect the Address with the persistence layer. Persistence operations are performed by classes that are typically named XService where X is the name of the class that the service stores and retrieves. For example, the CustomerService is used to add, retrieve, update, and search for Customer objects. In the case of our new address object, there is no need to create a separate service for storing and retrieving addresses because addresses always belong to a single customer. Instead, addresses are added to a Customer object, and the Customer object is persisted by calling CustomerService.update(Customer). This update will "cascade" insert or update the customer's addresses as well.

## Hibernate mapping

In ElasticPath, persistence is accomplished using the Hibernate library which provides a framework for persisting objects in a relational database. For each persistent class, we create a Hibernate mapping file that provides instructions to Hibernate on how the object is to be persisted. Hibernate mapping files for each persistent class can be found in the WEB-INF/conf/hibernate_mapping directory in the core project. Each persistent class typically has its own hibernate mapping file named X.hbm.xml where X is the name of the class being mapped.

The mapping file for our address class is shown below.

The following is the complete mapping file for the new Address class.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class table="TADDRESS" discriminator-value="0"
    name="com.elasticpath.domain.customer.Address">
    <id name="uidPk">
      <column name="UIDPK"/>
      <generator class="hilo">
        <param name="table">HIBERNATE_UNIQUE_KEYS</param>
        <param name="column">value</param>
      </generator>
    </id>
    <discriminator type="integer"
      formula="case when UIDPK &gt; 0 then 1 else 0 end"/>
    <property name="lastName"
      length="100" type="java.lang.String" column="LAST_NAME"/>
    <property name="firstName"
      length="100" type="java.lang.String" column="FIRST_NAME"/>
    <property name="phoneNumber"
      length="50" type="java.lang.String" column="PHONE_NUMBER"/>
    <property name="faxNumber"
      length="50" type="java.lang.String" column="FAX_NUMBER"/>
    <property name="street1"
      length="200" type="java.lang.String" column="STREET_1"/>
    <property name="street2"
      length="200" type="java.lang.String" column="STREET_2"/>
    <property name="city"
```

```
      length="200" type="java.lang.String" column="CITY"/>
    <property name="subCountry"
      length="200" type="java.lang.String" column="SUB_COUNTRY"/>
    <property name="zipOrPostalCode"
      length="50" type="java.lang.String" column="ZIP_POSTAL_CODE"/>
    <property name="country"
      length="200" type="java.lang.String" column="COUNTRY"/>
    <property name="commercialAddress"
      type="java.lang.Boolean" column="COMMERCIAL"/>
    <subclass
name="com.elasticpath.domain.customer.impl.CustomerAddressImpl"
        discriminator-value="1">
      <property name="guid" not-null="true" length="64"
        type="java.lang.String" column="GUID" unique="true"/>
    </subclass>
  </class>
</hibernate-mapping>
```

Because we are now using the extended TutorialCustomer class instead of Customer, we need to rename the customer.hbm.xml hibernate mapping file to tutorialcustomer.hbm.xml. We can then update this file to adapt it to save TutorialCustomers instead of Customers. To make this change, change the name of the implementation class and add the new mapping for the new address collection. This portion of the TutorialCustomer mapping file is shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">


<hibernate-mapping>
  <class table="TCUSTOMER" discriminator-value="0"
     name="com.elasticpath.domain.customer.Customer">
    <id name="uidPk">
      <column name="UIDPK"/>
      <generator class="hilo">
        <param name="table">HIBERNATE_UNIQUE_KEYS</param>
        <param name="column">value</param>
      </generator>
    </id>
    <discriminator type="integer"
      formula="case when UIDPK &gt; 0 then 1 else 0 end"/>
    <property name="userId" index="I_C_USERID" not-null="true"
length="255"
```

```
      type="java.lang.String" column="USER_ID"/>
    <property name="email" index="I_C_EMAIL" not-null="true"
length="255"
      type="java.lang.String" column="EMAIL"/>


...
    <subclass name="com.elasticpath.domain.customer.
          impl.TutorialCustomerImpl"
          discriminator-value="1">

       <set lazy="false" cascade="all, delete-orphan"
           name="addresses"  fetch="join">
         <key column="CUSTOMER_UID" />
         <one-to-many class="com.elasticpath.domain.customer.Address"/>
       </set>

    </subclass>

...
```

## Updating the database schema

The next step in persisting the Address is to update the database schema. If you are using the AquaLogic Commerce Services build system to create database scripts, an ant task is used to translate a database-independent schema defined in XML into database-specific SQL. The database independent schema can be found in database/src/schema.xml in the database project. The table definition for the Address class is as follows.

```
<table name="TADDRESS" description="">
        <column  name="UIDPK" primaryKey="true"
            required="true" type="BIGINT"/>
        <column name="LAST_NAME" size="100" type="VARCHAR"/>
        <column name="FIRST_NAME" size="100" type="VARCHAR"/>
        <column name="PHONE_NUMBER" size="50" type="VARCHAR"/>
        <column name="FAX_NUMBER" size="50" type="VARCHAR"/>
        <column name="STREET_1" size="200" type="VARCHAR"/>
        <column name="STREET_2" size="200" type="VARCHAR"/>
        <column name="CITY" size="200" type="VARCHAR"/>
        <column name="SUB_COUNTRY" size="200" type="VARCHAR"/>
        <column name="ZIP_POSTAL_CODE" size="50" type="VARCHAR"/>
        <column name="COUNTRY" size="200" type="VARCHAR"/>
        <column default="0" name="COMMERCIAL" type="BOOLEANINT"/>
        <column name="GUID" required="true" size="64" type="VARCHAR"/>
```

```
        <column name="CUSTOMER_UID" type="BIGINT"/>
        <foreign-key foreignTable="TCUSTOMER">
            <reference foreign="UIDPK" local="CUSTOMER_UID"/>
        </foreign-key>
               <unique name="TADDRESS_UNIQUE">
                      <unique-column name="GUID"/>
           </unique>
    </table>
```

If you are not using this build process, simply edit the database-specific schema.sql file you are using.

## *Creating the user interface for adding an address*

At this point we have created our Address domain model and added support for persisting addresses. We can now implement a simple page to display an HTML form for the user to input thier address.

In AquaLogic Commerce Services, the Spring MVC framework is used in the presentation layer. The key components are the Velocity templates that render HTML and Spring "Controllers" that mediate between the Velocity UI and the underlying domain model and services. The "form-backing" object is a Java object that Spring populates with the data a user enters into an HTML form before passing the object to the controller for processing. Spring also provides support for validating field input before passing the form-backing object to the controller. Controllers and the URLs that invoke them are configured in a Spring XML file called url-mapping.xml.

### Controller

The first step creating a new page is to create the controller class. Because our controller implements an HTML form, we will extend AbstractEpFormController and call the controller AddressFormControllerImpl. Form controllers are implemented in the com.sfweb.controller.impl package in the StoreFront project (com.elasticpath.sf). In our form controller, we override the onSubmit method to implement the action that should be taken when the user submits the form. Spring will validate the form before this method is invoked, so we can assume that the address information is correct when this method is executed. In this example, we add the address to the customer, persist the customer, and return a new ModelAndView object that displays the success view that is defined in the Spring configuration file.

```
public ModelAndView onSubmit(final HttpServletRequest request,
    final HttpServletResponse response, final Object command,
    final BindException errors) {


    final CustomerAddress address = (CustomerAddress) command;
    final ShoppingCart shoppingCart =
getRequestHelper().getShoppingCart(request);
    final Customer customer =
shoppingCart.getCustomerSession().getCustomer();
```

```
    customerService.update(customer);

    return new ModelAndView(getSuccessView());
}
```

## URL mapping and controller Spring configuration

The controller is configured in the url-mapping.xml file which can be found in the Storefront WEB-INF/conf/spring/web directory. In the SimpleUrlHandlerMapping bean definition in this file, we add the entry below to map the create-address.ep URL to our addressFormController. This will cause the code in AddressFormController to be invoked when users visit the create-address.ep URL.

```
<prop key="/create-address.ep">addressFormController</prop>
```

In the same file, we add the bean definition and configuration for the new AddressFormController as shown in the XML code below.

```
<bean id="addressFormController"

class="com.elasticpath.sfweb.controller.impl.AddressFormControllerImpl"
  parent="abstractEpFormController">

  //Set the name used to reference the form-backing object in the
velocity template
  <property name="commandName">
    <value>address</value>
  </property>

  //Set the form-backing object to be the AddressImpl class
  <property name="commandClass">
    <value>
      com.elasticpath.domain.customer.impl.AddressImpl
    </value>
  </property>

  //Specify the validator for the HTML form fields
  <property name="validator">
    <ref bean="defaultBeanValidator" />
  </property>

  //Wire in the customer service used to persist the address
  <property name="customerService">
```

```
    <ref bean="customerService" />
  </property>

  //refers to a velocity template called create.vm in the address
folder
  <property name="formView">
    <value>address/create</value>
  </property>

  //The page to be redirected to when the form submit succeeds
  <property name="successView">
    <value>redirect:/manage-account.ep</value>
  </property>

</bean>
```

In this bean configuration, we specify that the parent bean is "abstractEpFormController." This causes Spring to apply the configuration required by the AbstractEpFormController class that the AddressFormController extends. The configuration for abstractEpFormController is defined elsewhere in this file. The "commandName" property specifies the name that is used to reference the form backing object in the velocity template. The class of the form backing object is defined in the next property, "commandClass." The bean configuration also wires in a customerService object that will be used to update Customer objects when a new address is added to them. The formView property indicates the name and path of the Velocity template to be used to display the form. The .vm file extension is not specified and paths are relative to the /WEB-INF/templates/velocity directory. The successView property indicates the URL the user should be taken to when the form submission has completed successfully.

## Input Validation

The Spring framework provides a mechanism for validating form input that is based on the Apache Commons Validator. To configure validation for the address form, the validator must be set in the Spring configuration for the form controller as shown below.

```
<bean id="addressFormController"

class="com.elasticpath.sfweb.controller.impl.AddressFormControllerImpl"
  parent="abstractEpFormController">

  . . .

  //Specify the validator for the HTML form fields
  <property name="validator">
    <ref bean="defaultBeanValidator" />
  </property>
```

Next, the validation that should be performed on the form input is specified in the validation.xml file. This file can be found in WEB-INF/conf/misc/validation in the Storefront project. The following code snippet shows how validation is specified for the firstName and country properties. The form name must match the class name of the command object and the field property names match the properties of the command object. The depends="..." block describes the validation constraints that are to be applied to the field. These constraints are defined in validator-rules.xml. Note that the key values such as "globals.firstname" correspond with property file keys that are described further below.

```
<form name="addressImpl">
  <field property="firstName" depends="required,maxlength">
    <arg0 key="globals.firstname" />
    <arg1 name="maxlength" key="${var:maxlength}" resource="false" />
    <var>
      <var-name>maxlength</var-name>
      <var-value>100</var-value>
    </var>
  </field>

  <field property="country" depends="required"><arg0
key="globals.country" /></field>

  <field property="subCountry" depends="subCountryRequired">
    <arg0 key="globals.subcountry" />
    <var>
      <var-name>countryProperty</var-name>
      <var-value>country</var-value>
    </var>
  </field>

  . . .
```

The validation specified in the file as shown above will be executed against the form input before the onSubmit() method is called in the AddressControllerImpl.

> **Note**
>
> If a new validation constraint is required, it can be defined in the validator-rules.xml file and implemented in the EpFieldChecks class.

## Displaying the Add Address form

The view layer in AquaLogic Commerce Services is implemented with Velocity templates. Velocity is a templating language that allows Java objects to be referenced from within an HTML

file. Velocity also supports basic programming constructs such as if statements and loops. Velocity instructions in an HTML file begin with the '#' character. Each page in the Storefront corresponds to a velocity template file. We can create a new Velocity template file to display the address form called create-address.vm in the Storefront WEB-INF/templates/velocity/address directory. The following code shows the velocity template for displaying the form which has been annotated with explanations for key lines.

```
## Declares global variables used in all Storefront templates
## This references a Velocity macro defined in VM_global_library.vm
#templateInit()

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    ## Include the content of head.vm to set the pagetitle
    ## and perform additional configuation
    #parse("includes/head.vm")
  </head>

  <body>
    <div id="wrapper">

      ## Include the customer sign-in status area
      #parse("includes/header.vm")

      ## Include the product category tabs
      #parse("includes/topmenu.vm")

      <div class="top-menu-bar"></div>

      <div class="clear10"></div>
      <div id="body">

        ## Post the form to the form to the new controller
        <form action="$baseUrl/create-address.ep" method="post"
name="address">
          <fieldset>

            <table border="0" cellspacing="0" cellpadding="3">
              <tr>
```

```
              ## The #springMessage macro below retrieves a
              ##localized string from a .properties file
              <td align="right"><label for="address.firstName">
                #springMessage("globals.firstname")*</label></td>
              <td>

                ## Spring macro that displays an input box and
                ## binds it to the form firstName field
                #springFormInput("address.firstName"
"maxlength='100'")

                ## If there are field validation errors, they will be
displayed
                ## here with the "req" CSS class applied
                #springShowErrors("" "req")

              </td>
            </tr>
            <tr>
              <td align="right"><label for="address.lastName">
                #springMessage("globals.lastname")*
              </label></td>
              <td>
                #springFormInput("address.lastName"
"maxlength='100'")
                #springShowErrors("<br>" "req")
              </td>
            </tr>

            ## Remaining fields omitted.

          </table>
        </fieldset>
        <div class="fieldset-footer">
          <input type="submit" class="button-normal"
             value=" #springMessage("address.add") " />
        </div>
      </form>
    </div>
    <div class="clear20"></div>
```

```
    </div>
    #parse("includes/footer.vm")  ## Include the standard footer
  </body>
</html>
```

AquaLogic Commerce Services uses properties files to provide localized strings to be displayed in the user interface. Strings that are used from several velocity templates are defined in globals.properties which can be found in WEB-INF/templates/velocity. For properties that are specific to the new create-address.vm template, we create a new properties file called create-address.properties. A sample in this file are shown below.

```
address.add=Add new Address
```

Each new .properties file must be specified in velocity.xml for it to be accessible from velocity templates. The velocity.xml file can be found in WEB-INF/conf/spring/views/velocity. In the file, add an entry to the "parentMessageSource" bean definition to indicate the location of the new properties file as shown below. Note that the .properties extension is omitted from the file name.

```
<property name="parentMessageSource">
  <ref bean="globalMessageSource"/>
</property>
  <property name="basenames">
    <list>
    . . .
    <value>/WEB-INF/templates/velocity/address/create-address</value>
    . . .
```

## Securing the Add Address page

AquaLogic Commerce Services uses the ACEGI security framework that integrates with Spring for user authentication and permissions. The create address screen should only be accessible to users who are logged into their accounts. We can specify this security constraint in the acegi.xml file found in WEB-INF/conf/spring/security. Open this file and and add the following line to the filterInvocationInterceptor bean definition.

```
\A/create-address.ep\Z=ROLE_CUSTOMER
```

This will ensure that users cannot access the page unless they have been authenticated and have the role, ROLE_CUSTOMER.

This concludes the tutorial which has identified the key steps and considerations in implementing a new Storefront feature. Although many implementation details have been omitted or simplified, many of the features you will implement will follow a similar set of steps.

## How to Get an ElasticPath Instance

There are 2 ways to get an ElasticPath instance:

- Through Spring Bean Injection
  A Spring bean named "elasticPath" is defined. You can inject this bean to other spring beans. This is the preferred way.
- Through static method ElasticPathImpl.getInstance()
  This way is provided to those classes that are not managed by Spring.
  This way should be used as little as possible.

### Getting the WEB-INF path in AquaLogic Commerce Services

Many parts of the AquaLogic Commerce Services system require the full path to the location of the WEB-INF folder so that configuration files can be found wherever the application has been deployed. To get the WEB-INF path in production code, call elasticPath.getWebInfPath(). This will return the path as requested from the ServletContext using getRealPath(), which is implemented by the application server. It is necessary to get the web inf path from this method because the application may be run from a different internal location from where the application is actually deployed. However, the servlet specification does not require that getRealPath() work when the application is running from a WAR file. For example, Weblogic will return null if getRealPath() is called in a WAR'ed application. Therefore, you may need to manually explode a war file during deployment if the application server does not automatically do so.

When running JUnit tests, the web context and getRealPath() are unavailable. In this case, JUnitUtilImpl.getWebInfPath() is called by JUnit test code to get the path to the WEB-INF folder, which will typically be in a different location on each development machine. The JUnit-specific version of getWebInfPath() uses the class loader to find the path. This method should not be called from production code.

# Tutorial 2 - Customizing One Page checkout

In this tutorial we will see how to customize several different aspects of the One Page checkout. We will begin with some simple layout and style changes and move our way up to more complex modifications. Finally, we will walk through an example of creating a new complex form object and configuring it to be automatically rendered by the javascript code.

## *Changing the overall layout*

The layout of One Page is specified in the checkout.vm Velocity template found in the Storefront WEB-INF/templates/velocity/onepage directory. You can change the positions of different sections here such as the arangement of the cart on the left and the checkout accordian on the right as shown below. Try moving some of the sections around to see how this works.

```
<div style="float:left;width:35%;">
  <h2>#springMessage("checkout.heading.cart")</h2>
  <div id="carteditor-container" style="height:350px;overflow-y:auto;">
    <!-- This is rendered in JavaScript by the CartEditor -->
  </div>
  <div id="summary" style="position:relative;height:210px;">
    . . .
  </div>
```

```
</div>

<!-- THE CHECKOUT ACCORDION -->
<div id="checkout_container" style="width:480px; float:right;">
  . . .
</div>
```

All of the forms in the checkout accordian are also specified in this file inside the checkout_container div and can be modified here.

## *Customizing the style of elements*

Most of the styling is done with css in the files found in the Storefront template-resources/stylesheet/onepage directory. The various css files are all concatenated together during the build into one file called onepage.css in the directory just previously mentioned. Most of the files you would want to edit are located in the ep/global and ep/ui subdirectories and are organized by the various functional units they correspond to. For now, locate the onepage.css file and make some simple modifications to the colors and positioning of the various elements.

**Be Careful**

For quick testing you can edit the styles directly in the onepage.css file. Be sure to make any permanent changes to the other files however, otherwise your changes will be overwritten during the next build.

## *Altering the sequence of steps taken during checkout*

The default behavior during the checkout process is to autofill as many of the accordian panes as possible when the customer logs in (or on page load if the customer is already signed in). If the customer then makes changes to the information in one of the panes the pane directly under it will be opened.

This behavior is specified in the checkout.js file found in the template-resources/js/onepage/ep/ui directory. The logic for setting the pane states on page load and on customer login is shown below. To always open the Shipping pane on login or page refresh try commenting out all the code inside the main if block except for the line that reads "this.getShippingPane().open()".

```
ep.ui.Checkout.prototype.refreshPaneStates = function()
{
  var shippingVerified = false;
  var billingVerified = false;

  var customer = ep.session.customer.getCustomer();
  if (customer)
  {
    if (customer.preferredShippingAddress) {
```

```
      shippingVerified = true;
      ep.session.shoppingCart.getCart().selectedShippingAddressUidPk =
        customer.preferredShippingAddress.uidPk;
      this.renderShippingSummary();
    } else {
      this.getShippingPane().open();
    }

    if (customer.preferredBillingAddress) {
      billingVerified = true;
      ep.session.shoppingCart.getCart().selectedBillingAddressUidPk =
        customer.preferredBillingAddress.uidPk;
      this.billingAddressSummary();
    } else if (shippingVerified) {
      this.getBillingPane().open();
    }

    if (shippingVerified && billingVerified) {
      this.showPaymentPane();
    }
  }
};
```

## *Adding a new complex form object*

Complex form elements are specified in their own javascript classes which handle the tasks of rendering, user interaction, and form element updating. In this example we will create a new complex form element for editing a multi-sku cart item. This would assume that we did not already have the MatrixSelector class found in matrixselector.js in the template-resources/js/onepage/ep/ui directory.

We would first create the file called matrixselector.js and define the constructor for the form element as shown below.

```
nitobi.lang.defineNs('ep.ui');


ep.ui.MatrixSelector = function(id, item, options, availability,
choice, skuGuid)
{
  this.item = item;
  this.options = options;
  this.availability = availability;
  this.setId(id);
```

```
  this.skuGuid = skuGuid;
  this.choice = choice || new Array();

  nitobi.event.EventManager.publish('ep.ui.MatrixSelector<[' +
    this.getId()+']>.setChosen');
  nitobi.event.EventManager.subscribe('ep.ui.MatrixSelector<[' +
    this.getId()+']>.setChosen', this, this.userDidChoose);
};

nitobi.lang.extend(ep.ui.MatrixSelector, nitobi.ui.InteractiveElement);
```

The next thing we need to do is create the method that will render the html for the element and setup all the javascript events. The main function for this can be seen below.

```
ep.ui.MatrixSelector.prototype.render = function()
{
  var renderString = "";
  for (var i = 0; i < this.options.length; i++)
  {
    var listId = this.getId() + '_list_'+this.options[i].optionKey;
    renderString +=
      . . .
    for (var j = 0; j < this.options[i].optionValues.length; j++)
    {
      renderString += . . . ;
        . . .
    }
    renderString += . . . ;
  }
  renderString+= . . . ;
  var container = this.getHtmlElementHandle();
  container.innerHTML = renderString;
  this.attachEvents();
  this.updateAvailability();
  this.updateChosen();
  this.updateFormElements();
};
```

A number of other functions to handle the logic controlling user interaction would then need to be written, but we will ignore those for now. To see the full details the matrixselector.js file mentioned above can be referenced.

To have this form element dynamically created we need to tell the form renderer about it. This can be done in the form.js file located in the template-resources/js/onepage/nitobi/ui directory. The method we need to edit is shown below along with the code to use our new form component.

```
nitobi.ui.Form.renderComplexFormElements = function(elements,
arguments)
{
  for (var i=0; i < elements.length; i++)
  {
    if (elements[i].childNodes.length > 0)
    {
      nitobi.ui.Form.renderComplexFormElements(elements[i].childNodes,
arguments);
    }

    if (nitobi.html.Css.hasClass(elements[i], "ep-ui-swatchselector-
container"))
    {
      . . .
    }
    else if (nitobi.html.Css.hasClass(elements[i], "ep-ui-
matrixselector-container"))
    {
      if (!elements[i].javascriptObject)
      {
        matrixSelector = new ep.ui.MatrixSelector(elements[i].id, {}, .
. . );

        elements[i].javascriptObject = matrixSelector;
        matrixSelector.render();
      }
    }
    . . .
  }
 };
```

# Unit Testing

AquaLogic Commerce Services code in the core project is extensively unit tested using the JUnit testing framework. This test coverage provides a quick and effective way to detect bugs that may have been introduced while making modifications. When customizing AquaLogic Commerce Services, it is good practice to write JUnit tests for new or modified code and run all JUnit tests prior to committing changes. The majority of the JUnit tests in AquaLogic Commerce Services cover code in the com.elasticpath.core project, which the web projects depend on. Within the core

project, test coverage by line is approximately 70%. Domain model classes in the com.elasticpath.domain package are particularly well covered.

## *Running JUnit tests*

JUnit tests can be run via an ant task or from within the Eclipse development environment.

### Running tests with Weblogic Workshop or Eclipse

JUnit tests can also be run from within Eclipse. To run an individual test case, right-click on the file in the Package Explorer and select Run as... > JUnit Test. You can also run all tests in an entire project by right-clicking on the project in the Package Explorer and selecting Run As... > Junit Test. JUnit tests run from Eclipse will run several times faster than the ant task, but several unit tests that pass when run by ant will fail in Eclipse. This is due to a JUnit configuration issue and the tests cases will be updated in an upcoming release of AquaLogic Commerce Services so that they will pass using the default Eclipse JUnit configuration.

**Setting memory options in Eclipse**

The default memory for launching applications in Eclipse is insufficient for executing all unit tests in the core project. You can increase the default memory settings by navigating to Window > preferences > Java > Installed JREs > Select your JRE > Edit... and set your memory settings in the "Default VM Arguments" input box. The recommended setting is "-Xmx512m".

## *Creating unit tests*

### Adding a new unit test

The JUnit test case for a class is typically the name of the class being tested with "Test" appended to it. Test cases should be in the same package as the class they test so that they have protected access to the class's members. However, to avoid cluttering the production code with unit test classes, unit tests are placed in separate directory structure that mirrors the "WEB-INF/src" source folder but is rooted at "WEB-INF/test".

Within the test case, a constructor is not required unless there is initialization that must be performed only once for the entire test case. A main method that runs the text UI or Swing UI is also not required as it is not used by the Eclipse JUnit runner the or Ant JUnit task.

Test cases in AquaLogic Commerce Services typically use the setup() method to instantiate the object tested by each test in the test case. The setup() method is also frequently used to configure any mock objects that are not specific to any one test.

### Mocking objects

AquaLogic Commerce Services uses the JMock framework to facilitate testing a single class in isolation. JMock eases the creation of mock objects that can be referenced by the class under test

instead of an instance of the real class used in production. This allows the output of the collaborating class to be controlled in so that specific situations can be tested in the class under test. Furthermore, the mock objects can be given expections of which methods will be invoked on it by the class under test. These expections can optionally specify the parameters that should be passed as well as the number of times the method should be invoked. Since JMock integrates with JUnit, the test case will fail if the mock object's expectations are not met.

The following annotated code shows a typical mock object usage pattern. In this example, the Product class is being tested and we use a mock ProductType object to test Product's interaction with ProductType.

```
//Create a mock object to mock a product type
final Mock mockProductType = mock(ProductType.class);


//Pass the mock object to the product class, casted to a ProductType
this.productImpl.setProductType((ProductType) mockProductType.proxy());


//Specify that the product type's getProductAttributeGroup() method
must
//be called one time and that it will return to the product the set of
//attribute groups we wish to test the product with (returned by the
//createAttributeGroup() method)
mockProductType.expects(once()).method("getProductAttributeGroup").will
(returnValue(createAttributeGroup()));


//Invoke the method being tested, which will interact with the mock
object
this.productImpl.performAttributeRelatedOperation();


//Check the results of the operation here
```

## Mocking ElasticPathImpl using the ElasticPathTestCase base class

Most domain objects and services in the core project maintain a reference to the ElasticPathImpl class and use it to get new instances of domain objects or references to singleton services. For this reason, most test cases in AquaLogic Commerce Services extend from ElasticPathTestCase, which configures an ElasticPathImpl instance that can be passed to the object being tested. To obtain a reference to the configured ElasticPath instance, call setupElasticPath() (typically from the setup() method) and then simply call getElasticPath().

It is very frequently the case that the class you are testing will call getElasticPath().getBean(ContextIdNames.A_BEAN_ID) to retrieve an instance of a domain object. The typical pattern in this case is to create a real or mock object to be returned from the getBean method when the class under test calls ElasticPath.getBean(..). This is configured as follows.

```
//Create the domain object to be returned (not a mock, in this case)
```

```
orderCriterion = new OrderCriterionImpl();

//Indicate that mockWebApplicationContext.getBean(..) will
//return the domain object to be passed to the object
//being tested.
getMockWebApplicationContext().stubs().method(GET_BEAN)
        .with(eq(ContextIdNames.ORDER_CRITERION))
        .will(returnValue(orderCriterion));

//Set the instance of AquaLogic Commerce Services in the object being
tested
orderServiceImpl.setElasticPath(getElasticPath());
```

Note that the getBean() method is mocked on the mock WebApplicationContext object rather than ElasticPathImpl itself. The ElasticPathImpl.getBean() method will invoke the getBean() method of WebApplicationContext and this approach allows both mocked and "Real" ElasticPathImpl instances to be able to return mocked domain model objects.

The ElasticPathTestCase class also contains useful methods that return pre-configured objects such as Products, ShoppingCarts, Money, and PersistenceEngine. Call setupCatalogBeans() before requesting pre-configured catalog objects from ElasticPathTestCase. There are additional similar setup() methods that should be invoked when using ElasticPathTestCase to produce test objects, depending on the objects you require.

Any additional functionality that is used by many test cases can be added to ElasticPathTestCase or a subclass of ElasticPathTestCase that is used as the standard base class for your test cases.

## *Useful resources*

- JUnit: http://www.junit.org/
- JMock: http://www.jmock.org/

## *JMock Tips*

### How to return different instances on subsequent calls to mocked methods

Consider the following case in which a method that returns a Money object has been mocked.

```
mockWebApplicationContext.stubs().method(GET_BEAN_METHOD).with(eq("mone
y")).will(returnValue(new MoneyImpl())
```

In this case, JMock will always return the same Money instance when the getBean("money") method is called. This may not be your intent and in this case returning the same instance of Money in subsequent calls will likely cause your unit test to fail. The following code shows how to make the mocked method return different instances.

```
mockWebApplicationContext.stubs().method(GET_BEAN_METHOD).with(eq("money")).will(
onConsecutiveCalls(returnValue(new MoneyImpl()), returnValue(new MoneyImpl())));
```

```
NOTICE: the onConsecutiveCalls() method can only accept a maximum of 4 return values.
```

The following is an alternative technique that supports returning more than 4 distinct instances.

```
final Stub resultStub = new StubSequence(new Stub[] { returnValue(
 returnValue(new MoneyImpl()),
 returnValue(new MoneyImpl()),
 returnValue(new MoneyImpl()),
 returnValue(new MoneyImpl()),
 returnValue(new MoneyImpl()),
 ....
});
mockWebApplicationContext.stubs().method(GET_BEAN_METHOD).with(eq("money")).will(resultStub);
```

## Primitive types must be specified exactly in jmock constraints

Jmock is very strict regarding types in constraints. For example, the get() method siganature in the PersistencEngine class is declared as shown below:

```
Persistence get(final Class persistenceClass, final long uidPk) throws EpPersistenceException;
```

This method cannot be mocked as shown below because the type of the 1 parameter to eq doesn't match the long type in the method signature.

```
getMockPersistenceEngine().stubs().method("get").with(eq(Brand.class), eq(1)).will(returnValue(brand1));


....


int i=1;
getMockPersistenceEngine().stubs().method("get").with(eq(Brand.class), eq(i)).will(returnValue(brand1));
```

Instead, the method must be mocked with the type explicityly stated as shown below.

```
getMockPersistenceEngine().stubs().method("get").with(eq(Brand.class), eq(1L)).will(returnValue(brand1));


....
```

```
long i=1L;
 getMockPersistenceEngine().stubs().method("get").with(eq(Brand.class)
, eq(i)).will(returnValue(brand1));
```

# Developer Checklists

Here is are some simple development checklists for you to use.

When you change **Java** code:

1. jUnit tests - run using either Ant or Eclipse.
2. Code formatting - run code formatting checker (CTRL+SHIFT+F in Eclipse).
3. checkstyle - run using either Ant or Eclipse.
4. pmd - run using either Ant or Eclipse.
5. Private build testing - to make a further confirmation, start the application your modified and access the feature you coded.
6. Scalability - does your feature scale to handle large numbers of customers, orders, products and SKUs?
7. Domain objects - must be serializable.
8. Acceptance testing - have the "customer" review the feature for completeness and accuracy.

When you have changed **Spring configuration files**, **Hibernate mapping files**, **database schema** or **other configuration** files:

1. Start the application you modified to see if anything is broken.
2. Try to access the feature(s) affected by the file(s) you modified.

# Customizing and Extending AquaLogic Commerce Services Applications

The following sections contain many useful suggestions and best practices for customizing AquaLogic Commerce Services. It is highly recommended that you read these before you start making significant changes.

## *General Customization Practices*

### Making updates simple

In order to avoid having to merge code changes when you receive patches or updates we recommend that you touch as few of the core java classes as possible. The sections that follow contain suggestions for how to go about this.

**Tip**

To avoid confusion you may want to create your extension package structure to mirror the AquaLogic Commerce Services package structure.

**Example:**

AquaLogic Commerce Services class:
com.elasticpath.domain.customer.impl.CustomerImpl.java

your extending class:
com.yourcompany.domain.customer.impl.YourCustomerImpl.java

## Customizing domain objects

Depending on whether you need to customize a domain object application-wide or just certain instances of that class, say inside of a new service method that has been created, you may wish to choose from one of the options below.

**Tip**
Before customizing a domain object, check to see if it allows for localized attributes to be added in the commerce manager. Several of the objects that clients usually wish to extend have already been configured to allow dynamic attribute creation.

### Extending domain objects

If you need to add properties to some of the domain object classes you can do so by extending them with your own java classes. Any accessors or convenience methods that perform some logic that you need to change can be overridden in your extending class. If you find that you need to make extensive changes to a domain object, but wish to maintain the same interface (recommended), then you can create a new implementation of the domain object's interface.

### Wrapping domain objects: the Decorator pattern

To customize a domain object only in certain scenarios while leaving it unchanged in the rest of the application consider using the Decorator pattern. Create a new implementation of the interface that the domain object uses and add a property to it that references the original domain object instance. Implement all the property accessors and other methods required by the interface and have them delegate the call to the reference of the domain object that you have wrapped, adding custom logic where needed. In the service/controller method where you need to use this customized wrapper simply create a new instance of the wrapper object and insert the original domain object into it using the setter method you created. Since the interface is identical between the wrapper and the original object it will be indistinguishable to most of the application.

**Please Note**

If you are dealing with large lists of domain objects that you need to wrap then you may want to extend the object instead. Iterating over the list and wrapping each object may cause a slight performance loss. If you only need to wrap a single object or a small collection of objects then using the Decorator pattern to wrap them should be fine.

## Customizing service methods

There are a number of ways to customize service methods depending on what changes in logic are required.

### Spring hooks: method interception & pre/post execution hooks

If the core logic inside a service method does not need to be altered, but additional routines need to be run before or after the method call then consider using the Spring AOP framework.

With a few changes to the Spring configuration files you can add hooks to an existing service method and perform additional logic before or after the method's execution. If you need access to the parameters passed into the service method you can use a method interceptor. After configuring a proxy class in the Spring configuration file you can have the method call rerouted to your own method, perform the additional logic desired, and then pass the call along to the original method.

**Example:** method interceptor for validating objects passed in as method parameters

**service.xml - adding a validation interceptor**

```
<!--
Note:
the referenced class needs to implement
org.aopalliance.intercept.MethodInterceptor
-->

<bean id="validationInterceptor"
    class="com.elasticpath.service.interceptor.ValidateInterceptor">
  <property name="defaultBeanValidator">
    <ref bean="defaultBeanValidator" />
  </property>
</bean>
```

**service.xml - adding a validation advisor**

```
<bean id="validationAdvisor"
```

```
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
  <property name="advice">
    <ref bean="validationInterceptor" />
  </property>
  <property name="patterns">
    <list>
      <value>.*add.*</value>
      <value>.*update.*</value>
    </list>
  </property>
</bean>
```

**service.xml - adding a proxy wrapper**

```
<bean id="customerAjaxController"
    class="org.springframework.aop.framework.ProxyFactoryBean">
  <property name="proxyInterfaces">

<value>com.elasticpath.sfweb.ajaxservice.CustomerAjaxController</valu
e>
  </property>
  <property name="interceptorNames">
    <list>
      <value>validationAdvisor</value>
      <value>customerAjaxControllerTarget</value>
    </list>
  </property>
</bean>
```

## Extending service classes

In order to change the internal logic of a service method you can extend the service class with your own. If you need to change the logic of a significant number of methods in a service class then you can create a new implementation of the service class's interface. After doing this edit the Spring service configuration files to point to your newly created service instead of the original one.

## *Customizing the Storefront*

### Customizing the UI

AquaLogic Commerce Services has been built to allow maximum flexibility in customizing the look and feel of the store front.

If you do not need to customize any core logic then you will probably only need to edit the velocity and css files. Velocity itself handles most of the display logic and quite a lot of customization can be done just using the velocity macros. All of the html styling is handled in css and can be easily customized by editing the various css files.

### Customizing the checkout process

The default checkout process is handled by a collection of Spring controllers, form beans, and configuration files. In general, the service layer methods should not need to be edited. If, however, you need to alter service layer behavior then see the section General customization practices for suggestions about how to go about doing this.

#### Maintaining the default checkout sequence

If the logical sequence of checkout steps is being maintained then you can probably just copy the default checkout-related spring controllers into your own package directory and alter them as required. If the UI for the checkout process is being customized as well then the velocity templates linked to the controllers in the spring configuration files will need to be edited as well. For any controllers that have been copied you will need to update the spring url-mapping configuration file to use your newly created controller classes instead of the default ones.

#### Changing the checkout sequence

If you need to change the sequence of steps taken through the checkout process you will have to do a fair bit more work. You still shouldn't have to change the service methods, but you will need to create your own velocity templates and spring controllers. You will also need to edit the Spring url-mapping configuration file to link all these up.

Regardless of what you need to do, you can probably save a fair bit of time looking at the default checkout velocity templates and spring controllers.

## *Example – Customizing domain objects through extension*

By extending the implementation of existing domain object, we can customize the implementation of existing domain objects to change business logic or to add new information.

By using the customized domain objects, the Storefront can be customized where needed by creating and using instances of customized classes.

Here we add nicknames to CustomerAddress by extending the existing implementation. To integrate this customized domain object in our application, we will need to update our Spring controllers where needed, our Form beans to use the new class, our view to display the

nicknames, the Hibernate mapping files to persist our new object, and the database schema where needed.

Our extended implementation of CustomerAddress is shown below.

**ExtendedCustomerAddressImpl.java**

```
public class ExtendedCustomerAddressImpl extends CustomerAddressImpl {
        private String nickName;
        public static final long serialVersionUID = 5000000001L;
        public String getNickName() {
                return nickName;
        }
        public void setNickName(String nickName) {
                this.nickName = nickName;
        }
}
```

We will define this new class in our Spring configuration file for domain objects.

**domainModel.xml**

```
<bean id="extendedAddress"
class="com.elasticpath.domain.customer.ExtendedCustomerAddressImpl"
                scope="prototype"
                parent="epDomain">
</bean>
```

In controller classes all we need to do is use the ElasticPath bean to get a new instance of our new class.

**AddressFormControllerImpl.java**

```
public Object formBackingObject(final HttpServletRequest request) {
//NOPMD
        //get the formbean
        final CustomerAddressFormBean customerAddressFormBean =
(CustomerAddressFormBean) getElasticPath().getBean(

        ContextIdNames.CUSTOMER_ADDRESS_FORM_BEAN);
...
        final ExtendedCustomerAddress address =
        (ExtendedCustomerAddressImpl)
        getElasticPath().getBean("extendedAddress");
...
        customerAddressFormBean.setCustomerAddress(address);
        return customerAddressFormBean;
```

```
}
```

We will need to make the form bean aware of the new object by updating its getters and setters to use ExtendedCustomerAddressImpl.

**CustomerAddressFormBean.java**

```
public interface CustomerAddressFormBean extends EpFormBean {
       ExtendedCustomerAddressImpl getCustomerAddress();
```

The new field can now be accessed by our view (Velocity templates) by

```
 "customerAddressFormBean.customerAddress.nickName"
```

To persist our custom class data, we will need to make the necessary database schema changes, and update the Hibernate mapping files.

Since we extended the CustomerAddress class, we need to add a definition to the CustomerAddress.hbm.xml to persist the extra fields in our subclass.

**CustomerAddress.hbm.xml**

```
    <subclass
name="com.elasticpath.domain.customer.ExtendedCustomerAddress"
discriminator- value="1">
      <property name="nickName" length="200" type="java.lang.String"
column="NICK_NAME"/>
      <property name="guid" not-null="true" length="64"
type="java.lang.String" column="GUID" unique="true"/>
    </subclass>
```

Hibernate will now use this mapping where ever an instance of ExtendedCustomerAddress is used and persist our nickname.

### *Example – Customizing domain objects through decoration*

An alternative to extension is to use the decorator pattern to wrap existing domain objects in a custom domain object and add the additional behavior we require.

This method requires our custom object to implement all methods of the component domain object, and delegate method calls to it. To conform with the persistence layer, we will need to implement a few getters and setters as well.

We can decorate the CustomerAddress class by creating a DecoratedCustomerAddress class that implements CustomerAddress.

We delegate to the component all the methods we don't need to customize. Note that we can customize the implementation of any existing CustomerAddress method in this class.

**DecoratedCustomerAddress.java**

```
public abstract class DecoratedCustomerAddress implements
CustomerAddress {
        protected CustomerAddress customerAddress;
        private long uidPk;
        public CustomerAddress getCustomerAddress() {
                return customerAddress;
        }

        public void setCustomerAddress(CustomerAddress address) {
                this.customerAddress = address;
        }
        public DecoratedCustomerAddress(CustomerAddress address){
                this.customerAddress = address;
        }
        public DecoratedCustomerAddress(){
                super();
        }
        public String getGuid() {
                return customerAddress.getGuid();
        }

...Delegate all getters/setters to component CustomerAddress...

        public ElasticPath getElasticPath() {
                return customerAddress.getElasticPath();
        }
        public void setElasticPath(ElasticPath arg0) {
                this.customerAddress.setElasticPath(arg0);
        }
        public long getUidPk() {
                return this.uidPk;
        }
        public boolean isPersistent() {
                return this.uidPk > 0;
        }
        public void setDefaultValues() {
                customerAddress.setDefaultValues();
        }
```

```
        public void setUidPk(long arg0) {
                this.uidPk = arg0;
        }
}
```

This class is intentionally abstract, so that we can decorate the CustomerAddress class with a subclass. Here we decorate it with a nickname field.

**NicknameDecoratedCustomerAddress.java**

```
public class NicknameDecoratedCustomerAddress extends
DecoratedCustomerAddress {

        private String nickName;

        public static final long serialVersionUID = 5000000001L;

        public String getNickname() {
                return nickName;
        }

        public void setNickname(String nickName) {
                this.nickName = nickName;
        }

        public NicknameDecoratedCustomerAddress(
                CustomerAddress address){
                super(address);
        }

        public NicknameDecoratedCustomerAddress(){
                super();
        }

}
```

We will need a new Hibernate mapping file to persist our DecoratedCustomerAddress, which will also include our component CustomerAddress.

Our new DecoratedCustomerAddress will still be in the TADDRESS table, which can be modified to accomodate the nicknames, or a new table can be created.

**DecoratedCustomerAddress.hbm.xml**

```
<hibernate-mapping>
  <class table="TADDRESS" discriminator-value="0"
name="com.elasticpath.domain.customer.DecoratedCustomerAddress">
    <id name="uidPk">
      <column name="UIDPK"/>
      <generator class="hilo">
        <param name="table">HIBERNATE_UNIQUE_KEYS</param>
        <param name="column">value</param>
      </generator>
    </id>

    <discriminator type="integer" formula="1"/>
    <component name="customerAddress"
class="com.elasticpath.domain.customer.impl.CustomerAddressImpl">
           <property name="lastName" length="100"
type="java.lang.String" column="LAST_NAME"/>
      ...
           <property name="commercialAddress" type="java.lang.Boolean"
column="COMMERCIAL"/>
           <property name="guid" not-null="true" length="64"
type="java.lang.String" column="GUID" unique="true"/>
       </component>

          <subclass
name="com.elasticpath.domain.customer.NicknameDecoratedCustomerAddress"
discriminator-value="1">
      <property name="nickName" type="java.lang.String"
column="NICKNAME"/>
     </subclass>
  </class>
</hibernate-mapping>
```

For CustomerAddresses to be persisted with decorated addresses, we need to update the Customer object's hibernate mapping file to use the new mapping file when updating its addresses.

**Customer.hbm.xml**

```
<bag lazy="false" cascade="all, delete-orphan" name="addresses"
fetch="join">
      <key column="CUSTOMER_UID" />
      <one-to-many
class="com.elasticpath.domain.customer.DecoratedCustomerAddress"/>
    </bag>
```

When it comes time to use the customized domain object, say when creating new Addresses for customers, we can modify the AddressFormControllerImpl in the Storefront to decorate our CustomerAddresses.

**AddressFormControllerImpl.java**

```
public Object formBackingObject(final HttpServletRequest request) {
//NOPMD
        //get the formbean
        final CustomerAddressFormBean customerAddressFormBean =                (CustomerAd
        ContextIdNames.CUSTOMER_ADDRESS_FORM_BEAN);
...
        final CustomerAddress componentAddress = (CustomerAddress)
        getElasticPath().getBean(ContextIdNames.CUSTOMER_ADDRESS);
        final NicknameDecoratedCustomerAddress address = new
        NicknameDecoratedCustomerAddress(componentAddress);
...
```

Our form bean will also need to be made aware of the new custom object so that our view will be able to retrieve the new field(s).

**CustomerAddressFormBean.java**

```
public interface CustomerAddressFormBean extends EpFormBean {
        NicknameDecoratedCustomerAddress getCustomerAddress();
```

Our velocity templates can now reference the nickname through

```
 "customerAddressFormBean.customerAddress.nickName"
```

## *Example – Creating a new service*

We can create a new service to get a list of customer address nicknames since the previous code did not know anything about our customizations. This service can be used to perform the simple function of displaying the list of address nicknames that a user has.

First we define the new service interface and implementation, which needs a CustomerService to be injected later.

**NicknameService.java**

```
public interface NicknameService {
        List getCustomerNicknames(long customerUid);
}
```

And the implementation.

**NicknameServiceImpl**

```
public class NicknameServiceImpl implements NicknameService{

	private CustomerService customerService;

	public void setCustomerService(CustomerService service){
		this.customerService = service;
	}

	public CustomerService getCustomerService(){
		return customerService;
	}

	public List getCustomerNicknames(long customerUid) {
		Customer customer = customerService.get(customerUid);
		List addresses = customer.getAddresses();
		List nickNames = new ArrayList();
		ExtendedCustomerAddressImpl nicknamedAddress = null;
		for(Iterator i = addresses.iterator(); i.hasNext();) {
			Object o = i.next();
			if(o instanceof ExtendedCustomerAddressImpl) {
				nicknamedAddress =
(ExtendedCustomerAddressImpl) o;
			}
			nickNames.add(nicknamedAddress.getNickName());
		}
		return nickNames;
	}
}
```

In our web Spring config file we can add an entry to the new NicknameService, while injecting in the CustomerService. NicknameService can then be injected into our controllers to be used.

Here we inject it into the manageAccountController bean.

**url-mapping.xml**

```
<bean id="nicknameService"
	class="com.elasticpath.service.impl.NicknameServiceImpl">
	<property name="customerService">
```

```
                <ref bean="customerService" />
        </property>
</bean>
<bean id="manageAccountController"
        class="com.elasticpath.sfweb.controller.impl.
                                ManageAccountControllerImpl"
        parent="abstractEpController">
        <property name="customerService">
                <ref bean="customerService" />
        </property>
        <property name="nicknameService">
                <ref bean="nicknameService"/>
        </property>
        <property name="orderService">
                <ref bean="orderService" />
        </property>
        <property name="unauthorizedView">
                <value>redirect:/sign-in.ep</value>
        </property>
        <property name="successView">
                <value>account/manage</value>
        </property>
</bean>
```

We'll want to display this list of nicknames on the manage customer account page, so we modify the Controller ManageAccountControllerImpl to use the new service.

**ManageAccountControllerImpl.java**

```
private NicknameService nicknameService;
protected ModelAndView handleRequestInternal(final HttpServletRequest
request, final HttpServletResponse response) throws Exception {
        if (LOG.isDebugEnabled()) {
                LOG.debug("entering 'handleRequest' method...");
        }

        Customer customer;
...
        Map domainModelMap = new HashMap();
        List nickNames =
this.nicknameService.getCustomerNicknames(customer.getUidPk());
        domainModelMap.put("customer", customer);
```

```
        domainModelMap.put("nicknames", nickNames);
        domainModelMap.put("orders",
orderService.findOrderByCustomerGuid(customer.getGuid(), true));

        return new ModelAndView(getSuccessView(), domainModelMap);
}

public NicknameService getNicknameService() {
        return this.nicknameService;
}

public void setNicknameService(final NicknameService service) {
        this.nicknameService = service;
}
```

Now the list of nicknames are available to our view through the session domain map's "nicknames" property.

A simple example to display the list of nicknames in a table:

**manage.vm**

```
...
#set ($customer = $!sesShoppingCart.customerSession.customer)
#set ($nicknames = $!sesShoppingCart.customerSession.nicknames)
<div class="title">#springMessage("manage.name")</div>
<div class="value">$!customer.getFirstName()
$!customer.getLastName()</div>
<div class="title">#springMessage("manage.email")</div>
...
#foreach ($nick in $nicknames)
<tr>
  <td>$!nick</td>
</tr>
#end
```

# Useful Technology Tutorials and Resources

AquaLogic Commerce Services is built on a large list of open source frameworks and in order to develop on the AquaLogic Commerce Services 5 platform you should be familiar with the core frameworks. This Developer Guide provides some training on these frameworks, but for more detailed information, we recommend the resources below.

## *Core 3rd-party frameworks*

- Velocity: http://jakarta.apache.org/velocity/
- Spring: http://www.springframework.org/
    - including: Spring MVC, ACEGI (security) and Commons Validator integration
    - recommended book: "Spring in Action"
    - Intro to the Spring Framework - May 2005 (Intro level) - http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework
    - The Spring Series (Part 1-4) - An introduction to Spring, Spring using Hibernate persistence, Spring MVC and Spring messaging. - June 2005 - (Intro to Intermediate level) - http://www-128.ibm.com/developerworks/library/wa-spring1/
- Hibernate: http://hibernate.org/
    - recommended book: "Hibernate in Action"

## *Other 3rd-party frameworks*

- ACEGI Security Solution - http://acegisecurity.org/;
- AJAX technologies used:
    - JavaScript
    - Dojo (JavaScript widget library): http://dojotoolkit.org/
    - DWR (Java remoting): http://getahead.ltd.uk/dwr
    - recommended book: "Ajax in Action"
- Ant (build tool): http://ant.apache.org/
- Axis (Web Services): http://ws.apache.org/axis/
- Java Advanced Imaging API (JAI): http://java.sun.com/products/java-media/jai/iio.html
    - used for dynamic image resizing
- Jboss Rules (rules engine): http://www.jboss.com/products/rules
- jUnit (unit testing): http://www.junit.org
    - recommended book: "jUnit in Action"
    - also recommend jMock for mock objects: http://www.jmock.org/
- Log4J (logging): http://logging.apache.org/log4j/docs/
- Lucene (search engine): http://lucene.apache.org/java/docs

    - recommended book: "Lucene in Action"
- Quartz (scheduler): http://www.opensymphony.com/quartz/
- Xdoclet (annotations): http://xdoclet.sourceforge.net/xdoclet/index.html
    - recommended book: "Xdoclet in Action"

**Note**: We don't get any kickback from Manning Publications. We just like their books. ☺

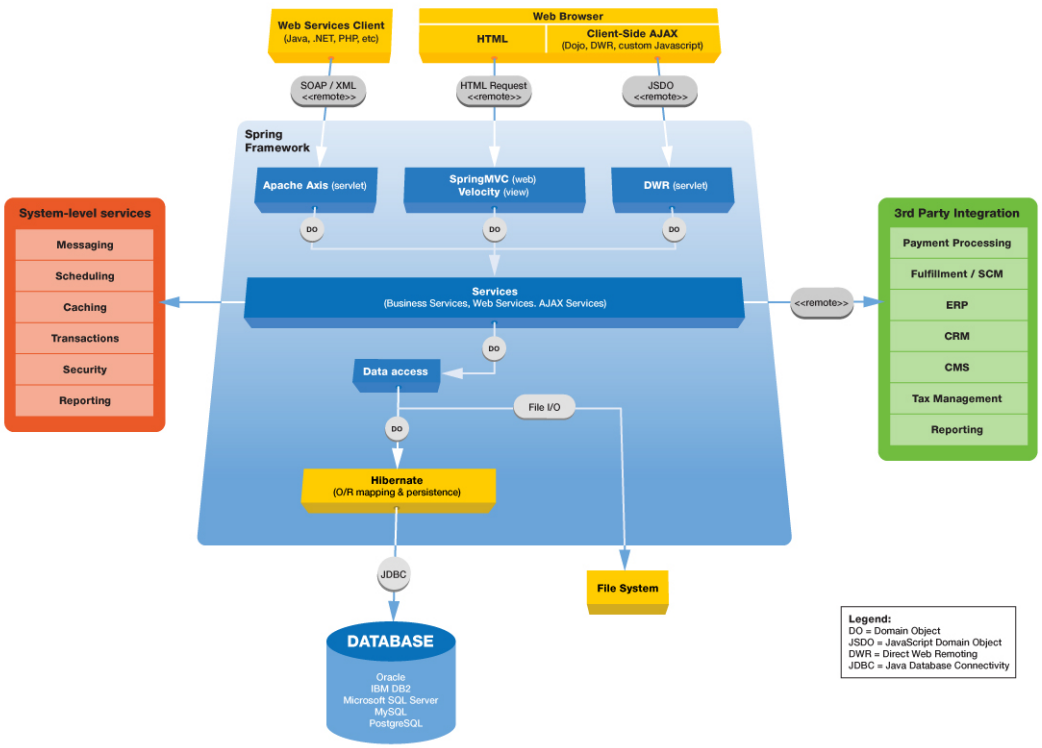
# 3 – Architecture Reference

The purpose of this section is to explain the underlying architectural aspects of AquaLogic Commerce Services that apply to a wide range of features. For a quick introduction to programming with AquaLogic Commerce Services, see the “Programming with AquaLogic Commerce Services” section.

## Application Layers

At a high-level, AquaLogic Commerce Services can be viewed as a set of layers with the data access layer at the bottom and the view layer at the top. This section describes the key design concepts and technologies used in each layer.

The architecture diagram below presents the key components of the system, including the technologies used. The details of these components and how to work with them are described in this section. Click on the thumbnail below to see the full-sized architecture diagram.



## *View*

The view layer is responsible for presenting content to the user in the browser. The view layer also collects user input to be processed by lower layers of the architecture. The view layer is composed of the following technologies.

- Velocity - A template processing technology that embeds information from Java objects within HTML pages
- CSS - Cascading Style Sheets provide formatting and positioning for elements across all HTML pages
- Dojo - A JavaScript library used to create rich user interfaces in the browser

## Velocity Templating Engine

Velocity is a template engine that serves as an alternative to JSP that separates Java from the presentation tier. When working with Velocity, you will typically begin with a static HTML page and then add Velocity directives (or "code") to access and display the properties of Java objects. Files containing static text and Velocity code are called templates and have a .vm file extension. A Velocity engine is invoked to process the template, rendering pure text without any Velocity code. In addition to the UI layer, Velocity is used in AquaLogic Commerce Services to generate email messages and generate configuration files during the build process.

### Syntax reference

The following subset of Velocity directives demonstrates the key funtionality provided by Velocity and also serves as a quick reference.

- $!customer.address – Displays the address property of the customer object
- $!customer.getAddress() – Displays the result of invoking the getAddress() method on the customer object
- #if [#elseif] [#else] #end – Synax for Conditionals
- #foreach( $ref in arg ) statement #end – Syntax for iteration (For Each loops)
- #set($variable = "value") – Sets the value of a variable. The variable does not need to be declared
- #include - Renders files that are not parsed by Velocity
- #parse - Renders files that are parsed by Velocity
- #macro - Runs a Velocity macro

### Specifying paths and URLs in Velocity templates

The AquaLogic Commerce Services Search Engine Optimization (SEO) feature rewrites the URL of some pages in the StoreFront. For this reason, pages using SEO cannot use paths relative the the page's path. In this case, you must use the absolute path to images or URLs. By including the #templateInit() directive at the top of your page, a Velocity variable namved $baseUrl will

contain the base portion of the absolute path you will need. For example, use syntax below to link to index.ep.

```
<a href="$baseUrl/index.ep">#springMessage("bc.home")</a>
```

### Best practices

- Presentation logic of any significant complexity should be invoked from a Velocity macro
- Velocity templates should be formatted so that they are easy to read
- A Velocity template should consist of mostly HTML sprinkled with velocity macros and functions
- Use parameters to help abstract logic within a macro
- Templates should NOT contain business logic
- Most macros should be in the global macro library
- Only macros that are highly-specific to a particular page should be defined at the top of the page

There is a flag called "autoload" that determines whether Velocity will automatically reload global library macros. During development, this should be set to "on" so that you do not need to restart your app server when making a change to the global Velocity macro library.

## Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) is a mechanism for separating the style of a web page (fonts, colors, spacing, positioning, etc.) from the content of the page. When using CSS, a CSS file defines the style for various HTML elements such as hypertext links. CSS files also define "classes" which define style information for arbitrary HTML elements. In this case, elements in the HTML declare the name of their CSS class to have that particular style applied to them. CSS class names should refer to what the HTML element is (e.g. Order Summary) rather than suggest the style that should be applied to it.

By using CSS, style information can be changed in a single CSS file and it will automatically "cascade" across all HTML pages on the site.

### CSS in AquaLogic Commerce Services

The main CSS file that defines various styles in AquaLogic Commerce Services is called master.css and can be found in template-resources/stylesheet.

## Dojo – JavaScript UI Toolkit

Dojo is the Open Source JavaScript toolkit for developing rich user interfaces that run in the web browser. A major part of Dojo is a collection of user interface widgets that can be assembled to create sophisticated interactive screens. In this section we describe some Dojo fundamentals and patterns of Dojo usage in AquaLogic Commerce Services.

## Dojo Libraries

Dojo includes many core libraries that are useful for many aspects of web development. The following list of libraries provides a high-level overview of the functionality provided by Dojo.

- dojo.event: Browser compatible event system.
- dojo.widget: Build reusable widgets.
- dojo.lang: Utility routines to make JavaScripts easier to use.
- dojo.string: String manipulation routines.
- dojo.dom: DOM manipulation routines.
- dojo.style: CSS style manipulation routines.
- dojo.html: HTML specific operations.
- dojo.collections: Provides ArrayList, Queue, SortedList, Set and Stack manipulations.
- dojo.graphics: Support for nifty HTML effects.
- dojo.dnd: Drag and Drop support.
- dojo.animation: Create animation effects.
- dojo.storage: Flash storage API
- dojo.xml: XML parsing

## Using Topics for anonymous communication

JavaScript objects can communicate with each other using a publish/subscribe event mechanism. Events that can be published must first be registered. Then any object can publish or listen to the event. Events can optionally be published with one or more object references that become available as parameters to methods that are invoked as a result of subscribing to the event. The syntax below shows how events are used.

```
//Register the name of an event
dojo.event.topic.registerPublisher("onCustomerEditStart");

//Publish an event, in this case an optional object
//reference is passed with the event
dojo.event.topic.publish("onCustomerEditStart", this.customer);

//Subscribe to the event "onCustomerEdit" and call
//the "customerEdit" method on "this."
 dojo.event.topic.subscribe("onCustomerEdit",this,"customerEdit");
```

## Connecting Dojo events

Dojo supports a construct similar to Aspect-Oriented Programming (AOP) which allows events to be connected with other events.

The dojo.event.connect() function connects any DOM event or any function with another function that can be run before or after the first event or function.

```
Example1:
   dojo.event.connect(this.changePasswordButton,
     "onClick",  this, "onChangePassword");

Example 2:
  var exampleObj = {counter: 0,
                    foo: function(){
                          alert("foo");
                              this.counter++;
                          },
                         bar: function(){
                           alert("bar");
                              this.counter++;
                          }
          };
  To make sure exampleObj.bar() gets called whenever exampleObj.foo()
is called:
  dojo.event.connect(exampleObj, "foo", exampleObj, "bar");
```

- Connect can handle multiple listeners (called in the order they are registered)
- Here's how to make sure only to fire once even if the same connect method is called multiple times:

```
    dojo.event.kwConnect({
        srcObj: this.dateOfBirthPicker,
        srcFunc:"onSetDate",
        targetObj: this,
        targetFunc:"setDateOfBirthField",
        once:true
       });
```

- Use advice (like AOP, it supports before advice, after advice, and around advice) to ensure that "bar" gets alerted before "foo" when exampleObj.foo() is called:

```
       dojo.event.connect("before", exampleObj, "foo", exampleObj,
       "bar");
```

## AquaLogic Commerce Services integration with Dojo

- Dojo source code is found in [appRoot]/templates-resources/js/dojo
- The Ant task 'dojo' has been provided to unzip the dojo src from the EP_LIB dir and overwrite certain files with AquaLogic Commerce Services customizations.
- dojo.js is a pre-built "profile". elasticpath.js is the AquaLogic Commerce Services customized "profile".

- util.js contains some common utilities functions, i.e. trim(), showDate().

### How to add Dojo functionality to a new page

The following code snippet shows the Dojo-related statements required to add Dojo functionality to a page.

```
<script type="text/javascript">
var djConfig = {isDebug: true, parseWidgets: false, searchIds: [] };
</script>
<script type="text/javascript" xsrc="template-
resources/js/dojo/dojo.js">
</script>
<script type="text/javascript" xsrc="template-
resources/js/elasticpath.js">
</script>
<script type="text/javascript" xsrc="template-
resources/js/elasticpath/util.js">
</script>
...
<script>
...
djConfig.searchIds.push("customerDetailTab");
djConfig.searchIds.push("changePasswordTab");
...
</script>
```

- Set isDebug to **false** for production deployment.
- Set parseWidgets to **false** and register node ids for widget node on page.

### Commerce Manager JavaScript design components

The following diagram shows the key components of the Dojo user interface mechanism behind most tabs in the Commerce Manager. The diagram also includes the relationship between these components and several components on the server.

Browser-Side Components

- Velocity Template - The Velocity template becomes the initial HTML page that is displayed to the user. Each tab in the Commerce Manager typically has only one Velocity template associated with it. The Velocity template declares the Dojo widgets that will appear on it.
- Widget - The JavaScript files that implement the behavior of the Dojo widgets. Each Dojo widget has an HTML template. Widgets communicate with other widgets and the Controller using event messages.
- Template - The HTML templates define the initial appearance of the Dojo widgets.

- Controller - The Controller is a JavaScript file whose primary responsibility is to communicate with the server on behalf of the Dojo widgets. Each page has a single controller. Communication with the server is accomplished through calls to JavaScript proxy objects created by DWR.
- Loader - The Loader is responsible for loading required dojo libraries, utility objects, and defining constants. There is one Loader per page.
- Kernel - The Kernel is a reference to the Controller on which constants and other values are defined. There is one Kernel per page.

Server-Side Components

- Service - Singleton services in the domain layer are often invoked through the browser using DWR.
- AJAX Service - When a page requires information that is not sufficiently generic to be implemented in a Service, an AJAX Service will provide this functionality and expose it through DWR.
- AJAX Bean - When a bean to be sent to the browser requires additional information, it is wrapped in an AJAX Bean that contains more data.

## How to create a custom widget

Dojo widgets consist of an HTML template that defines its initial appearance and a corresponding JavaScript file that defines its behavior. To create a new widget, you will need to create both of these files.

## Create the JavaScript file for the widget

Add a new JavaScript files to a subfolder in template-resources/js/elasticpath/widget/, e.g., csr/CustomerDetailTab.js. The following code is for a UI widget that collects information about a customer. It is annotated to explain key constructs.

**CustomerDetailTab.js**

```
// tell the package system what classes get defined here
dojo.provide("elasticpath.widget.csr.CustomerDetailTab");

// load dependencies
dojo.require("dojo.widget.HtmlWidget");
...

// constructor
elasticpath.widget.CustomerDetailTab= function(){
    // inheritance: call superclass's constructor
    dojo.widget.HtmlWidget.call(this);
```

```
    this.widgetType="CustomerDetailTab";

    // declare locale properties
    this.customer="";
    ...

    // declare the template to be used for this widget
    this.templatePath =

dojo.uri.dojoUri("../elasticpath/widget/templates/csr/CustomerDetailT
ab.html");

    // could also add a templateCSSPath
    // this.templateCSSPath = dojo.uri.dojoUri("xxx/xxx.css");

    // register for listening to certain events

dojo.event.topic.subscribe("onCustomerCreate",this,"onCustomerCreateS
tart");
    ...
}

// complete the inheritance process
dojo.inherits(elasticpath.widget.CustomerDetailTab,
dojo.widget.HtmlWidget);

// could call superclass's function by
//
elasticpath.widget.CustomerDetailPane.superclass.\[function\].call(pa
rams);

// extend the functionality of this widget
dojo.lang.extend(elasticpath.widget.CustomerDetailTab, {
    // The postCreate function will be called right after the widget
is created.
    postCreate: function() {
        this.dateOfBirthNameNode.innerHTML += " (" +
EP.UTIL.dateFormat + ")";

        ...
    },
    onCustomerCreateStart: function(customer) {
        ...
    },
```

```
    ...
});

// register this widget with Dojo's parser so that it
// can find the widget automatically
dojo.widget.tags.addParseTreeHandler("dojo:CustomerDetailTab");
```

### Create the HTML template file for the widget

For each Dojo widget, you will need to create the HTML template. The template defines the initial appearance of the Widget and declares integration points with the JavaScript file. The following two integration points are frequently used.

- DojoAttachPoint - Used to create a reference to a DOM node that can be conveniently accessed through JavaScript.

```
<!-- Attach point in an HTML file -->
<div dojoAttachPoint="commitErrorNode"></div>

//Referencing an attach point in a JavaScript file
    this.commitErrorNode.innerHTML="Validation Error";
```

- DojoAttachEvent - Used to attach events that will invoke functions in JavaScript.

```
<!-- Clicking on this save button will call
     the "onAttributeSave" JavaScript function -->
<input type="button" class="btn"
        dojoAttachEvent="onClick:onAttributeSave" value="Save"
/>
```

Dojo currently requires that templates define a single top-level DOM node which gets assigned to the domNode property of the widget.

If multiple elements appear, only the first one will be used. Currently, you can not declare widgets in the template. The workaround for this is to create the widget in the postCreate function, attaching it to a dojoAttachPoint in the template.

**Do not declare functions in constructors**

Do not declare functions inside the constructor. For every instance of the object created, a new function will be created, which may cause a memory leak because it can easily create a closures that cannot be garbage-collected.

### Register the widget

New widgets that you create must be registered in the *_package_*.js file in the directory where the widget JavaScript file is created. Use the following syntax to register your widget.

***_package_*.js**

```
dojo.kwCompoundRequire({
```

```
 browser: [PD:
 ...
 "elasticpath.widget.CustomerDetailTab",
 ...
 ]
});
dojo.hostenv.startPackage("elasticpath.widget.csr.*");
```

Hostenv is an array of modules relevant to the particular host env, including common, browser, rhino and etc);

### Load the package in the loader file

For each page that uses your widget, find the corresponding loader JavaScript file and check that your package and widget are being loaded. In most cases you will be adding to an existing page with a loader and will not need to modify this file.

**csrLoader.js**

```
...
//elasticpath widgets
dojo.hostenv.setModulePrefix('elasticpath', '../elasticpath/');
dojo.widget.registerWidgetPackage("elasticpath.widget");
dojo.require("elasticpath.widget.csr.*");
...
```

### Additional widget considerations

Break widgets into subdirectories. Leaving widgets in template-resources/js/elasticpath/widget folder with a flat structure makes it hard to maintain. As shown in the previous examples, we put CustomerDetailTab widget into the csr subfolder and register it in the *_package_*.js file, which includes widgets for the csr page only. This helps to improve performance by only loading the widgets required for that page.

- We have customized the Dojo FloatingPane widget to make it hidden by default (added "dojo.style.hide(this.domNode)" into FloatingPane.js's fillInTemplate function). We prefer to not have the floating panes displayed while the page is initially loaded. If you need the FloatingPane to be shown by default, please add dojo.style.show(someFloatingPaneWidgetInstance) in the pageOnLoad function.
- Make sure the dojo event name are unique on each page.
- Make sure the dojo widget id are unique on each page.
- Make sure there is no extra comma when declare widget function or using json notation.

> **Be careful with the last comma**
> When declaring a set of functions or using JSON, there cannot be a comma after the last item. It is particularly important to check for this when developing

with Firefox because Firefox will ignore the comma. In Internet Explorer, however, the page will fail to load when there is an unnecessary trailing comma.

## MVC pattern in AJAX

The Model View Controller (MVC) pattern is used in the Commerce Manager browser-side design.

- View: HTML and CSS
- Model: the model object, e.g. a Customer exposed through DWR
- Controller: event handling performed by the Controller JavaScript object

## JavaScript limitations

- JavaScript has a concept of objects and classes but no built-in concept of inheritance.
- JavaScript only has a few built-in types (boolean, float, string, object, array) and it converts from one to another quite freely.
- JavaScript objects (some call them associative arrays) are like maps in a O-O style with map keys of type String.
- Use the following to find out whether a JavaScript object supports a certain property or function:

```
if (typeof (myObj,someProperty) != undefined) {
            ...
       }
if (myObj instanceof MyObj) {
    ...
            }
```

The instance instantiated from JSON is always of type Object Array, therefore it does not work well with the "instanceof" operator.

## JavaScript Utilities

There are many tools that can assist JavaScript development.

### Debugging

- DOM Inspector (built into Mozilla, need to grab IE plugins) - To observe the current state of the page
- Venkman or Microsoft Script Debugger - to troubleshoot behaviorial problems
- Ajax Debugging Tool (http://blog.monstuff.com/archives/000252.html)
- IE Leak Detector (http://blog.monstuff.com/archives/000252.html)
- TrimBreakpoint (http://trimpath.com/project/wiki/TrimBreakpoint)

- JavaScript console in your browser - To check for errors

**Misc**

- JSDoc (http://jsdoc.sourceforge.net/)
- JsLint (http://www.crockford.com/javascript/lint.html)

**Unit Testing**

- JsUnit (http://sourceforge.net/projects/jsunit)
- Selenium (http://selenium.thoughtworks.com)

**Editor**

- JSEclipse - eclipse plugin for js editing

## *Web*

The web layer is responsible for managing the interaction between the view layer and the lower layers of the application. The two main technologies used in the AquaLogic Commerce Services web layer are Spring MVC and DWR. The AJAX Controller Layer is a sub-layer of the web layer that is used to provide fine-grained page-specific services that are invoked through AJAX.

### Spring MVC

Spring MVC is a web framework used to separate Velocity templates in the view layer from the underlying domain model and services. Refer to Spring MVC - Web Framework for more information.

### Direct Web Remoting (DWR)

DWR is a technology that allows JavaScript running in the browser to interact with Java running on the server. Refer to DWR - AJAX Remoting Framework for more information.

### AJAX Controller Layer

The AJAX controller layer provides support for DWR-enabled pages and fills a similar role to that of the Spring MVC controller layer. An Ajax controller contains functionality specific to one page and serves both an incoming and outgoing role in satisfying a request. DWR-enabled pages often require data in a custom structure for presentation. The AJAX controller is responsible for requesting this information from the underlying service layer, formatting it, and making it available to the client side. When multiple objects or collections must be made available on the client side, they should be wrapped in an AjaxBean. When data is sent back to the server, the AJAX controller may be required to perform an intermediate processing role to convert data into a form that is accepted by the service layer. If it is necessary to simply invoke a method on a domain object, the AJAX controller should perform this action and make the result available to the client side.

# Spring MVC Web Framework

Spring MVC is a web application framework that performs a similar role to that of Apache Struts. The purpose of Spring MVC is to separate view layer logic from the underlying domain objects and services to maintain loose coupling and application maintainability.

## Model View Controller (MVC)

The MVC pattern is used to achieve the separation between view and domain. In the Spring MVC layer, the "Model" is the domain objects and services in lower layers of the system. The "View" is the Velocity templates that render information about the model and collect user input. The "Controller" component represents a set of Java classes called controllers that manage the interaction between the View and the Model. At a high level, the intent of the Controllers is to prevent coupling between the domain (and services) and the Velocity that is used to present them.

## Spring MVC Controllers

The Spring MVC Controllers typically manage a single page request and serve one or more of the following roles in displaying the page.

- Specify the Velocity template that will be displayed to satisfy the request.
- Obtain and optionally prepare domain objects for display by the Velocity template.
- Declare a validator to validate field input.
- Invoke methods on domain objects or services as required to satisfy the request.
- Return an appropriate response after performing the required task.

Spring MVC Controllers are declared and configured in the url-mapping.xml file.

- This file declares and configures the controller classes as beans.
- The URL mapping section at the top of the file specifies which controller should be invoked for a given URL.

Spring MVC Controllers should contain only a small amount of logic to initiate operations in lower layers. Work flow logic should appear in services, not in Spring Controllers.

- The problem with implementing workflow logic in controllers is that only Spring MVC has access to it – Web services and DWR will not be able to reuse the logic.
- Therefore, even when it doesn't look like there's much need for a service layer call (ie. only a few methods will be executed in the service layer method) logic should still be implemented in the service layer.

## AquaLogic Commerce Services Controller hierarchy

Controllers in AquaLogic Commerce Services will typically inherit from one of the following base classes. The base class to extend depends on the purpose of the controller.

- **SimplePageController** - Use this controller to load a static page that does not invoke any service layer methods such as a "Return Policy" page. There is no need to implement a new controller class when using SimplePageController, you can just use Spring configuration in url-mapping.xml to create an instance of SimplePageController and specify the Velocity template to be displayed.
- **AbstractEpController** - Extend this controller when your page does not use a form, but you want to perform an action on the service layer or a domain object. For example, a page that changes the user's locale might read a local parameter from the request and change the locale in the user's profile. When extending AbstractEpController, override handleRequestInternal() to implement the logic that is performed when a user visits the page.
- **AbstractEpFormController** - Extend this controller to create pages with HTML forms. For example, consider a "forgotten password" page that requests the user's email address and then emails them a new password. When using AbstractEpFormController, a "form-backing" object is specified as the "command" property in url-mapping.xml. This form-backing object will receive the form input specified by the user. Override onSubmit() to perform the action that occurs when the user submits the form. If your form-backing object requires further preparation before display, you can perform this initialization by overriding formBackingObject() and returning a prepared object.

**Specify a Validator**

If your form input requires validation, specify a validator in the Controller's Spring configuration block. Validation for new form-backing objects can be specified in validation.xml.

## DWR – AJAX Remoting Framework

Direct Web Remoting (DWR) is a technology that allows JavaScript running in the browser to interact with Java running on the server. The two main roles of DWR are

- Allow JavaScript code in the browser to remotely invoke Java object methods on the server.
- Convert objects between Java and JavaScript representations so that they can be passed between the browser and the server.

At a high-level, this is accomplished by

- Connecting to a web application (server) to receive incoming requests.
- Generating JavaScript code to be used in the browser that has a similar interface to the Java code that is being 'wrapped'.
- Executing JavaScript callback functions when asynchronous requests are processed by the server.

DWR functionality is added to an application by adding the DWR Servlet to the web.xml file. The server-side methods and objects that are available in JavaScript are configured in dwr.xml.

### How to expose server-side methods to JavaScript running in the browser

The server-side methods and objects that are available in JavaScript are configured in dwr.xml. The following code sample demonstrates the key sections of this file and how they are used together to expose server-side functionality.

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
  <!-- Extending DWR -->
  <init>
    <creator id="null"
class="uk.ltd.getahead.dwr.create.NullCreator"/>
    <converter id="epBean"
      class="com.elasticpath.web.ajax.EpPersistedBeanConverter"/>
  </init>

  <allow>
    <!-- create creator="spring" javascript="customerService"
         scope="application"-->
    <!-- scope value defaults to "page". Use of "session"
         requires cookies.  -->
    <!-- possible scope values: "application", "session",
         "request" and "page". -->
    <create creator="spring" javascript="customerService"
scope="request">
      <param name="beanName" value="customerService"/>
      <include method="get"/>
      <include method="findCustomerLike"/>
      <include method="update"/>
    </create>
    ...
    <convert converter="epBean"
match="com.elasticpath.domain.Customer">
       <param name="include" value="uidPk, userId, email,firstName,
         lastName, creationDate, lastEditDate, dateOfBirth,
         encryptedPassword, addresses, orders"/>
    </convert>
    ...
  </allow>
  <signatures>
```

```
    <![CDATA[PD:
      import java.util.Set;
      import java.util.List;
      import com.elasticpath.sfweb.ajaxservice.impl.
        SkuConfigurationServiceImpl;
      SkuConfigurationServiceImpl.getAvailableOptionValues(long
productUid,
        List<String> selectedOptionValueCodes);
      ]]>
  </signatures>
</dwr>
```

Configuration elements in the code above are defined in the notes below.

- The <init> - (optional) defines new/customized Creator or Converter.
- The <allow> - (required) defines which classes DWR can create and convert.
- The <create> - (required) specifies how to create remote beans.
- The creator - specifies how to create the object (ie, "Spring", "new").
    - javascript - specifies an arbitrary name for the JavaScript proxy.
    - scope - specifies the scope of the bean instance.
    - include/exclude - elements define which methods are to/not to be exposed (can NOT use both).
- <convert> - specifies how to convert the method parameters.
    - converter - specifies the converter to marshal data between the client and the server.
    - match - the class of the parameter to be converted.
    - param - "include"/"exclude" specify the list of properties to/not to be exposed (can NOT use both).
- <signatures> - Contains additional type information for the conversion process. This type information is required when passing a collection of objects to the server. In this case, DWR will use information in the <signatures> block to determine the type of the objects in the collection.

By default, the DWR servlet will look for a configuration file named dwr.xml under [appRoot]/WEB-INF. It also allows you to use multiple configuration files with with locations specified in the web.xml.

**Signatures Block**

Use the signatures block in dwr.xml to declare the type of collections of JavaScript objects being passed to the server.

### How to invoke server-side methods from JavaScript

JavaScript files that invoke server-side methods on a JavaScript proxy must make the following declarations.

```
<!-- Reference the javascript proxy for the server-side object whose
method is to be
invoked. Include a line such as the one below for each Java class
where methods will
be invoked. -->
<script type='text/javascript'
xsrc='dwr/interface/customerService.js'></script>


<!-- Reference the dynamic JavaScript proxy generation library
(engine.js) -->
<script type='text/javascript' xsrc='dwr/engine.js'></script>


<!-- Reference the DWR utility library (util.js) -->
<script type='text/javascript' xsrc='dwr/util.js'></script>
```

Once the above references are declared, server-side calls can be made by simply invoking methods on the JavaScript proxy. The last parameter to the server-side method declares the JavaScript callback method that will be invoked when the call returns from the server such as in the following example.

```
customerService.get(customerId, this.getCustomerCallback);
```

If an error handler is needed to perform additional logic on the browser side in the event that the call fails, use the following syntax.

```
customerService.update(customerToCommit,
       {callback:this.onCustomerCommitCallBack,
        timeout:5000,
        errorHandler:this.onCustomerCommitErrorHandler});
```

In the above example, a JavaScript call metadata object is the last parameter to the remote method call. This object specifies both the call back method and the method that should be invoked in the event that the remote operation fails.

All callback methods have only one parameter, which is the return value from the remote method. A JavaScript closure is needed to pass additional information.

```
shippingRegionService.list({callback:function(results) {
        message = function () {};
        if (results != null && typeof results == 'object') {
            message.data = results;
            if (source == "fromRegionSummaryPane") {
```

```
dojo.event.topic.publish("onShippingRegionListReturnToSummary",
                  message);
            } else if (source == "fromServiceLevelDetailPane") {

dojo.event.topic.publish("onShippingRegionListReturnToService",
                  message);
            }
        }
    }
});
```

## How to configure a new object for conversion by DWR

When a remote method returns a Java object, DWR needs to know how to convert that Java object into a JavaScript object that can be used in the browser. Similarly, when the browser sends a JavaScript object as a parameter to a remote method call, DWR needs to know how to convert the JavaScript object into a Java object.

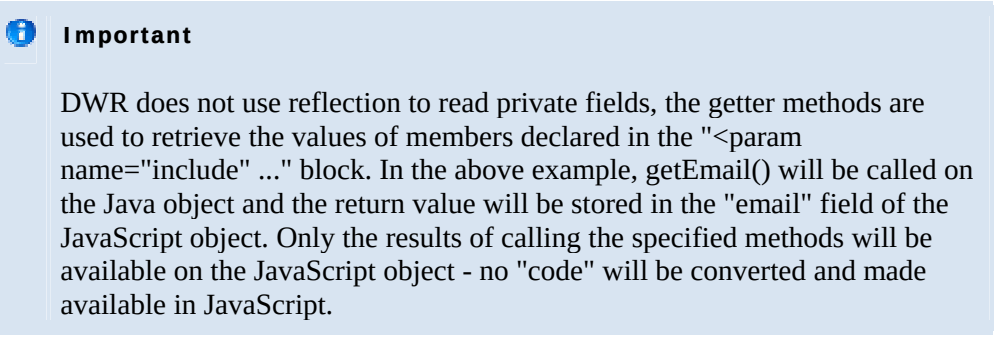
There are two steps to configure a new object for conversion by DWR.

**Step 1 - Declare the object in dwr.xml**

The following snippet from dwr.xml shows how to declare a converter for the new object.

```
<convert converter="epBean" match="com.elasticpath.domain.Customer">
      <param name="include" value="uidPk, userId, email, firstName,
       lastName, creationDate, lastEditDate, dateOfBirth,
       encryptedPassword, addresses, orders"/>
</convert>
```

This specifies that Java objects of type com.elasticpath.domain.Customer will be converted by a converter with the bean id, "epBean." epBean is the bean id given to EpBeanConverter.java. Therefore, EpBeanConverter will be responsible for converting Customer objects. The parameter block declares which members of the Customer object will be available on the JavaScript object.



**Important**

DWR does not use reflection to read private fields, the getter methods are used to retrieve the values of members declared in the "<param name="include" ..." block. In the above example, getEmail() will be called on the Java object and the return value will be stored in the "email" field of the JavaScript object. Only the results of calling the specified methods will be available on the JavaScript object - no "code" will be converted and made available in JavaScript.

**Step 2 - Create or update a converter**

The "converter" attribute of the <convert> block above specifies the bean name of the class that will actually perform the conversion. In most cases, the existing EpBeanConverter class can be used to convert your new class.

## Using EpBeanConverter

When using EpBeanConverter to convert a new class, the converter needs to know how to get an instance of the new class during **inbound** conversion. If objects of the new class are typically retrieved from a singleton Service, add a new map entry in EpBeanConverter.java as shown below.

```
BEAN_SERVICE_MAPPING.put(Order.class,
  ELASTICPATH.getBean(ContextIdNames.ORDER_SERVICE));
```

The above example tells the converter that Order instances can be retrieved by calling getObject() on the order service.

If your bean does not have an identity of its own and any new instance will suffice, add a map entry to EpBeanConverter.java as shown below.

```
BEAN_CONTEXTID_MAPPING.put(OrderAddress.class,
  ContextIdNames.ORDER_ADDRESS);
```

In this case, the map entry tells the converter that instances of OrderAddress can be retrived by calling getBean() with the given bean name from the ContextIdNames constants.

## Using a custom converter

In some cases, you may wish to perform special processing logic upon inbound or outbound conversion. For example, you may wish to use a simplified string representation of a Locale or Currency on the browser side. In this case, you can implement DWR's Converter interface and specify your custom converter as the converter to use in the <converter> block in dwr.xml (Your converter will need to be configured elsewhere as a Spring bean with an id that can be specified in the <converter> block).

## DWR Servlet

The DWR servlet acts as a mini-server for delivering JavaScript to the browser and dynamically generating proxy objects. The following declarations in web.xml add DWR to the web application and do not need to be modified when adding additional DWR-enabled features.

```
<!- DWR - Direct Web Remoting ->
<!- Set debug to FALSE for production site ->
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <display-name>DWR Servlet</display-name>
```

```
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>true</param-value>
  </init-param>
  <!-- Specify the config file(s) if not using the
       default location: WEB-INF/dwr.xml
  <init-param>
    <param-name>configXXXXX</param-name>
    <param-value>WEB-INF/dwr.xml</param-value>
    <description>What config file(s) do we use?</description>
  </init-param>
  -->
</servlet>
<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

### Additional DWR features

- DWR makes it easy to display a "Loading..." indicator.
- Exposes a pre- and post- hook after some remote operation takes place:
    - DWREngine.setPreHook(function)
    - DWREngine.setPostHook(function)
- Worried about latency? DWR allows you to batch opereations:
    - DWREngine.beginBatch()
    - DWREngine.endBatch()
- Race conditions caused by asynchronicity got you down?
    - DWREngine.setOrdered(true) forces serial FIFO execution of DWR requests
- Once the application is up, you can check all the services/methods exposed at:

```
http://[host]:[port]/[context]/dwr/
```

- You can use DWRUtil.toDescriptiveString(**object**, **level**) to investigate the content of the specified **object** to a certain **level**. For example,

```
// To display ojb1 if obj1 exists, use:
dojo.debug(DWRUtil.toDescriptiveString(obj, 1);

// If obj1 contains a reference to obj2 and you want to
// investigate obj2's properties, use:
   dojo.debug(DWRUtil.toDescriptiveString(obj, 2);
```

## Exception Handling

- DWR can marshall exceptions, and they will become errors in JavaScript (they can't be thrown since this will probably be happening asynchronously). The first parameter of the error handler contains the "message" of the caught exception i.e. the result of calling getMessage() on the exception. To have access to more information about the exception, we can "allow" DWR conversion of the exception object by adding:

```
<convert converter="bean"
match="com.elasticpath.service.EpServiceException">
  <param name="include" value="name, message,
localizedMessage"/>
   </convert>
```

  to dwr.xml. The converted exception object will be the second parameter to the error handler.

- In addition, to make it easy to identify the caught exception, we have added a "getName" method to EpServiceException. It will return the full classname of the caught exception, i.e. com.elasticpath.service.EmailExistException. Using this information, the specified field on the user input form can be highlighted.

## Best Practices

- Use create "null" to expose server session object, for example shoppingCart.
- Only expose the required properties of the model by using "include".
- Customize the bean converter - the default DWR bean converter uses new operation on inbound conversion (from js object to java object). This does not work when a constructor is not exposed.
- EP bean converters (Customized bean converters can be categorized into 3 types):
    - Persisted bean converter - EpPersistedBeanConverter:
        - Define a static bean to service mapping for entity bean (beans that has a corresponding service object);
        - Define a static bean to bean id mapping for non-entity beans;
        - On converting, load the bean instance using the above two maps, and populate the properties after.
    - Transient bean converters:
        - AbstractEpTransientBeanConverter is the parent class, which extends dwr's BeanConverter.
        - EpConstantConverter, EpRuleElementConverter and etc., all extends AbstractEpTransientBeanConverter.
    - Utility connverter - these are used for some special needs, i.e., as a workaround for not being able to use objects as the map key in JavaScript.
        - EpCurrencyConverter (to convert java.util.Currency object to/from java.lang.String)

- EpLocaleConverter (to convert java.util.Locale object to/from java.lang.String)

### Limitations

- JavaScript does **NOT** support method overloading, so it is **impossble** to expose two Java methods with the same name but different signatures
- You can not expose methods in non-singleton domain objects. If you want to expose business logic inside a domain object, you will need to create a wrapper method in the service layer.
- Only maps with String or other primitive-typed keys can be used in JavaScript.

**Tip**

Using complex types as hash keys presents a problem when trying to expose the map to the client side

In JavaScript, array keys must be literals. The following are identical:

```
var a={3:"hello"};
var a={"3":"hello"};
```

As a result, the java Map with a complex object as the key cannot be converted by DWR as it is.

There are two workarounds:

1. Always use Maps with String keys.
2. Leave the map with the complex typed key, and customize the DWR Converter to take care of the difference.

Solution 1 is preferred. When using solution 2, there will be some performance overhead and different object representation on the client and server side.

## *Service*

The service layer provides services to various consumers in the web layer as well as web service consumers. There are several types of services that serve different roles in the application.

### Persistence Services

Persistence Services provide the capability to save and retrieve domain objects. Persistence Services extend from AbstractEpPersistenceServiceImpl and offer methods for adding new domain objects, retrieving domain objects by their identifier, and searching for domain objects with specific criteria. PersistenceServices are named XService where X is the class name of the objects that it can save and retrieve.

### Domain Services

Domain services typically implement the logic for a use case that is inappropriate for encapsulation by any one domain object. For example, a service that performs a checkout will

contain logic for the flow of the interaction between several domain objects. This domain service logic is typically at a higher level of abstraction than the fine-grained domain logic that is implemented by an individual domain object. Domain services will often use other services in combination with domain object logic to accomplish a task. For example, the checkout service uses domain logic to check for sufficient inventory while using the InventoryService persistence service to persist inventory levels.

In AquaLogic Commerce Services, domain service logic specific to a particluar domain object is often implemented within the Persistence Service for that object.

### Integration Services

Integration services implement functionality that is invoked by domain services but considered outside the domain of an ecommerce application and typically integrate with other systems or technologies. The following are examples of integration services.

- EmailService - Sends email on behalf of other services.
- CustomerIndexBuildService - Constructs a search index used by the Lucene search feature.
- BirtReportService - Provides access to the BIRT reporting engine.

### System Services

System services handle various concerns that cut across many parts of the application. These services are typically provided by Spring and configured in Spring configuration files. Examples of System services include object lifecyle management, caching, transactions, security, and scheduling.

### Web Services

Web services expose service layer functionality to web services clients. These services are ultimately delivered to external systems via SOAP.

## *Domain*

The Domain layer contains an object model of the ecommerce domain. This object model consists of classes that model real-world entities such as customers and products. The behavior and relationships of these classes should be a reflection of the real-world entities. For example, customers have collections of addresses and products have references to price objects. As much as possible, domain objects should encapsulate their behavior so that the objects who collaborate with them are unaware of internal implementation details. Furthermore, domain objects should be kept relatively free from the constraints of frameworks, persistence, etc. so that they are a relatively pure expression of the domain.

The following sections cover key topics and design considerations in domain model development.

## Domain object inheritance structure

When creating a new domain object, you will need to consider which abstract domain class to inherit from. In most cases, you will need to inherit from one of the leaf nodes of the existing inheritance tree structure: Transient, Entity, or ValueObject. The following interfaces (and their corresponding abstract implementation classes) define the inheritance tree structure.

- **EpDomain** - represents a general domain object with a reference to ElasticPathImpl.
    - **Transient** - Extends EpDomain and represents a transient (not persistent) domain object.
    - **Persistence** - Extends EpDomain and represents a persistent domain object.
        - **Entity** - Extends Persistence and represents a domain object with its own identity.
        - **ValueObject** - Extends Persistence and represents a domain object that is a value with no identity of its own.

## How to create new instances of Domain Objects

In AquaLogic Commerce Services, new instances of built-in classes are often obtained by calling getElasticPath().getBean(BEAN_NAME). All objects are typically "wired" with an instance of ElasticPathImpl and inherit the getElasticPath() method so that this functionality is always available. The BEAN_NAME parameter is a constant bean name defined in ContextIdNames.java.

```
//Create a new instance of Customer
Customer myCustomer = (Customer)
getElasticPath().getBean(ContextIdNames.CUSTOMER);
```

If you are adding a new domain object to the system, you can instantiate it using the new operator and then set the ElasticPath instance.

### Spring configuration

In some cases, you will need to declare multiple domain objects that are initialized with different property values. In this case add the bean definition using standard Spring configuration to domainModel.xml or domainModelCM.xml if the domain model object will only be accessed by the Commerce Manager. Note that this way of declaring instances is used for all singleton service objects. Instances of these objects can be retrieved using the following syntax.

```
MyService myService = (MyService) getElasticPath().getBean(“Bean Id of
MyService”);
```

## Domain object identity - UIDPK vs GUID

Domain objects in AquaLogic Commerce Services have two kinds of identifiers, a UIDPK and a GUID.

The UIDPK is a surrogate key which is generated by the Hibernate HI/LO algorithm automatically when a record is added to a table. After it is created, its value cannot be changed. In database tables, UID_PK is a unique primary key.

GUID is the acronym for Globally Unique IDentifier. In AquaLogic Commerce Services it is used as the general name for the identifier of an entity object. In most cases, the GUID is the natury key of an entity object. For example, the natural key of ProductSku is its SKU code, so the SKU code is the GUID for ProductSku objects. In other words, you can think of GUID as a generic name for identifier of entity object. However, some specific entities may have their own name for the same identifier, such as "SKU code."

The following table provides more comparison between the UIDPK and the GUID.

| | UIDPK | GUID | COMMENTS |
|---|---|---|---|
| TYPE | Integer | String | |
| LENGTH | 32 bit or 64 bit | 255 byte (maximum) | |
| SCOPE | One system | Multiple systems | The same product might have differenct UIDPK in the staging and production database, but they will always have the same GUID. |
| USAGE | Entity object or value object | Entity object | UIDPK is used to identify an entity or a value object in one system. It's also used in associations(foreign key, etc.) between entities and value objects.<br>GUID is only used to identify an entity. It can be used from in CRUD(create, retrieve, update & delete) operations on an entity from other systems (e.g. web services and the import manager). |
| GENERATION | Automatically through Hibernate HI/LO | Hybrid | You can call the setGuid() method to manually set a GUID. If you don't manually set one, the GUID is to assigned when you create a new entity. The default behavior is to allow the GUID to be automatically assigned. Unlike the UIDPK, a GUID can be changed after creation. |

| | UIDPK | GUID | COMMENTS |
|---|---|---|---|
| ALIASES | N/A | Can have aliases for different entity objects | Examples:<br>ORDER GUID is also called ORDER NUMBER<br>SKU GUID is also called SKU CODE<br>PRODUCT GUID is also called PRODUCT CODE<br>CATEGORY GUID is also called CATEGORY CODE<br>ATTRIUBTE GUID is also called ATTRIBUTE KEY |

## Bi-Directional Relationships

Avoid creating bi-directional relationships between parent objects and the child objects that they aggregate using a collection class. By avoiding bi-directional relationships we eliminate the complexity of maintaining the parent link when a child is added or removed from the collection. In some cases the bi-directional relationship cannot be avoided. For example, ProductSku references Product because it must fall back to the product's price when a client requests a price from the SKU but no price has been defined at the SKU level.

## Domain Object Serialization

Generally, domain objects should be made serializable because they might be replicated from one application server to another in a clustered application server environment. To make a domain object serializable, it must implement the "Serializable" interface and all of its aggregated fields must either be serializable or transient. When you declare a field reference to a service or a utility in a domain object, they should be defined as transient.

```
public class BrandFilterImpl extends AbstractTransientImpl implements
BrandFilter {
  private static final String ERROR_MSG = "Invalid brand filter id:";

  private String filterId;
  private int uid;
  private transient Utility utility;
  private transient BrandService brandService;
  private Brand brand;

  ...
}
```

## Setting default field values for domain objects

There are three ways to set default values for domain object fields.

- Set values in the domain object constructor - This technique is seldom used because the field initialization can no longer be controlled.

- Field initializer declaration - This may be used for fields whose default values are cheap to create.
- Set values in the setDefaultValues() method - This is the preferred technique.

Using setDefaultValues() is preferred because it can be used to control when default values are initialized. In production, it is wasteful to set expensive default values when creating new domain objects because they will typically be overwritten by Hibernate immediately. For example, Maps consume a lot of memory while computing fields like GUIDs and dates are CPU intensive. When running JUnit tests, however, we will need to set the default values so that the functionality can be tested without throwing NullPointerExceptions.

When setting default values in setDefaultValues(), check that the value has not yet been set before initializing fields.

```
public void setDefaultValues() {
  super.setDefaultValues();
  if (this.startDate == null) {
    this.startDate = new Date();
  }
  if (productCategories == null) {
    productCategories = new HashSet();
  }
  if (productPrices == null) {
    productPrices = new HashSet();
  }
  if (promotionPrices == null) {
    promotionPrices = new HashMap();
  }
  if (localeDependantFieldsMap == null) {
    localeDependantFieldsMap = new HashMap();
  }
  if (productSkus == null) {
    productSkus = new HashMap();
  }
}
```

## *Data Access*

The data access layer is responsible for saving and retrieving data from persistent storage. The majority of persistent data in AquaLogic Commerce Services is stored in the database using the Hibernate object/relational persistence and query service. A small number of configuration files are persisted directly to the file system using XML and properties files. Objects that are aware of

persistence implementation details such as file formats or whether data exists in a database are called Data Access Objects (DAO).

## Hibernate object/relational persistence

Hibernate is a persistence service that maps objects to tables in a relational database. See the Hibernate documentation for more information.

## File system persistence

Two types of data files are used by the file system persistence objects in AquaLogic Commerce Services, XML files and properties files.

## XML file persistence

AquaLogic Commerce Services uses a file called elasticpath.xml as the main source of configuration settings. This file is read by ElasticPathDaoXmlFileImpl, which uses the JDOM library to parse the file and store the configuration settings in the ElasticPathImpl singleton. This operation occurs once at system startup and the settings in elasticpath.xml cannot be changed at runtime.

### Properties file persistence

Some configuration data is stored in a small number of properties files. These files are stored in conf/resources and are read into the system by PropertiesDaoImpl. Examples of data stored in properties files include the list of countries and country codes, and the status of Lucene index builds.

## Hibernate – Data Persistence Framework

Hibernate is an object/relational mapping (ORM) persistence and query service. At a high level, the purpose of Hibernate is to map Java objects to tables in a relational database to simulate an object database in which objects can be saved and retrieved directly. We use Hibernate mapping files, which are XML configuration files, to instruct Hibernate on how to persist objects of a given class in a given table. In addition to specifying the table, the mapping files also specify which of an objects fields will be mapped to which columns in the table. Hibernate also supports collection mappings. In this case, Hibernate looks up mapping files as required to know how to persist a parent object and child objects that it aggregates. Using this feature, it is possible to persist only the parent object and have Hibernate automatically "cascade" persist changes to child objects.

Hibernate allows objects to be reconstructed from relational database tables when requested using the object's class and unique identifier. While this is useful, Hibernate also provides an object-oriented query language called Hibernate Query Language (HQL) that allows the retrieval of arbitrary Java objects that match a given query.

## Using Hibernate in AquaLogic Commerce Services

In AquaLogic Commerce Services, Hibernate is used only in the Data Access layer. Persistence Services in the Service layer typically maintain reference to an instance of PersistenceEngine and the default implementation of PersistenceEngine is HibernatePersistenceEngineImpl, which is in the Data Access Layer. HibernatePersistenceEngineImpl then invokes Hibernate-specific methods on a HibernateTemplate object to accomplish persistence tasks.

Queries in HQL are encapsulated in the Criteria.java constants file. Services pass these queries to the PersistenceEngine to query the database for objects requested by their clients.

## Hibernate mapping files

Hibernate mapping files are found in the core project in the WEB-INF/conf/hibernate_mapping directory. All mapping file names follow the convention X.hbm.xml where X is the name of the class being persisted. These mapping files indicate to Hibernate which fields should be stored in which table in order to persist an object. This is relatively simple for persisting primitive fields of an object. However, when specifying persistence instructions for an object's collections there are several design alternatives with corresponding requirements for the underlying database structure. Please refer to the Hibernate documentation for details on how to map collections. The following set of annotated XML snippets are from the mapping file for the Product class.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

  <!-- Specify that objects with the Product interface should be
stored in
       the TPRODUCT database table. -->
  <class table="TPRODUCT" discriminator-value="0"
      name="com.elasticpath.domain.catalog.Product">

  <!-- Specify the identifier field for this object. See the
       Hibernate identifiers section below for details. -->
  <id name="uidPk">
    <column name="UIDPK"/>
    <generator class="hilo">
       <param name="table">HIBERNATE_UNIQUE_KEYS</param>
       <param name="column">value</param>
    </generator>
  </id>
```

```
  <!-- Set the discriminator value. See the Interfaces and
discriminators
       section below for details -->
  <discriminator type="integer" formula="1"/>

  <!-- map object property names to database columns -->
  <property name="startDate" not-null="true"
      type="java.util.Date" column="START_DATE"/>

  . . .

  <property name="salesCount"
      type="integer" column="SALES_COUNT"/>

  . . .

  <!-- Mapping for a collection (a Map) called productSkus -->
  <map lazy="true" cascade="all,delete-orphan" inverse="true"
      name="productSkus" fetch="subselect">

    <!-- Specify the foreign key on the child table -->
    <key column="PRODUCT_UID" not-null="true"/>

    <!-- Specify the key of the map -->
    <map-key type="java.lang.String" formula="SKUCODE"/>

    <!-- Specify the aggregated interface -->
    <one-to-many class="com.elasticpath.domain.catalog.ProductSku"/>
  </map>

  <!-- Mapping for a simple reference to another object -->
  <many-to-one not-null="true" column="PRODUCT_TYPE_UID" lazy="proxy"
    name="productType"
class="com.elasticpath.domain.catalog.ProductType"/>

  <!-- Typical mapping for a collection as a set -->
  <set lazy="true" cascade="all,delete-orphan"
name="productCategories"
      fetch="subselect">
    <key column="PRODUCT_UID"  not-null="true"/>
    <one-to-many
class="com.elasticpath.domain.catalog.ProductCategory"/>
```

```
  </set>

  <!-- Components are referenced classes that are mapped in-line
instead of
       in a separate mapping file -->
  <component name="attributeValueGroup"

class="com.elasticpath.domain.attribute.impl.AttributeValueGroupImpl"
>



    <!-- Mapping for the collections and fields of the component -->
    <map table="TPRODUCTATTRIBUTEVALUE" lazy="true"
        cascade="all,delete-orphan" name="attributeValueMap"
fetch="subselect">
      <key column="PRODUCT_UID" not-null="true"/>
      <map-key type="java.lang.String"
column="LOCALIZED_ATTRIBUTE_KEY"/>
      <composite-element

class="com.elasticpath.domain.attribute.impl.AttributeValueImpl">
        <property name="localizedAttributeKey"
          type="java.lang.String" formula="LOCALIZED_ATTRIBUTE_KEY"/>
        <property name="shortTextValue"
          type="java.lang.String" column="SHORT_TEXT_VALUE"/>
        . . .
      </composite-element>
    </map>
  </component>

  . . .

  <!-- Specify the implementation class -->
  <subclass name="com.elasticpath.domain.catalog.impl.ProductImpl"
      discriminator-value="1">

    <!-- Properties of the implementation class that do not appear
         in the interface -->
      <many-to-one name="defaultCategory" lazy="proxy"
        column="DEFAULT_CATEGORY_UID" access="field"/>

      <property name="code" not-null="true" length="64"
        type="java.lang.String" column="CODE" unique="true"/>
```

```
    . . .
  </subclass>

  </class>
</hibernate-mapping>
```

### Hibernate identifiers

Hibernate requires that all persistent objects have a unique identifier. In AquaLogic Commerce Services, Hibernate generates the identifier using the Hi/Lo algorithm. There are many other algorithms available in Hibernate for generating identifiers. The following block of the Hibernate mapping file declares the unique identifier and the same code is used in all mapping files.

```
<!-- Specify the object field where the identifier is stored -->
  <id name="uidPk">

    <!-- Specify the database column -->
    <column name="UIDPK"/>

    <!-- Specify that the Hi/Lo algorithm is to be used to
         generate identifiers -->
    <generator class="hilo">

       <!-- Specify the table where the seed value used by
            Hi/Lo can be found -->
       <param name="table">HIBERNATE_UNIQUE_KEYS</param>

       <!-- Specify the column where the seed value used by
            Hi/Lo can be found -->
       <param name="column">value</param>
    </generator>
  </id>
```

### Interfaces and discriminators

Hibernate uses the concept of a "discriminator" to distinguish between objects of different Java types that are stored in the same table. The discriminator is typically a column in the table that identifies the type of object to instantiate when re-creating an object from persisted data. The discriminator is used in this way in certain situations in AquaLogic Commerce Services. However, the mapping files for most objects use the discriminator to distinguish between the interface and the single implementation class that is stored in the table. This allows Hibernate to map most database columns to properties of an interface rather than an implementation class.

To use this technique, we first specify that the discriminator value for an interface is "0" using the following line.

```
<class table="TPRODUCT" discriminator-value="0"
  name="com.elasticpath.domain.catalog.Product">
```

Next, we specify that the discriminator for the interface is a formula whose value is "1".

```
<discriminator type="integer" formula="1"/>
```

Specifying a formula (instead of a column) avoids the need for a column to hold the discriminator value. A column is not necessary because the discriminator value is always 1.

Within the interface mapping block, we specify the subclass as follows.

```
<subclass name="com.elasticpath.domain.catalog.impl.ProductImpl"
  discriminator-value="1">
```

This indicates that the implementation class for this table is ProductImpl, with a discriminator value of "1". Since the discriminator formula defined above always has the value "1" the implementation class will always be instantiated. This allows most properties to be mapped using the interface while still specifying the implementation class to use. If there are properties of the implementation class that do not appear in the interface, they are mapped inside the <subclass> block. The following XML snippet ties these concepts together.

```
<!-- Declare interface and its discriminator -->
  <class table="TPRODUCT" discriminator-value="0"
      name="com.elasticpath.domain.catalog.Product">

  <!-- Set the discriminator value to always be 1 -->
  <discriminator type="integer" formula="1"/>

  <!-- Mappings for properties of the interface -->
  <property name="startDate" not-null="true"
     type="java.util.Date" column="START_DATE"/>

  <!-- Declare the implementation class with a
    discriminator of 1 so it is always instantiated -->
  <subclass name="com.elasticpath.domain.catalog.impl.ProductImpl"
      discriminator-value="1">

      <!-- Mappings for properties that do not appear in the
interface -->
      <property name="code" not-null="true" length="64"
        type="java.lang.String" column="CODE" unique="true"/>
```

```
  </subclass>

  </class>
</hibernate-mapping>
```

### Hibernate Query Language (HQL)

Hibernate also provides an object-oriented query language called Hibernate Query Language (HQL) that allows the retrieval of arbitrary Java objects that match a given query. HQL is similar to SQL but you select on objects and properties of objects rather than tables and fields. Many HQL queries used in AquaLogic Commerce Services return objects or lists of objects. However, it is possible to select on field values as well, which will be returned as a collection of arrays containing the String values.

HQL allows you join an entity with its child components and select on them. For example, if you want find Brands that have the display-name "Canon", you can use the following HQL statement.

```
select b from Brand as b inner join
b.localizedProperties.localizedPropertiesMap
  as value where value = 'Canon'
```

In AquaLogic Commerce Services, all HQL is encapsulated in Criteria.java (or a specialized DAO) to maintain a level of independence from the ORM tool.

### Using property accessor methods with Hibernate

When you define a property for an entity in a Hibernate mapping file, by default Hibernate will use the getter and setter to access the property. This can cause unexpected behavior if you have any fallback logic in your getter method. Consider the following example.

```
public String getName() {
if (this.name==null) {
    return "defaultName";
  }
  return this.name;
}
```

When Hibernate uses this getter method, it will think that the name property has changed from null to "defaultName." Due to this change, Hibernate will try to persist the entity even if that is not your intent and you just want to load the entity. To avoid this problem, we recommend writing another method called getNameWithFallback() that implements the fallback logic.

Another way to solve this problem is to set the hibernate access strategy to "field" in the mapping file as shown below.

```
<property name="name" not-null="true" type="java.lang.String"
  column="NAME" access="field"/>
```

In this case Hibernate will access the field directly and bypass the getter logic. This work around is slightly discouraged because it will also indiscriminately bypass any other logic that might be added to the accessor.

## When to use Hibernate update() and when to use merge()

Hibernate provides two methods for you to persist changes on a detached object, update() and merge().

The update method will take the object you specify and save it exactly as-is. The merge method will try to figure out what has been changed on the object and only apply the changes to the database. This means the merge method is normally lighter than the update method.

The update method requires that no object with the same identifier is loaded in the hibernate session and, after an update, your detached object becomes attached. If another object with the same identifier is loaded when you update, Hibernate will throw an exception to inform you that another object with the same ID already exists in the hibernate session.

The merge method can only be used once on a detached object. After a merge, your detached object remains detached. If you call merge more than once on the same detached object, the same changes will be applied multiple times. This is not what you want and may actually cause unexpected database errors, such as constraint violations.

We suggest that you use the update method whenever possible. When you need to use the merge method on a detached object, ensure that merge is never called more than once.

## Hibernate collection mapping tips

**key and map-key must be mapped to columns**

Hibernate generates SQL statements based on your collection mappings. If your mapping files is inappropriate, the generated SQL statement won't be efficient and may result in data errors.

## Using map

When using map as a collection mapping, the key and map-key must be mapped as a column to make hibernate generate the correct SQL statement.

## Buggy example using map

```
<map lazy="false" cascade="all,delete-orphan" name="priceTiers"
         table="TPRODUCTPRICETIER" order-by="MIN_QUANTITY asc">
  <key column="PRODUCT_PRICE_UID"/>
  <map-key type="java.lang.Integer" formula="MIN_QUANTITY" />
  <composite-element
class="com.elasticpath.domain.catalog.impl.PriceTierImpl">
  <property name="listPrice" type="java.math.BigDecimal"
column="LIST_PRICE"/>
  <property name="salePrice" type="java.math.BigDecimal"
column="SALE_PRICE"/>
```

```
  <property name="minQty" type="int" column="MIN_QUANTITY"/>
  </composite-element>
</map>
```

The update SQL statement hibernate created for the buggy example is as follows.

```
update TPRODUCTPRICETIER set LIST_PRICE=?, SALE_PRICE=?,
MIN_QUANTITY=?
where PRODUCT_PRICE_UID=? and LIST_PRICE=? and SALE_PRICE=? and
MIN_QUANTITY=?
```

The above sql statement will fail if any column, such as SALE_PRICE, has a null value in the database because "SALE_PRICE=null" doesn't work well.

### Correct example using map

Notice that the "map-key" line has been changed to "column" rather than "formula".

```
<map lazy="false" cascade="all,delete-orphan" name="priceTiers"
         table="TPRODUCTPRICETIER" order-by="MIN_QUANTITY asc">
  <key column="PRODUCT_PRICE_UID"/>
  <map-key type="java.lang.Integer" column="MIN_QUANTITY" />
  <composite-element
class="com.elasticpath.domain.catalog.impl.PriceTierImpl">
    <property name="listPrice" type="java.math.BigDecimal"
column="LIST_PRICE"/>
    <property name="salePrice" type="java.math.BigDecimal"
column="SALE_PRICE"/>
    <property name="minQty" type="int" formula="MIN_QUANTITY"/>
  </composite-element>
</map>
```

The update sql statement hibernate created for the above correct example is shown below.

```
update TPRODUCTPRICETIER set LIST_PRICE=?, SALE_PRICE=?,
MIN_QUANTITY=?
where PRODUCT_PRICE_UID=? and MIN_QUANTITY=?
```

### Using set

When using set as a collection mapping, the key column and any columns marked not-null will be used in the "where" part of the SQL update statement. If no columns are specified as not-null, all columns will be used in the "where" part of the update sql statement.

### Buggy example using set

```
<set lazy="false" cascade="all,delete-orphan" name="parameters"
                          table="TSHIPPINGCOSTCALCULATIONPARAM">
```

```
  <key column="SCCM_UID"/>
  <composite-element

class="com.elasticpath.domain.shipping.impl.ShippingCostCalculationPa
rameterImpl">
    <property name="key" type="java.lang.String" column="PARAM_KEY"/>
    <property name="value" type="java.lang.String" column="VALUE"/>
    <property name="displayText" type="java.lang.String"
column="DISPLAY_TEXT"/>
  </composite-element>
</set>
```

The update SQL statement hibernate created for the buggy example is as follows.

```
update TSHIPPINGCOSTCALCULATIONPARAM set PARAM_KEY=?, VALUE=?,
DISPLAY_TEXT=?
where SCCM_UID=? and PARAM_KEY=? and VALUE=? and DISPLAY_TEXT=?
```

The above SQL statement will fail if any column, such as DISPLAY_TEXT, has a null value in the database because "DISPLAY_TEXT=null" doesn't work well.

### Correct example using set

Notice the "PARAM_KEY" property has been marked "not-null".

```
<set lazy="false" cascade="all,delete-orphan" name="parameters"
                          table="TSHIPPINGCOSTCALCULATIONPARAM">
  <key column="SCCM_UID"/>
  <composite-element
      class="com.elasticpath.domain.shipping.impl.
             ShippingCostCalculationParameterImpl">
    <property name="key" not-null="true" type="java.lang.String"
column="PARAM_KEY"/>
    <property name="value" type="java.lang.String" column="VALUE"/>
    <property name="displayText" type="java.lang.String"
column="DISPLAY_TEXT"/>
  </composite-element>
</set>
```

The update SQL statement Hibernate created for the correct example is shown below.

```
update TSHIPPINGCOSTCALCULATIONPARAM set PARAM_KEY=?, VALUE=?,
DISPLAY_TEXT=?
where SCCM_UID=? and PARAM_KEY=?
```

### Set key to not null to allow cascading inserts or updates to maps

The key must be set to not null to allow cascading inserts or updates when using map in collection mappings. If you don't specify the key to not null for maps, you may encounter problems when performing cascading inserts or updates. This strange behavior doesn't apply when using set in collection mapppings. It's possible that this is a bug in Hibernate.

### Buggy Example

```
<map lazy="false" cascade="all,delete-orphan"
name="productSkuPrices">
  <key column="PRODUCT_SKU_UID"/>
  <map-key type="java.util.Currency" column="CURRENCY" />
  <one-to-many class="com.elasticpath.domain.catalog.SkuPrice"/>
</map>
```

### Correct Example

```
<map lazy="false" cascade="all,delete-orphan"
name="productSkuPrices">
  <key column="PRODUCT_SKU_UID" not-null="true"/>
  <map-key type="java.util.Currency" column="CURRENCY" />
  <one-to-many class="com.elasticpath.domain.catalog.SkuPrice"/>
</map>
```

## Database Compatibility Issues

This page lists database compatability issues and other database-specific considerations.

### Oracle

**Empty strings are considered null**

Oracle considers the empty String ("") to be a null value. If you assign the empty string to a column that is set to not-null, the database operation will fail.

**Query Length Limitation**

Oracle allows only 1000 expressions in a single query. Therefore, large queries with over 1000 expressions like "select ... from ... where ... in (..., ...)" must be separated into multiple queries. The following code from ProductServiceImpl.findByUids() shows the recommended way to accomplish this.

```
final List queries = this.getUtility().
  composeQueries(Criteria.PRODUCT_BY_UIDS,
  Criteria.PLACE_HOLDER_FOR_LIST, productUids);
final List result = getPersistenceEngine().retrieve(queries);
```

**Ampersand(&) cannot be used in SQL scripts.**

```
INSERT INTO TBRAND (UIDPK,CODE,GUID)
VALUES (1110,'D&H', 'D&H');
```

must be changed to the following.

```
INSERT INTO TBRAND (UIDPK,CODE,GUID)
VALUES (1110,concat('D', concat(CHR(38), 'H')),
        concat('D', concat(CHR(38), 'H')));
```

**Literal date values must use the TO_DATE function**

Literal date values like '2006-11-11 11:11:11' cannot be used in Oracle SQL scripts. Instead, the TO_DATE function must be used as shown below.

```
TO_DATE('2005-06-10','YYYY-MM-DD')
```

## Service-Layer Database Transactions

Database transaction behavior is defined on a per-method basis in the service.xml Spring configuration file. Service classes with methods that are to be run as transactions inherit the parent bean named "txProxyTemplate" in service.xml. txProxyTemplate has a property called "transactionAttributes" that defines the nature of transactions for methods matching particular name patterns. For example, the property <prop key="update*">PROPAGATION_REQUIRED</prop> declares that any method beginning with "update" in a service method of a bean whose parent is "txProxyTemplate" will be run in a transaction. The following transaction attributes are in use:

- PROPAGATION_REQUIRED - Supports a transaction if one already exists. If there is no transaction a new one is started.
- PROPAGATION_NEVER - Does not execute as a transaction and throws an exception if one already exists.
- PROPAGATION_SUPPORTS - A new transaction will not be started to run the method, however if a transaction is in progress it will propagate to include the call to this method as well.

The following attributes are also available:

- PROPAGATION_MANDATORY - Throws an exception if there is no active transaction
- PROPAGATION_NOT_SUPPORTED - Executes non-transactionally and suppends any existing transaction
- PROPAGATION_REQUIRES_NEW - Always starts a new transaction. If an active transaction exists, it is suspended.
- PROPAGATION_NESTED - Runs as a nested transaction if one exists, or starts a new one.

Note that all methods that access the database must have names that match one of the transaction attribute properties defined in txProxyTemplate or an exception will be thrown at runtime.

# Performance and Scalability

This document provides information and guidelines for developing scalable high-performance applications with AquaLogic Commerce Services. We will focus on issues and considerations that arise during development. For performance issues relating to deployment and configuration, please see Configuring AquaLogic Commerce Services for optimal performance in the Deployment Guide.

The first section covers general performance issues optimizations of AquaLogic Commerce Services. The Storefront and Commerce Manager sections focus on topics that are specific to those applications. Theses sections are followed by a set of performance tips for general Java development.

## *General*

This section covers general performance topics for the AquaLogic Commerce Services applications.

### Database

#### Create indexes on condition columns for faster queries

Creating an index on columns containing values used in query conditions can improve the performance of your queries. While this may introduce some overhead when updating data, query performance is usually more important in AquaLogic Commerce Services.

#### Denormalize tables to reduce joins

Normalized table designs reduce duplicate data, but this is usually at the expense of performance. Querying normalized data typically requres queries to join several tables, which can be slow. For this reason, judicious denormalization can be beneficial for performance critical query performance. For example, in AquaLogic Commerce Services a column called LOCALIZED_ATTRIBUTE_KEY exists in the table TPRODUCTATTIRIBUTEVALUE to avoid the need to join with TATTRIBUTE to retrieve the key.

#### Hibernate

The following Hibernate properties are of interest when tuning performance. These property values are set in hibernate-config.xml in the <property name="hibernateProperties"> block.

- **hibernate.show_sql** - Writes all SQL statements to the console (Set to false for production).
- **hibernate.generate_statistics** - If enabled, Hibernate will collect statistics which are helpful for performance tuning (Disable for production).
- **hibernate.jdbc.fetch_size** - Sets the number of rows that should be fetched from the database each time new rows are needed. Normally the most efficient fetch size is already the default for the driver. This setting simply allows a programmer to experiment to see if a certain fetch size is more efficient than the default for a particular application.

- **hibernate.jdbc.batch_size** - A non-zero value enables use of JDBC2 batch updates by Hibernate. The recommended value is 20.

### Use the Commerce Manager to perform batch jobs

Statistics and customer profiling data should be calculated as a batch job run by the Commerce Manager rather than be computed on the fly during Storefront transactions. These jobs are typically scheduled to run during times of low load using the Quartz scheduler. The top seller and product recommendation calculations are good examples of intensive computations that are performed as Commerce Manager batch jobs to conserve Storefront CPU resources.

### Use Lucene for faster searches

Use Lucene index search to achieve faster search results. Lucene is particularly well suited for full text search and will perform much faster than a database query.

### Storefront

The following performance topics apply only to the Storefront application.

- **Domain objects caching mode** - Read-only is the simplest and best performing strategy. Most of the domain objects are cached in read-only mode in the Storefront because the Storefront never updates catalog data (except the inventory).
- **IsolationLevel** - By default, AquaLogic Commerce Services uses the IsolationLevel READ_COMMITED. This provides the required data consistency and acceptable performance.
- **Locking mode** - Optimistic locking was added into Hibernate 3.2. It consumes more memory in return for better concurrency. In AquaLogic Commerce Services, most objects are cached as read-only, so the pessimistic locking is used instead to conserve memory.

### AquaLogic Commerce Services Product Cache

Although the Hibernate 2nd level cache can reduce database accesses, it doesn't reduce memory consumption. If two threads retrieve the same product through hibernate, they will get different instances of the product. A product is one of the most expensive object to compose from both a CPU perspective and a memory perspective. It is memory intensive because it aggregates a large amount of data such as prices, skus, attributes, inventory and recommendations etc. It is CPU intensive because promotion rules need to be applied to it to get the promotion price. It is quite natural to cache expensive products and share them across threads to improve performance. AquaLogic Commerce Services implements a custom cache in the Storefront for this purpose. The current implementation will only cache one product instance and share it across all threads. All catalog promotions are applied to products retrieved from this cache. Be default, the LRU eviction policy is used to keep cached products up-to-date.

The Storefront product cache is enabled by using the SingleCacheRetrievalStrategy as the retrieval strategy wired into service objects in the serviceSF.xml configuration file.

Note that the product domain object is not thread safe. It's not necessary for it to be thread safe because it is intended to be read-only and the lack of synchronization gives it better performance. However, it is important to remember this and not attempt to make changes to products returned from the singleCachingProductRetrieveSrategy.

Because the products are shared, this cache can only be used with the everyone eligibility in catalog promotions. If you require the use of other eligibilities, you will need to disable the product cache by using another retrieval strategy. This will result in significantly slower performance for catalog browsing pages in the Storefront.

### Use load tuners to retrieve only the data you need

Some domain objects such as Product aggregate a large amount of data such as prices, skus, attributes, inventory and recommendations etc. For a particular page, it's normally not necessary to load all of the aggregated data. For example, in search result page we only show the main information of a product such as its name and price. The attributes and product recommendations are not used. In this case, a load tuner can be used to instruct Hibernate on which collections should be loaded. The load tuner is typically passed to the service layer along with the id of the desired parent object. There are several pre-defined load tuners declared in domainModel.xml, but you will often wish to create your own tuner to load exactly the data you need.

### Attribute performance overhead

The AquaLogic Commerce Services attribute system provides a flexible way for business users to add or change category or product information. However, developers should be aware of the overhead of using attributes. A database table join and a Java map lookup is required to retrieve an attribute value for a category or product. Therefore, the number of attributes used should be limited in peformance-critical situations.

### Only store critically necessary data in the HTTP session

It's always good for scalability and performance to make the HTTP session lighter because thousands of sessions may get created in an application server at peak time.
Consider the following calculations as an example.

```
Session timeout : 30 minutes
Average customer visit time : 5 minutes
Average customer arrival rate : 10 users / second
Average session size : 100Kbyte / session
Total sessions =  10 users / sec * 60 sec / minute
        * (30 minutes + 5 minutes) = 21,000 sessions
Total memory consumed by sessions = 100Kbyte / session
        * 21,000 sessions = 2.1GB
```

From the calculations above, we can see how critical it is to limit the data in the session because of its overall effect on memory usage. Furthermore, many of the objects stored in the session are long-lived and have a high chance of being pushed to older segments of JVM memory where they

are more expensive to garbage collect. A site's throughput to drop severely when the available memory is consumed and the JVN begins performing full garbage collections.

When you consider storing additional data in the session, the following questions can help you determine if this is really necessary.

- How much memory will the data consume when multiplied across all sessions?
- Can this data be shared by customers? if not, is it acceptable to have thousands of copies of this data?
- What is the probability that successive requests will use this data?
- If it is not stored in the session, is there a cheap way for successive requests to generate or retrieve this data when they require it?

### Cache clustering

AquaLogic Commerce Services does not use cache clustering. Cache clustering is complex, error prone, and poor performing. Since most objects are cached as read-only in the Storefront, there is no need to cluster the cache.

### Optimize load balancer polls

If you are using a load balancer that makes periodic requests to check on a server's health, this can be a source of performance problems. Each time a request is made, a CustomerSession object is created to track information about the session and potential customer. If the load balancer poll repeatedly makes requests, a high number of unnecessary customer sessions may be created. For this reason, create a specialized page to handle polls without creating any session data.

## Java code performance tips

### Use Collections.EMPTY_LIST, Collections.EMPTY_SET, Collections.EMPTY_MAP

If a method returns a list, set or map, it should return an empty list, set or map rather than null. If the returned list, set or map is just for reference (which should be in most cases), you should consider using Collections.EMPTY_LIST, Collections.EMPTY_SET and Collections.EMPTY_MAP instead of creating a new collection. Theses Collections constants are immutable instances and can be used to avoid unnecessary memory allocation.

```
public Collection getProductUids() {
  if (topSellerProducts == null) {
    return Collections.EMPTY_SET;
  }
  return topSellerProducts.keySet();
}
```

## Map Iteration

Avoid iterating on maps in the way shown below.

```
for (Iterator keyIter = mapToSort.keySet().iterator();
keyIter.hasNext();) {
  String currKey = (String) keyIter.next();
  String currValue = (String) mapToSort.get(currKey);
  reverseMap.put(currValue, currKey);
  sortedValueSet.add(currValue);
}
```

This method accesses the value of a Map entry using a key that was retrieved from a keySet iterator. It is more efficient to use an iterator on the entrySet of the map to avoid the Map.get(key) lookup. Use the following code instead.

```
for (Iterator iter = mapToSort.entrySet().iterator(); iter.hasNext();)
{
  final Entry entry = (Entry) iter.next();
  final String currKey = (String) entry.getKey();
  final String currValue = (String) entry.getValue();
    reverseMap.put(currValue, currKey);
  sortedValueSet.add(currValue);
}
```

## Explicit garbage collection

```
Runtime.getRuntime().gc();
```

Explicit requests for garbage collection as shown above is abnormal. This should only be used in benchmarking or profiling code. Explicitly invoking the garbage collector in routines such as close or finalize methods can lead to extremely poor performance. Garbage collection can be expensive and any situation that forces hundreds or thousands of garbage collections will bring a machine to a crawl.

## Concatenating Strings using + in a loop

Concatenating Strings using + is slower than using StringBuffer.
Slow example:

```
for (int i = 0; i < optionValueCodes.size(); i++) {
  query += "optVal.optionValueKey = '" + (String)
optionValueCodes.get(i) + "' ";
  if (i < optionValueCodes.size() - 1) {
   query += "or ";
  }
}
```

Better example:

```
final StringBuffer sbf = new StringBuffer("Select sku.uidPk
from ProductSku as sku inner join sku.optionValueMap as optVal where
(");

for (int i = 0; i < optionValueCodes.size(); i++) {
  sbf.append("optVal.optionValueKey = '")
     .append((String) optionValueCodes.get(i)).append("' ");
  if (i < optionValueCodes.size() - 1) {
    sbf.append("or ");
  }
}
```

### Use static inner classes when possible

If you have an inner class that does not use its embedded reference to the object which created it, it can be declared static. The reference makes instances of the class larger and may keep the reference to the creator object alive longer than necessary. If possible, the class should be made static.

# File System Structure

This document describes the contents of key directories in the AquaLogic Commerce Services file system structure.

## *War file structure*

The directory structure of the Storefront, and Commerce Manager applications are quite similar and typically follow the structure below.

- **target** - Contains the output of the project, which may be a Jar file or WAR file
- **template-resources** - Contains JavaScript and Dojo resources, see "template-resources structure" below
- **WEB-INF** - Contains web application resources that are not directly available to web browsers
    - **build-stats** - Reports generated by ant scripts
    - **classes** - Compiled source code
    - **conf** - Configuration files, see "Conf structure" below
    - **lib** - Jar libraries required by the project
    - ***Index** - Directories ending with "Index" contain Lucene search index data files
    - **src** - Source code
        - **main** - Production Java source code

        - **test** - JUnit test code
    - **templates** - Velocity templates
    - **report** - BIRT report designs (Commerce Manager Only)

## *conf structure*

The following structure appears in WEB-INF/conf in the AquaLogic Commerce Services projects.

- **misc** - Miscellaneous configuration
    - **validation** - Commons Validator configuration
    - **velocity** - Velocity toolbox configuration
- **resources** - Properties files for Lucene, payment processors, country configuration
- **spring** - Spring configuration
    - **dataaccess** - Spring configuration for Data Access Objects (DAO), hibernate mapping files, hibernate configuration
    - **models** - Spring configuration for domain model objects and form beans
    - **scheduling** - Quartz configuration
    - **security** - Acegi Security configuration
    - **service** - Spring configuration for service objects
    - **views** - Velocity configuration
    - **web** - Spring MVC Controller and URL-mapping configuration

## *template-resources structure*

The following structure appears in template-resources in the AquaLogic Commerce Services Storefront and Commerce Manager projects.

- **template-resources** - Contains JavaScript and Dojo resources
    - **dojo** - The Dojo toolkit
    - **ep-dojo** - Customizations to Dojo
    - **yui** - Yahoo UI widgets, e.g. Calendar
    - **elasticpath** - Widget extensions primarily used in the Commerce Manager UI
        - **controller** - JavaScript controllers
        - **loader** - JavaScript loaders
        - **widget** - JavaScript UI widgets

In the Commerce Manager project, the majority of the user interface is implemented with Dojo widgets structured as above. In the Storefront, Dojo widgets in a similar file system structure implement the AquaLogic Commerce Services One Page feature.

# Database Reference

This section serves as a reference guide for the AquaLogic Commerce Services database model.

## *Data Model*

Any given database data model displays the relationship among database tables in the schema, for a particular topic. The data models are grouped in the following categories:

- Catalog
- Customer
- Import
- Orders
- Roles
- Rules
- Shipping
- Taxes

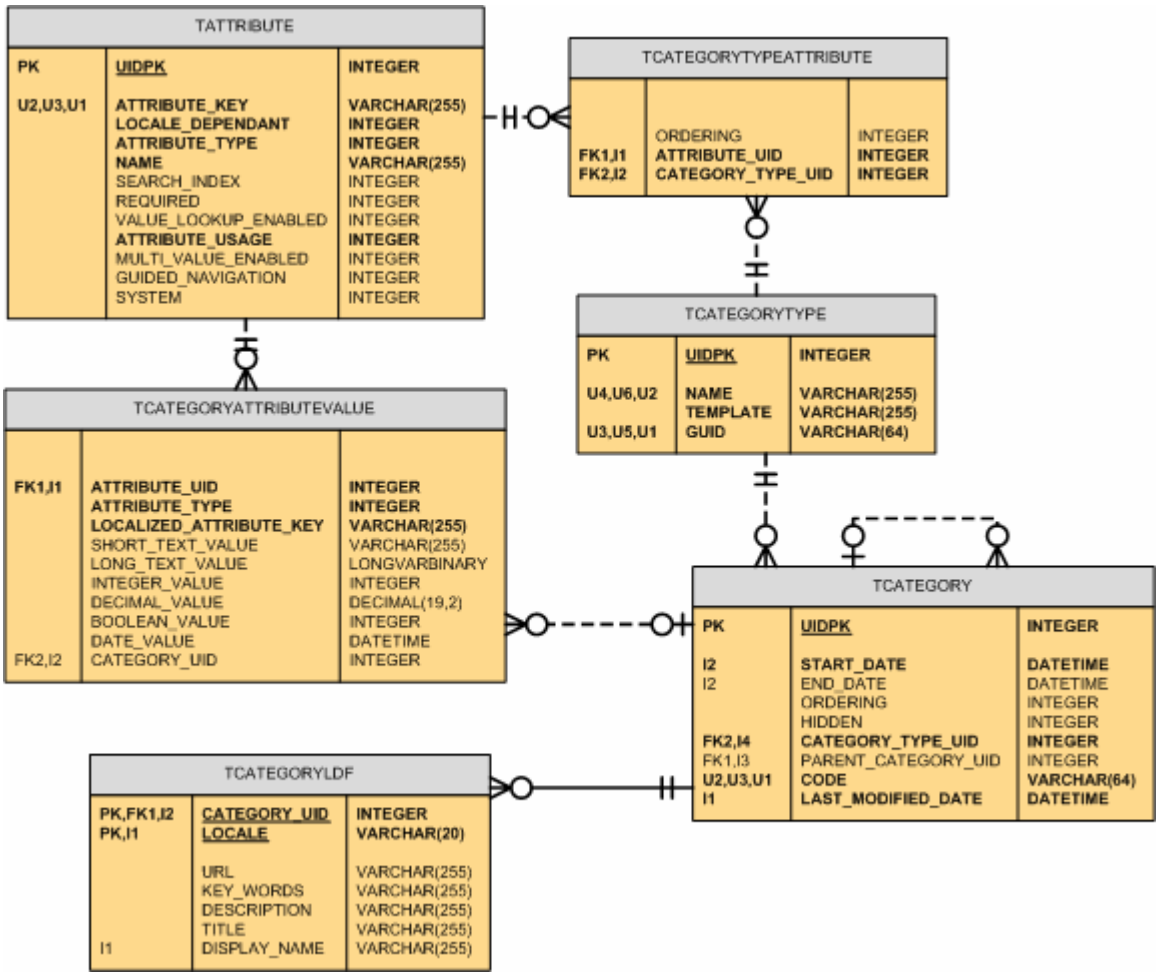
For the larger data model source graphics, see http://edocs.bea.com/alcs/docs51/pdf/alcs_data_model.zip in this document's directory.

### Catalog

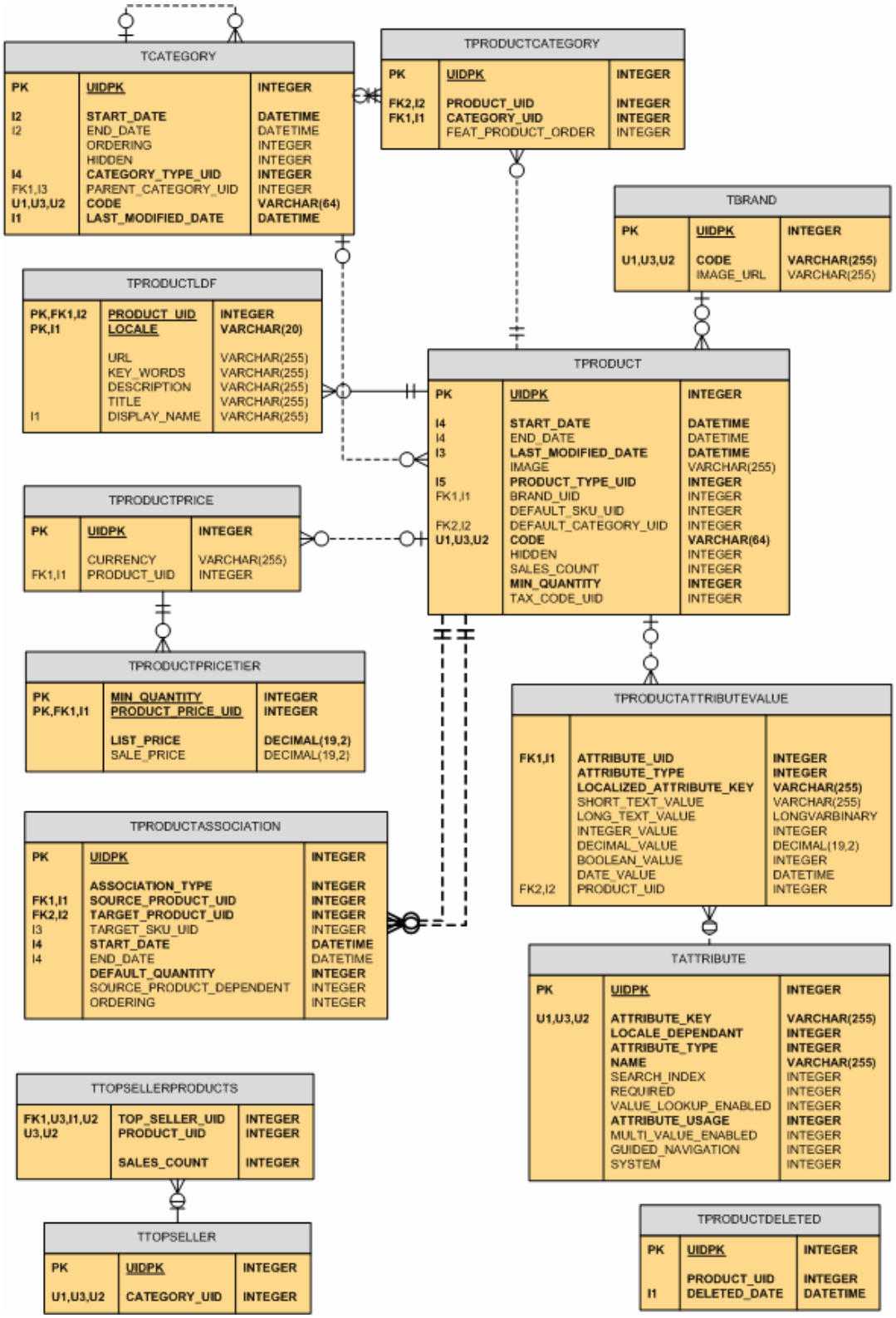
The data model diagrams related to the Catalog subsystem are:

- Category
- Product
- Product SKU

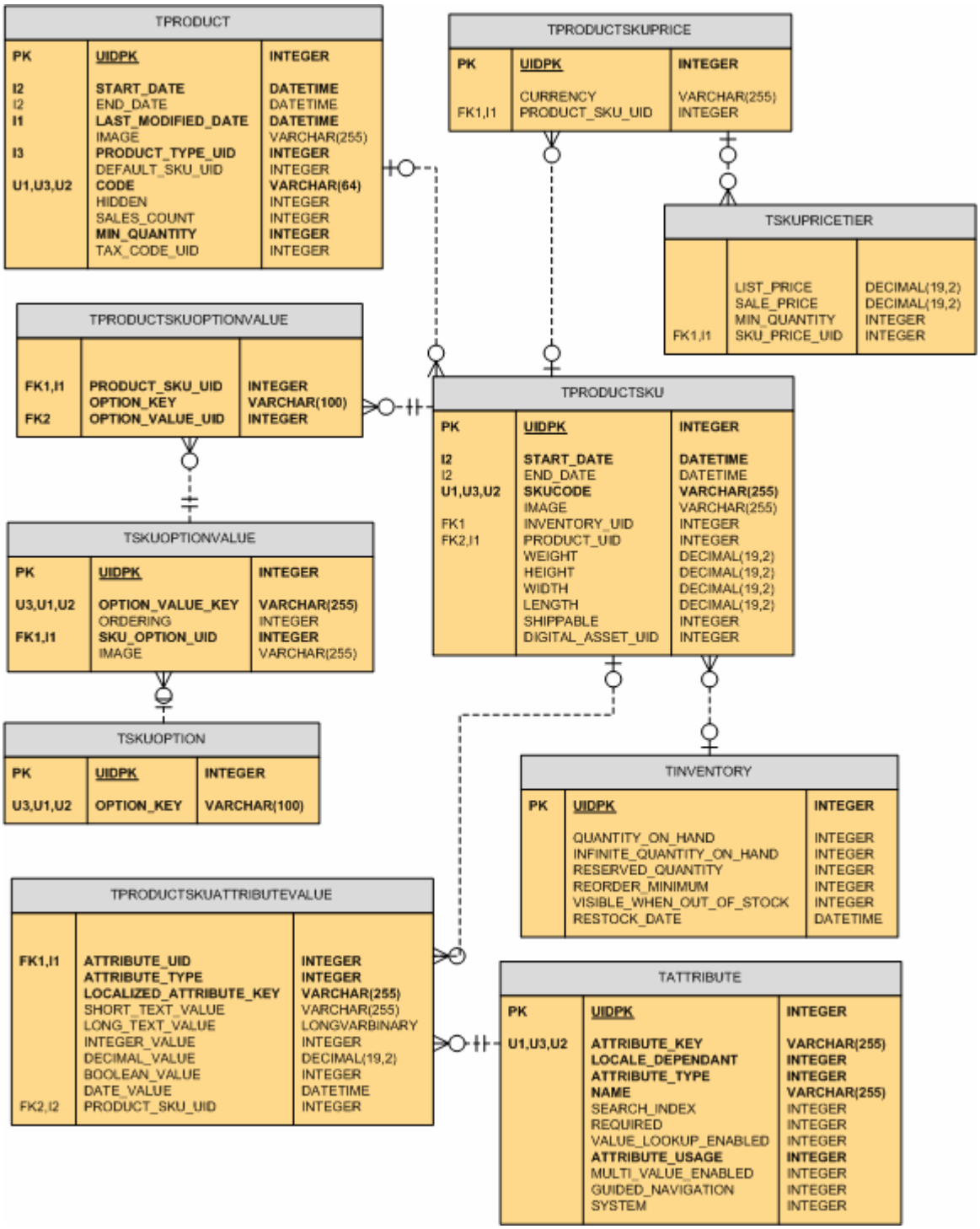
## Category


TATTRIBUTE
PK
UIDPK
INTEGER
U2,U3,U1
ATTRIBUTE_KEY
VARCHAR(255)
LOCALE_DEPENDANT
INTEGER
ATTRIBUTE_TYPE
INTEGER
NAME
VARCHAR(255)
SEARCH_INDEX
INTEGER
REQUIRED
INTEGER
VALUE_LOOKUP_ENABLED
INTEGER
ATTRIBUTE_USAGE
INTEGER
MULTI_VALUE_ENABLED
INTEGER
GUIDED_NAVIGATION
INTEGER
SYSTEM
INTEGER
TCATEGORYTYPEATTRIBUTE
ORDERING
INTEGER
FK1,I1
ATTRIBUTE_UID
INTEGER
FK2,I2
CATEGORY_TYPE_UID
INTEGER
TCATEGORYTYPE
PK
UIDPK
INTEGER
U4,U6,U2
NAME
VARCHAR(255)
TEMPLATE
VARCHAR(255)
U3,U5,U1
GUID
VARCHAR(64)
TCATEGORYATTRIBUTEVALUE
FK1,I1
ATTRIBUTE_UID
INTEGER
ATTRIBUTE_TYPE
INTEGER
LOCALIZED_ATTRIBUTE_KEY
VARCHAR(255)
SHORT_TEXT_VALUE
VARCHAR(255)
LONG_TEXT_VALUE
LONGVARBINARY
INTEGER_VALUE
INTEGER
DECIMAL_VALUE
DECIMAL(19,2)
BOOLEAN_VALUE
INTEGER
DATE_VALUE
DATETIME
FK2,I2
CATEGORY_UID
INTEGER
TCATEGORY
PK
UIDPK
INTEGER
I2
START_DATE
DATETIME
I2
END_DATE
DATETIME
ORDERING
INTEGER
HIDDEN
INTEGER
FK2,I4
CATEGORY_TYPE_UID
INTEGER
FK1,I3
PARENT_CATEGORY_UID
INTEGER
U2,U3,U1
CODE
VARCHAR(64)
I1
LAST_MODIFIED_DATE
DATETIME
TCATEGORYLDF
PK,FK1,I2
CATEGORY_UID
INTEGER
PK,I1
LOCALE
VARCHAR(20)
URL
VARCHAR(255)
KEY_WORDS
VARCHAR(255)
DESCRIPTION
VARCHAR(255)
TITLE
VARCHAR(255)
I1
DISPLAY_NAME
VARCHAR(255)

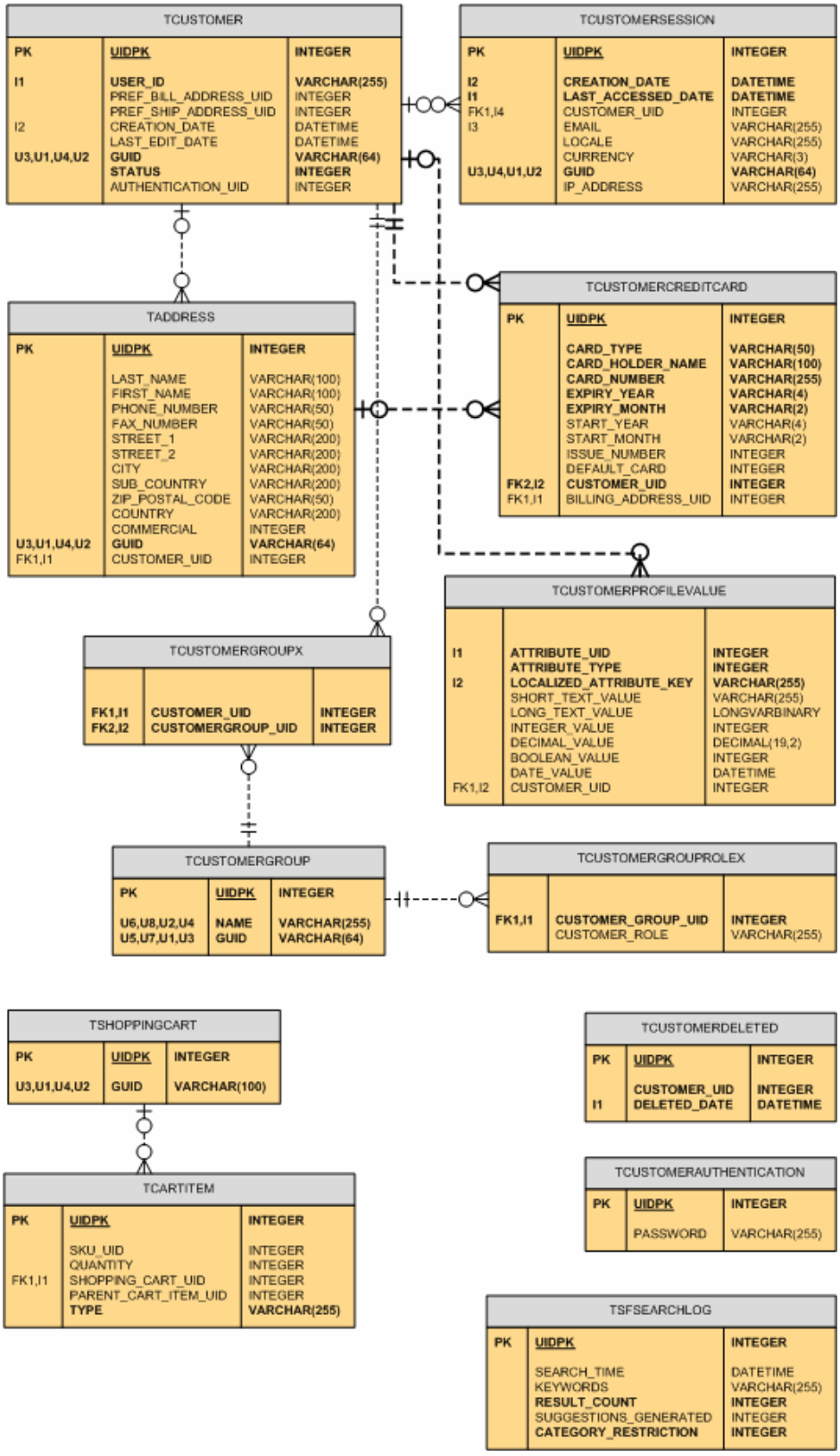
## Product


TCATEGORY
PK UIDPK INTEGER
I2 START_DATE DATETIME
I2 END_DATE DATETIME
ORDERING INTEGER
HIDDEN INTEGER
I4 CATEGORY_TYPE_UID INTEGER
FK1,I3 PARENT_CATEGORY_UID INTEGER
U1,U3,U2 CODE VARCHAR(64)
I1 LAST_MODIFIED_DATE DATETIME
TPRODUCTCATEGORY
PK UIDPK INTEGER
FK2,I2 PRODUCT_UID INTEGER
FK1,I1 CATEGORY_UID INTEGER
FEAT_PRODUCT_ORDER INTEGER
TBRAND
PK UIDPK INTEGER
U1,U3,U2 CODE VARCHAR(255)
IMAGE_URL VARCHAR(255)
TPRODUCTLDF
PK,FK1,I2 PRODUCT_UID INTEGER
PK,I1 LOCALE VARCHAR(20)
URL VARCHAR(255)
KEY_WORDS VARCHAR(255)
DESCRIPTION VARCHAR(255)
TITLE VARCHAR(255)
I1 DISPLAY_NAME VARCHAR(255)
TPRODUCT
PK UIDPK INTEGER
I4 START_DATE DATETIME
I4 END_DATE DATETIME
I3 LAST_MODIFIED_DATE DATETIME
IMAGE VARCHAR(255)
I5 PRODUCT_TYPE_UID INTEGER
FK1,I1 BRAND_UID INTEGER
DEFAULT_SKU_UID INTEGER
FK2,I2 DEFAULT_CATEGORY_UID INTEGER
U1,U3,U2 CODE VARCHAR(64)
HIDDEN INTEGER
SALES_COUNT INTEGER
MIN_QUANTITY INTEGER
TAX_CODE_UID INTEGER
TPRODUCTPRICE
PK UIDPK INTEGER
CURRENCY VARCHAR(255)
FK1,I1 PRODUCT_UID INTEGER
TPRODUCTPRICETIER
PK MIN_QUANTITY INTEGER
PK,FK1,I1 PRODUCT_PRICE_UID INTEGER
LIST_PRICE DECIMAL(19,2)
SALE_PRICE DECIMAL(19,2)
TPRODUCTATTRIBUTEVALUE
FK1,I1 ATTRIBUTE_UID INTEGER
ATTRIBUTE_TYPE INTEGER
LOCALIZED_ATTRIBUTE_KEY VARCHAR(255)
SHORT_TEXT_VALUE VARCHAR(255)
LONG_TEXT_VALUE LONGVARBINARY
INTEGER_VALUE INTEGER
DECIMAL_VALUE DECIMAL(19,2)
BOOLEAN_VALUE INTEGER
DATE_VALUE DATETIME
FK2,I2 PRODUCT_UID INTEGER
TPRODUCTASSOCIATION
PK UIDPK INTEGER
ASSOCIATION_TYPE INTEGER
FK1,I1 SOURCE_PRODUCT_UID INTEGER
FK2,I2 TARGET_PRODUCT_UID INTEGER
I3 TARGET_SKU_UID INTEGER
I4 START_DATE DATETIME
I4 END_DATE DATETIME
DEFAULT_QUANTITY INTEGER
SOURCE_PRODUCT_DEPENDENT INTEGER
ORDERING INTEGER
TATTRIBUTE
PK UIDPK INTEGER
U1,U3,U2 ATTRIBUTE_KEY VARCHAR(255)
LOCALE_DEPENDANT INTEGER
ATTRIBUTE_TYPE INTEGER
NAME VARCHAR(255)
SEARCH_INDEX INTEGER
REQUIRED INTEGER
VALUE_LOOKUP_ENABLED INTEGER
ATTRIBUTE_USAGE INTEGER
MULTI_VALUE_ENABLED INTEGER
GUIDED_NAVIGATION INTEGER
SYSTEM INTEGER
TTOPSELLERPRODUCTS
FK1,U3,I1,U2 TOP_SELLER_UID INTEGER
U3,U2 PRODUCT_UID INTEGER
SALES_COUNT INTEGER
TTOPSELLER
PK UIDPK INTEGER
U1,U3,U2 CATEGORY_UID INTEGER
TPRODUCTDELETED
PK UIDPK INTEGER
PRODUCT_UID INTEGER
I1 DELETED_DATE DATETIME

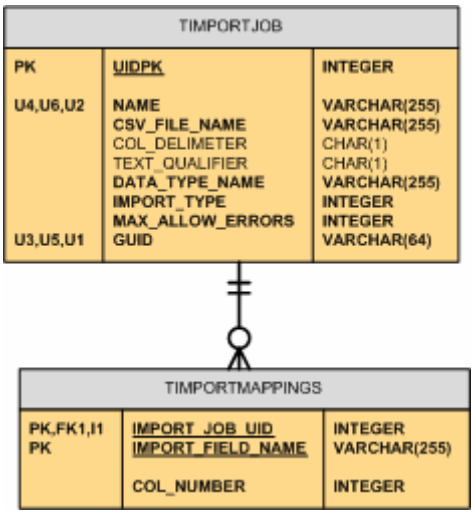
## Product SKU


TPRODUCT
PK UIDPK INTEGER
I2 START_DATE DATETIME
I2 END_DATE DATETIME
I1 LAST_MODIFIED_DATE DATETIME
IMAGE VARCHAR(255)
I3 PRODUCT_TYPE_UID INTEGER
DEFAULT_SKU_UID INTEGER
U1,U3,U2 CODE VARCHAR(64)
HIDDEN INTEGER
SALES_COUNT INTEGER
MIN_QUANTITY INTEGER
TAX_CODE_UID INTEGER
TPRODUCTSKUPRICE
PK UIDPK INTEGER
CURRENCY VARCHAR(255)
FK1,I1 PRODUCT_SKU_UID INTEGER
TSKUPRICETIER
LIST_PRICE DECIMAL(19,2)
SALE_PRICE DECIMAL(19,2)
MIN_QUANTITY INTEGER
FK1,I1 SKU_PRICE_UID INTEGER
TPRODUCTSKUOPTIONVALUE
FK1,I1 PRODUCT_SKU_UID INTEGER
OPTION_KEY VARCHAR(100)
FK2 OPTION_VALUE_UID INTEGER
TPRODUCTSKU
PK UIDPK INTEGER
I2 START_DATE DATETIME
I2 END_DATE DATETIME
U1,U3,U2 SKUCODE VARCHAR(255)
IMAGE VARCHAR(255)
FK1 INVENTORY_UID INTEGER
FK2,I1 PRODUCT_UID INTEGER
WEIGHT DECIMAL(19,2)
HEIGHT DECIMAL(19,2)
WIDTH DECIMAL(19,2)
LENGTH DECIMAL(19,2)
SHIPPABLE INTEGER
DIGITAL_ASSET_UID INTEGER
TSKUOPTIONVALUE
PK UIDPK INTEGER
U3,U1,U2 OPTION_VALUE_KEY VARCHAR(255)
ORDERING INTEGER
FK1,I1 SKU_OPTION_UID INTEGER
IMAGE VARCHAR(255)
TSKUOPTION
PK UIDPK INTEGER
U3,U1,U2 OPTION_KEY VARCHAR(100)
TINVENTORY
PK UIDPK INTEGER
QUANTITY_ON_HAND INTEGER
INFINITE_QUANTITY_ON_HAND INTEGER
RESERVED_QUANTITY INTEGER
REORDER_MINIMUM INTEGER
VISIBLE_WHEN_OUT_OF_STOCK INTEGER
RESTOCK_DATE DATETIME
TPRODUCTSKUATTRIBUTEVALUE
FK1,I1 ATTRIBUTE_UID INTEGER
ATTRIBUTE_TYPE INTEGER
LOCALIZED_ATTRIBUTE_KEY VARCHAR(255)
SHORT_TEXT_VALUE VARCHAR(255)
LONG_TEXT_VALUE LONGVARBINARY
INTEGER_VALUE INTEGER
DECIMAL_VALUE DECIMAL(19,2)
BOOLEAN_VALUE INTEGER
DATE_VALUE DATETIME
FK2,I2 PRODUCT_SKU_UID INTEGER
TATTRIBUTE
PK UIDPK INTEGER
U1,U3,U2 ATTRIBUTE_KEY VARCHAR(255)
LOCALE_DEPENDANT INTEGER
ATTRIBUTE_TYPE INTEGER
NAME VARCHAR(255)
SEARCH_INDEX INTEGER
REQUIRED INTEGER
VALUE_LOOKUP_ENABLED INTEGER
ATTRIBUTE_USAGE INTEGER
MULTI_VALUE_ENABLED INTEGER
GUIDED_NAVIGATION INTEGER
SYSTEM INTEGER

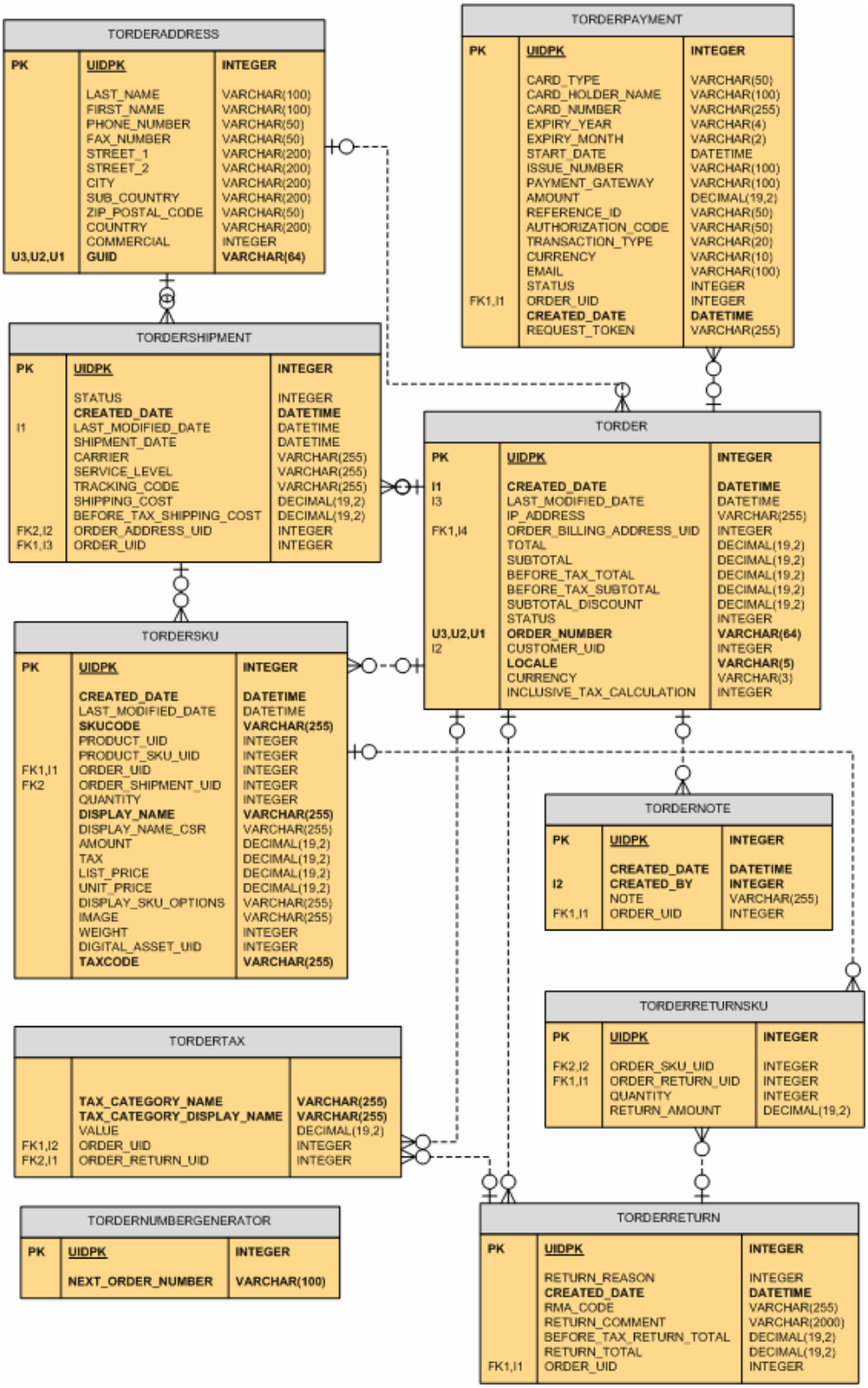
## Customer


TCUSTOMER
PK UIDPK INTEGER
I1 USER_ID VARCHAR(255)
PREF_BILL_ADDRESS_UID INTEGER
PREF_SHIP_ADDRESS_UID INTEGER
I2 CREATION_DATE DATETIME
LAST_EDIT_DATE DATETIME
U3,U1,U4,U2 GUID VARCHAR(64)
STATUS INTEGER
AUTHENTICATION_UID INTEGER
TCUSTOMERSESSION
PK UIDPK INTEGER
I2 CREATION_DATE DATETIME
I1 LAST_ACCESSED_DATE DATETIME
FK1,I4 CUSTOMER_UID INTEGER
I3 EMAIL VARCHAR(255)
LOCALE VARCHAR(255)
CURRENCY VARCHAR(3)
U3,U4,U1,U2 GUID VARCHAR(64)
IP_ADDRESS VARCHAR(255)
TADDRESS
PK UIDPK INTEGER
LAST_NAME VARCHAR(100)
FIRST_NAME VARCHAR(100)
PHONE_NUMBER VARCHAR(50)
FAX_NUMBER VARCHAR(50)
STREET_1 VARCHAR(200)
STREET_2 VARCHAR(200)
CITY VARCHAR(200)
SUB_COUNTRY VARCHAR(200)
ZIP_POSTAL_CODE VARCHAR(50)
COUNTRY VARCHAR(200)
COMMERCIAL INTEGER
U3,U1,U4,U2 GUID VARCHAR(64)
FK1,I1 CUSTOMER_UID INTEGER
TCUSTOMERCREDITCARD
PK UIDPK INTEGER
CARD_TYPE VARCHAR(50)
CARD_HOLDER_NAME VARCHAR(100)
CARD_NUMBER VARCHAR(255)
EXPIRY_YEAR VARCHAR(4)
EXPIRY_MONTH VARCHAR(2)
START_YEAR VARCHAR(4)
START_MONTH VARCHAR(2)
ISSUE_NUMBER INTEGER
DEFAULT_CARD INTEGER
FK2,I2 CUSTOMER_UID INTEGER
FK1,I1 BILLING_ADDRESS_UID INTEGER
TCUSTOMERPROFILEVALUE
I1 ATTRIBUTE_UID INTEGER
ATTRIBUTE_TYPE INTEGER
I2 LOCALIZED_ATTRIBUTE_KEY VARCHAR(255)
SHORT_TEXT_VALUE VARCHAR(255)
LONG_TEXT_VALUE LONGVARBINARY
INTEGER_VALUE INTEGER
DECIMAL_VALUE DECIMAL(19,2)
BOOLEAN_VALUE INTEGER
DATE_VALUE DATETIME
FK1,I2 CUSTOMER_UID INTEGER
TCUSTOMERGROUPX
FK1,I1 CUSTOMER_UID INTEGER
FK2,I2 CUSTOMERGROUP_UID INTEGER
TCUSTOMERGROUP
PK UIDPK INTEGER
U6,U8,U2,U4 NAME VARCHAR(255)
U5,U7,U1,U3 GUID VARCHAR(64)
TCUSTOMERGROUPROLEX
FK1,I1 CUSTOMER_GROUP_UID INTEGER
CUSTOMER_ROLE VARCHAR(255)
TSHOPPINGCART
PK UIDPK INTEGER
U3,U1,U4,U2 GUID VARCHAR(100)
TCARTITEM
PK UIDPK INTEGER
SKU_UID INTEGER
QUANTITY INTEGER
FK1,I1 SHOPPING_CART_UID INTEGER
PARENT_CART_ITEM_UID INTEGER
TYPE VARCHAR(255)
TCUSTOMERDELETED
PK UIDPK INTEGER
CUSTOMER_UID INTEGER
I1 DELETED_DATE DATETIME
TCUSTOMERAUTHENTICATION
PK UIDPK INTEGER
PASSWORD VARCHAR(255)
TSFSEARCHLOG
PK UIDPK INTEGER
SEARCH_TIME DATETIME
KEYWORDS VARCHAR(255)
RESULT_COUNT INTEGER
SUGGESTIONS_GENERATED INTEGER
CATEGORY_RESTRICTION INTEGER

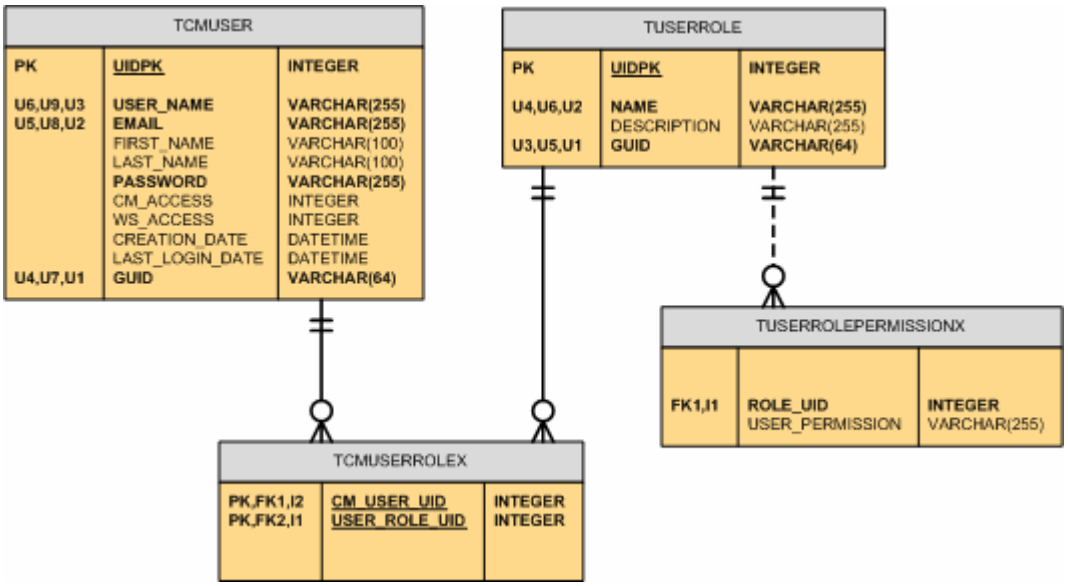
## Import


TIMPORTJOB
PK
UIDPK
INTEGER
U4,U6,U2
NAME
VARCHAR(255)
CSV_FILE_NAME
VARCHAR(255)
COL_DELIMETER
CHAR(1)
TEXT_QUALIFIER
CHAR(1)
DATA_TYPE_NAME
VARCHAR(255)
IMPORT_TYPE
INTEGER
MAX_ALLOW_ERRORS
INTEGER
U3,U5,U1
GUID
VARCHAR(64)
TIMPORTMAPPINGS
PK,FK1,I1
IMPORT_JOB_UID
INTEGER
PK
IMPORT_FIELD_NAME
VARCHAR(255)
COL_NUMBER
INTEGER

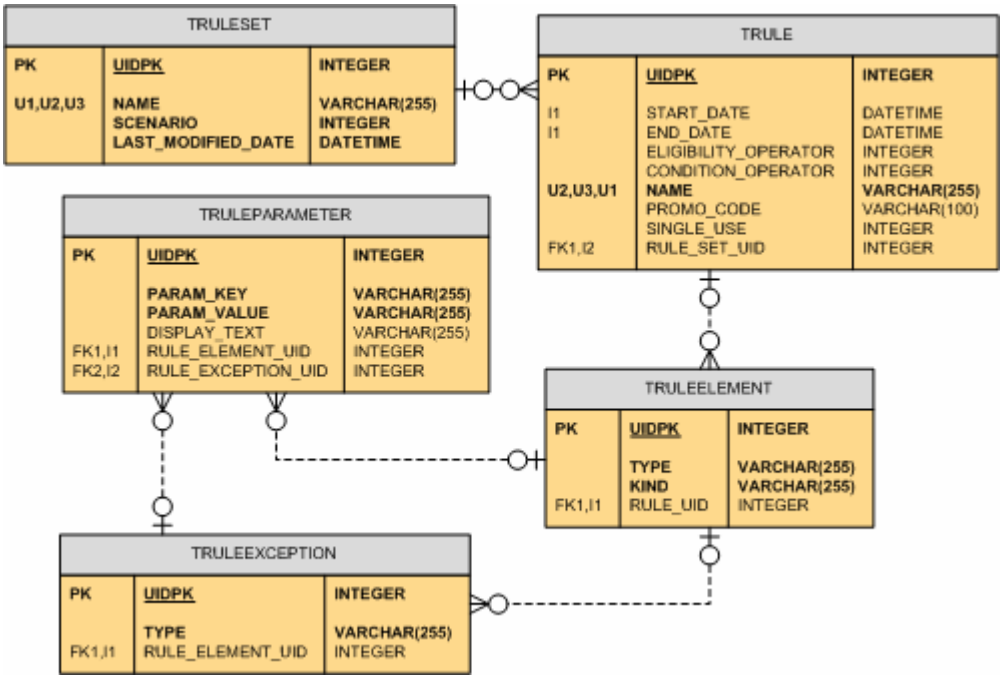
# Orders


TORDERADDRESS
PK UIDPK INTEGER
LAST_NAME VARCHAR(100)
FIRST_NAME VARCHAR(100)
PHONE_NUMBER VARCHAR(50)
FAX_NUMBER VARCHAR(50)
STREET_1 VARCHAR(200)
STREET_2 VARCHAR(200)
CITY VARCHAR(200)
SUB_COUNTRY VARCHAR(200)
ZIP_POSTAL_CODE VARCHAR(50)
COUNTRY VARCHAR(200)
COMMERCIAL INTEGER
U3,U2,U1 GUID VARCHAR(64)
TORDERPAYMENT
PK UIDPK INTEGER
CARD_TYPE VARCHAR(50)
CARD_HOLDER_NAME VARCHAR(100)
CARD_NUMBER VARCHAR(255)
EXPIRY_YEAR VARCHAR(4)
EXPIRY_MONTH VARCHAR(2)
START_DATE DATETIME
ISSUE_NUMBER VARCHAR(100)
PAYMENT_GATEWAY VARCHAR(100)
AMOUNT DECIMAL(19,2)
REFERENCE_ID VARCHAR(50)
AUTHORIZATION_CODE VARCHAR(50)
TRANSACTION_TYPE VARCHAR(20)
CURRENCY VARCHAR(10)
EMAIL VARCHAR(100)
STATUS INTEGER
FK1,I1 ORDER_UID INTEGER
CREATED_DATE DATETIME
REQUEST_TOKEN VARCHAR(255)
TORDERSHIPMENT
PK UIDPK INTEGER
STATUS INTEGER
CREATED_DATE DATETIME
I1 LAST_MODIFIED_DATE DATETIME
SHIPMENT_DATE DATETIME
CARRIER VARCHAR(255)
SERVICE_LEVEL VARCHAR(255)
TRACKING_CODE VARCHAR(255)
SHIPPING_COST DECIMAL(19,2)
BEFORE_TAX_SHIPPING_COST DECIMAL(19,2)
FK2,I2 ORDER_ADDRESS_UID INTEGER
FK1,I3 ORDER_UID INTEGER
TORDER
PK UIDPK INTEGER
I1 CREATED_DATE DATETIME
I3 LAST_MODIFIED_DATE DATETIME
IP_ADDRESS VARCHAR(255)
FK1,I4 ORDER_BILLING_ADDRESS_UID INTEGER
TOTAL DECIMAL(19,2)
SUBTOTAL DECIMAL(19,2)
BEFORE_TAX_TOTAL DECIMAL(19,2)
BEFORE_TAX_SUBTOTAL DECIMAL(19,2)
SUBTOTAL_DISCOUNT DECIMAL(19,2)
STATUS INTEGER
U3,U2,U1 ORDER_NUMBER VARCHAR(64)
I2 CUSTOMER_UID INTEGER
LOCALE VARCHAR(5)
CURRENCY VARCHAR(3)
INCLUSIVE_TAX_CALCULATION INTEGER
TORDERSKU
PK UIDPK INTEGER
CREATED_DATE DATETIME
LAST_MODIFIED_DATE DATETIME
SKUCODE VARCHAR(255)
PRODUCT_UID INTEGER
PRODUCT_SKU_UID INTEGER
FK1,I1 ORDER_UID INTEGER
FK2 ORDER_SHIPMENT_UID INTEGER
QUANTITY INTEGER
DISPLAY_NAME VARCHAR(255)
DISPLAY_NAME_CSR VARCHAR(255)
AMOUNT DECIMAL(19,2)
TAX DECIMAL(19,2)
LIST_PRICE DECIMAL(19,2)
UNIT_PRICE DECIMAL(19,2)
DISPLAY_SKU_OPTIONS VARCHAR(255)
IMAGE VARCHAR(255)
WEIGHT INTEGER
DIGITAL_ASSET_UID INTEGER
TAXCODE VARCHAR(255)
TORDERNOTE
PK UIDPK INTEGER
CREATED_DATE DATETIME
I2 CREATED_BY INTEGER
NOTE VARCHAR(255)
FK1,I1 ORDER_UID INTEGER
TORDERRETURNSKU
PK UIDPK INTEGER
FK2,I2 ORDER_SKU_UID INTEGER
FK1,I1 ORDER_RETURN_UID INTEGER
QUANTITY INTEGER
RETURN_AMOUNT DECIMAL(19,2)
TORDERTAX
TAX_CATEGORY_NAME VARCHAR(255)
TAX_CATEGORY_DISPLAY_NAME VARCHAR(255)
VALUE DECIMAL(19,2)
FK1,I2 ORDER_UID INTEGER
FK2,I1 ORDER_RETURN_UID INTEGER
TORDERNUMBERGENERATOR
PK UIDPK INTEGER
NEXT_ORDER_NUMBER VARCHAR(100)
TORDERRETURN
PK UIDPK INTEGER
RETURN_REASON INTEGER
CREATED_DATE DATETIME
RMA_CODE VARCHAR(255)
RETURN_COMMENT VARCHAR(2000)
BEFORE_TAX_RETURN_TOTAL DECIMAL(19,2)
RETURN_TOTAL DECIMAL(19,2)
FK1,I1 ORDER_UID INTEGER

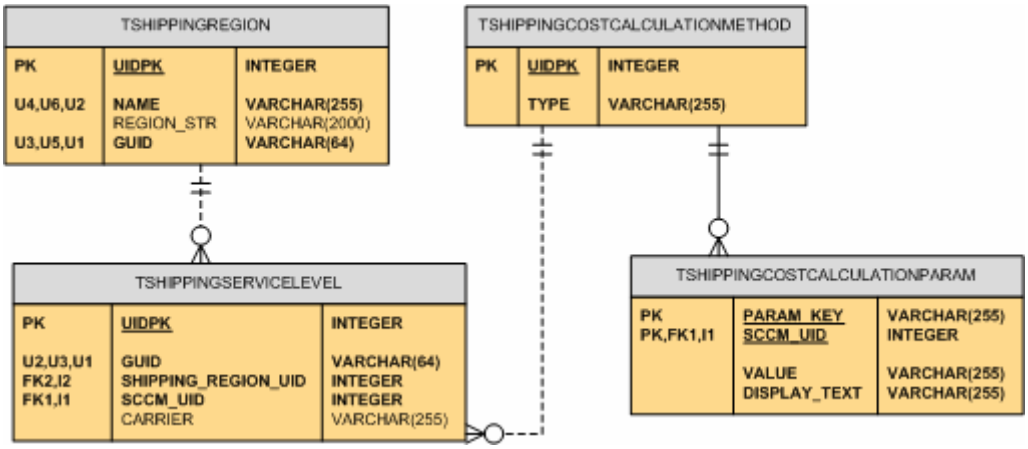
## Roles


TCMUSER
PK
U6,U9,U3
U5,U8,U2
U4,U7,U1
UIDPK
USER_NAME
EMAIL
FIRST_NAME
LAST_NAME
PASSWORD
CM_ACCESS
WS_ACCESS
CREATION_DATE
LAST_LOGIN_DATE
GUID
INTEGER
VARCHAR(255)
VARCHAR(255)
VARCHAR(100)
VARCHAR(100)
VARCHAR(255)
INTEGER
INTEGER
DATETIME
DATETIME
VARCHAR(64)
TUSERROLE
PK
U4,U6,U2
U3,U5,U1
UIDPK
NAME
DESCRIPTION
GUID
INTEGER
VARCHAR(255)
VARCHAR(255)
VARCHAR(64)
TUSERROLEPERMISSIONX
FK1,I1
ROLE_UID
USER_PERMISSION
INTEGER
VARCHAR(255)
TCMUSERROLEX
PK,FK1,I2
PK,FK2,I1
CM_USER_UID
USER_ROLE_UID
INTEGER
INTEGER

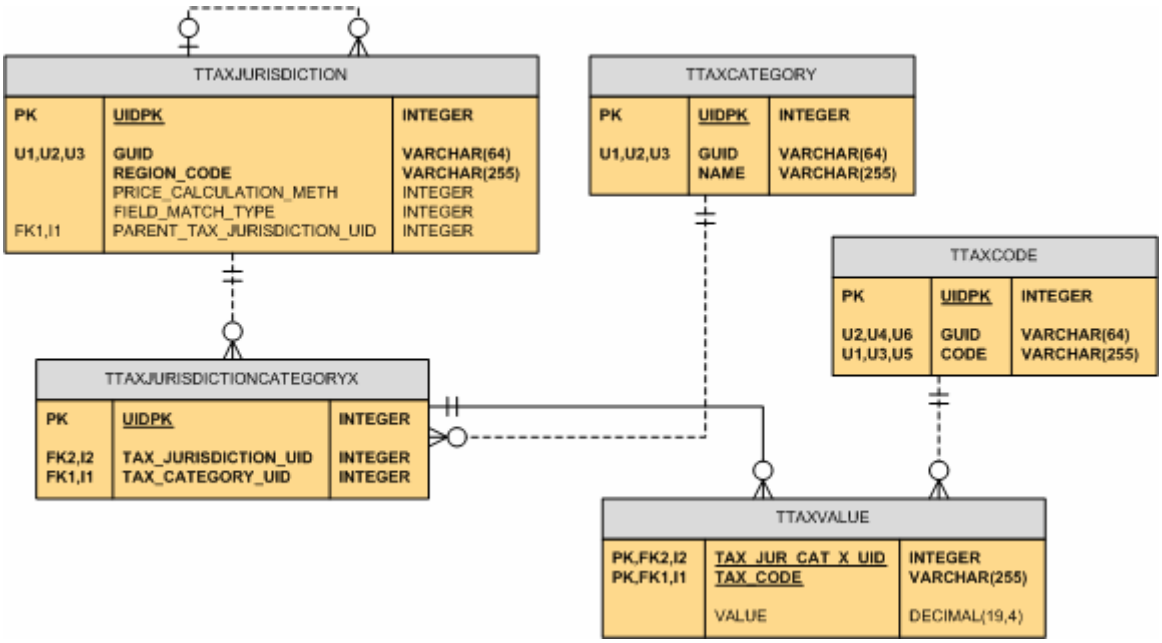
## Rules


TRULESET
PK
U1,U2,U3
UIDPK
NAME
SCENARIO
LAST_MODIFIED_DATE
INTEGER
VARCHAR(255)
INTEGER
DATETIME
TRULE
PK
I1
I1
U2,U3,U1
FK1,I2
UIDPK
START_DATE
END_DATE
ELIGIBILITY_OPERATOR
CONDITION_OPERATOR
NAME
PROMO_CODE
SINGLE_USE
RULE_SET_UID
INTEGER
DATETIME
DATETIME
INTEGER
INTEGER
VARCHAR(255)
VARCHAR(100)
INTEGER
INTEGER
TRULEPARAMETER
PK
FK1,I1
FK2,I2
UIDPK
PARAM_KEY
PARAM_VALUE
DISPLAY_TEXT
RULE_ELEMENT_UID
RULE_EXCEPTION_UID
INTEGER
VARCHAR(255)
VARCHAR(255)
VARCHAR(255)
INTEGER
INTEGER
TRULEELEMENT
PK
FK1,I1
UIDPK
TYPE
KIND
RULE_UID
INTEGER
VARCHAR(255)
VARCHAR(255)
INTEGER
TRULEEXCEPTION
PK
FK1,I1
UIDPK
TYPE
RULE_ELEMENT_UID
INTEGER
VARCHAR(255)
INTEGER

## Shipping




## Taxes



## *Database Tables*

| Table Name | OM Class | Description |
|---|---|---|
| HIBERNATE_UNIQUE_KEYS | HibernateUniqueKeys | Allows Hibernate to generate unqiue primary keys for other DB tables |
| TDIGITALASSETS | Tdigitalassets | This table represents a reference to digital good files stored on the file system. |
| TTAXCATEGORY | Ttaxcategory | This table represents a category of tax, i.e. GST (Canada). |
| TTAXCODE | Ttaxcode | This table represents a TaxCode. For example goods and books. |
| TTAXJURISDICTION | Ttaxjurisdiction | This table represents a TaxJurisdiction, a geographical area i.e. Canada. |
| TTAXJURISDICTIONCATEGORYX | Ttaxjurisdictioncategoryx | This table represents the TaxJurisdiction and TaxCategory association, include the taxValues for each configured TaxCode. |
| TTAXVALUE | Ttaxvalue | This table represents a TaxValue for a given TaxJurisdiction and TaxCategory. |
| TATTRIBUTE | Tattribute | This table reprents a customized property of an object such as Category, Product or Product Sku. |
| TCUSTOMER | Tcustomer | This table |

| Table Name | OM Class | Description |
|---|---|---|
| | | respresents a Customer account. |
| TCUSTOMERAUTHENTICATION | Tcustomerauthentication | This table respresents a Customer authentication. |
| TCUSTOMERPROFILEVALUE | Tcustomerprofilevalue | This table represents the value of a customer profile. |
| TCUSTOMERDELETED | Tcustomerdeleted | This table represents the audit of a deleted Customer. |
| TADDRESS | Taddress | This table represents a Customer Address. |
| TCATEGORYTYPE | Tcategorytype | This table represents the type of a Category, which determines the set of attributes that it has. An example of a category type would be Electronics. |
| TCATEGORY | Tcategory | This table represents a collection of related Products. |
| TCATEGORYDELETED | Tcategorydeleted | This table represents a deleted category. |
| TCATEGORYATTRIBUTEVALUE | Tcategoryattributevalue | This table represents the Attribute Values for a given Category. |
| TCATEGORYLDF | Tcategoryldf | This table contains locale-dependent information about a Category. |
| TCATEGORYTYPEATTRIBUTE | Tcategorytypeattribute | This table represents a mapping of |

| Table Name | OM Class | Description |
|---|---|---|
| | | Category Attributes to a CategoryType |
| TCMUSER | Tcmuser | This table represents a person with an account in the system for accessing the Commerce Manager or web services. |
| TUSERROLE | Tuserrole | This table represents a user''s role. |
| TCMUSERROLEX | Tcmuserrolex | This table represents the UserRoles assigned to a Customer |
| TCUSTOMERGROUP | Tcustomergroup | This table represents a customer group. |
| TCUSTOMERGROUPROLEX | Tcustomergrouprolex | This table represents the Role of a CustomerGroup. |
| TCUSTOMERGROUPX | Tcustomergroupx | This table represents the Customers assigned to a CustomerGroup. |
| TCUSTOMERSESSION | Tcustomersession | This table represents information about customers who may not be logged in. |
| TCUSTOMERCREDITCARD | Tcustomercreditcard | This table represents a credit card stored by a Storefront customer. |
| TIMPORTJOB | Timportjob | This table represents an import job. |
| TIMPORTMAPPINGS | Timportmappings | This table represents the column to field mapping for an |

| Table Name | OM Class | Description |
|---|---|---|
| | | import job. |
| TORDERADDRESS | Torderaddress | This table represents a copy of a customer Address created with an Order. |
| TORDER | Torder | This table represents a customer Order. |
| TORDERNUMBERGENERATOR | Tordernumbergenerator | This table represents the next available order number. |
| TORDERNOTE | Tordernote | This table represents a note made on an order by a CSR. |
| TORDERPAYMENT | Torderpayment | This table represents customer payments made against an Order. |
| TORDERSHIPMENT | Tordershipment | This table represents a customer''s order shipment. |
| TORDERRETURN | Torderreturn | This table represents a customer''s order return. |
| TORDERTAX | Tordertax | This table represents the tax paid on a customers Order. |
| TPRODUCTTYPE | Tproducttype | This table represents the type of a Product, which determines the set of attributes that it has. An example of a product type would be shoes. |
| TBRAND | Tbrand | This table represents product manufacturer/brand |

| Table Name | OM Class | Description |
|---|---|---|
| | | information. |
| TPRODUCT | Tproduct | This table represents represents a merchandise product. A product must have at least 1 ProductSku associated in order to be sold. |
| TORDERSKU | Tordersku | This table represents an order for a quantity of SKUs. |
| TORDERRETURNSKU | Torderreturnsku | This table represents the return of a quantity of SKUs for an order. |
| TPRODUCTATTRIBUTEVALUE | Tproductattributevalue | This table represents the value of a Product Attribute. |
| TPRODUCTCATEGORY | Tproductcategory | This table represents the association between a Category and it''s contained Products. |
| TPRODUCTLDF | TproductIdf | This table contains locale-dependent information about a Product. |
| TPRODUCTPRICE | Tproductprice | This table contains a set of ProductPriceTiers for each configured currency. |
| TPRODUCTPRICETIER | Tproductpricetier | This table contains a set of Product Prices Tiers. |
| TINVENTORY | Tinventory | This table represents the Inventory information for a |

| Table Name | OM Class | Description |
|---|---|---|
| | | ProductSku. |
| TINVENTORYAUDIT | Tinventoryaudit | This table contains an audit of inventory adjustment events. |
| TPRODUCTSKU | Tproductsku | This table represents a variation of a merchandise product. |
| TPRODUCTASSOCIATION | Tproductassociation | This table represents ProductAssociations between two products. Example of product associations are Cross-Sells, Up-Sells, Accessories. |
| TPRODUCTSKUATTRIBUTEVALUE | Tproductskuattributevalue | This table represents the Attribute Values for a given Product. |
| TPRODUCTSKUPRICE | Tproductskuprice | This table contains a set of SkuPriceTiers for each configured currency |
| TSKUPRICETIER | Tskupricetier | This table contains a set of Sku Prices Tiers. |
| TPRODUCTTYPEATTRIBUTE | Tproducttypeattribute | This table represents a mapping of Product Attributes to a ProductType. |
| TPRODUCTTYPESKUATTRIBUTE | Tproducttypeskuattribute | This table represents a mapping of ProductSKU Attributes to a ProductType. |
| TRULESET | Truleset | This table represents a set of promotion rules. |

| Table Name | OM Class | Description |
|---|---|---|
| TRULE | Trule | This table represents a Rule that can be applied by the Rules Engine. |
| TRULEELEMENT | Truleelement | This table represents the component of a Rule. For example a condition or an action. |
| TRULEEXCEPTION | Truleexception | This table represents an exception of either a Rule action or condition. |
| TRULEPARAMETER | Truleparameter | This table represents a parameter of a rule condition, such as the category that a product must belong to to qualify for a promotion. |
| TSHOPPINGCART | Tshoppingcart | This table represents the state of a customers shopping cart. |
| TCARTITEM | Tcartitem | This table represents a quantity of SKUs in a shopping cart, saved cart, wish list, etc. |
| TUSERROLEPERMISSIONX | Tuserrolepermissionx | This table represents the level of permission assigned to a given UserRole. |
| TSKUOPTION | Tskuoption | This table represents a SKU option that can be configured. For example size or color. |

| Table Name | OM Class | Description |
|---|---|---|
| TSKUOPTIONVALUE | Tskuoptionvalue | This table represents an available option value for a SKU option. Example option values include red, green, small, large, etc. |
| TPRODUCTTYPESKUOPTION | Tproducttypeskuoption | This table represents the mapping of SKU options to a ProductType. |
| TPRODUCTSKUOPTIONVALUE | Tproductskuoptionvalue | This table represents the mapping of a ProductSKU to a SKUOptionValue. |
| TSHIPPINGREGION | Tshippingregion | This table represents a region that will be associated with one or more shipping services. |
| TSHIPPINGCOSTCALCULATIONMETHOD | Tshippingcostcalculationmethod | This table represents a method to be used for shipping cost calculation; for example Fixed Price. |
| TSHIPPINGCOSTCALCULATIONPARAM | Tshippingcostcalculationparam | This table represents a parameter of a shipping cost calculation method, such as the dollar value of the fix base shipping cost. |
| TSHIPPINGSERVICELEVEL | Tshippingservicelevel | This table represents a ShippingOption, for example Next Day assoicated with a ShippingRegion. |
| TPRODUCTDELETED | Tproductdeleted | This table represents an audit |

| Table Name | OM Class | Description |
|---|---|---|
| | | of deleted products. |
| TLOCALIZEDPROPERTIES | Tlocalizedproperties | This table represents a list of localized properties. For example Brand names. |
| TDIGITALASSETAUDIT | Tdigitalassetaudit | This table represents the audit of a DigitalAsset download attempt. |
| TAPPLIEDRULE | Tappliedrule | This table represents a rule that has been applied to an order. |
| TTOPSELLER | Ttopseller | This table represents a category of top selling products. |
| TTOPSELLERPRODUCTS | Ttopsellerproducts | This table represents a rank of top selling products in the store. |
| TSFSEARCHLOG | Tsfsearchlog | This table represents a log of searches performed on the storefront. |

## HIBERNATE_UNIQUE_KEYS

Allows Hibernate to generate unqiue primary keys for other DB tables

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| VALUE | BIGINT | | | Value | | | | |

## TDIGITALASSETS

This table represents a reference to digital good files stored on the file system.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| FILE_NAME | VARCHAR | (255) | | FileName | | | X | |
| EXPIRY_DAYS | INTEGER | | | ExpiryDays | | | | |
| MAX_DOWNLOAD_TIMES | INTEGER | | | MaxDownloadTimes | | | | |

## TTAXCATEGORY

This table represents a category of tax, i.e. GST (Canada).

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| GUID | VARCHAR | (64) | | Guid | | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |

## TTAXCODE

This table represents a TaxCode. For example goods and books.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| GUID | VARCHAR | (64) | | Guid | | | X | |
| CODE | VARCHAR | (255) | | Code | | | X | |

## TTAXJURISDICTION

This table represents a TaxJurisdiction, a geographical area i.e. Canada.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X |
| GUID | VARCHAR | (64) | | Guid | | | X |
| REGION_CODE | VARCHAR | (255) | | RegionCode | | | X |
| PRICE_CALCULATION_METH | INTEGER | | 0 | PriceCalculationMeth | | | |
| FIELD_MATCH_TYPE | INTEGER | | | FieldMatchType | | | |
| PARENT_TAX_JURISDICTION_UID | BIGINT | | | ParentTaxJurisdictionUid | | X | |

## TTAXJURISDICTIONCATEGORYX

This table represents the TaxJurisdiction and TaxCategory association, include the taxValues for each configured TaxCode.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| TAX_JURISDICTION_UID | BIGINT | | | TaxJurisdictionUid | | X | X | |
| TAX_CATEGORY_UID | BIGINT | | | TaxCategoryUid | | X | X | |

## TTAXVALUE

This table represents a TaxValue for a given TaxJurisdiction and TaxCategory.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| TAX_JUR_CAT_X_UID | BIGINT | | | TaxJurCatXUid | X | X | X | |
| TAX_CODE | VARCHAR | (255) | | TaxCode | X | X | X | |
| VALUE | DECIMAL | (19,4) | | Value | | | | |

## TATTRIBUTE

This table reprents a customized property of an object such as Category, Product or Product Sku.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| ATTRIBUTE_KEY | VARCHAR | (255) | | AttributeKey | | | X | |
| LOCALE_DEPENDANT | INTEGER | | 0 | LocaleDependant | | | X | |
| ATTRIBUTE_TYPE | INTEGER | | | AttributeType | | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| SEARCH_INDEX | INTEGER | | 0 | SearchIndex | | | | |
| REQUIRED | INTEGER | | 0 | Required | | | | |
| VALUE_LOOKUP_ENABLED | INTEGER | | 0 | ValueLookupEnabled | | | | |
| MULTI_VALUE_ENABLED | INTEGER | | 0 | MultiValueEnabled | | | | |
| GUIDED_NAVIGATION | INTEGER | | 0 | GuidedNavigation | | | | |
| ATTRIBUTE_USAGE | INTEGER | | | AttributeUsage | | | X | |
| SYSTEM | INTEGER | | 0 | System | | | | |

## TCUSTOMER

This table respresents a Customer account.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| USER_ID | VARCHAR | (255) | | UserId | | | X | |
| PREF_BILL_ADDRESS_UID | BIGINT | | | PrefBillAddressUid | | | | |
| PREF_SHIP_ADDRESS_UID | BIGINT | | | PrefShipAddressUid | | | | |
| CREATION_DATE | DATE | | | CreationDate | | | | |
| LAST_EDIT_DATE | TIMESTAMP | | | LastEditDate | | | | |
| GUID | VARCHAR | (64) | | Guid | | | X | |
| STATUS | INTEGER | | | Status | | | X | |
| AUTHENTICATION_UID | BIGINT | | | AuthenticationUid | | | | |

## TCUSTOMERAUTHENTICATION

This table respresents a Customer authentication.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| PASSWORD | VARCHAR | (255) | | Password | | | | |

## TCUSTOMERPROFILEVALUE

This table represents the value of a customer profile.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| ATTRIBUTE_UID | BIGINT | | | AttributeUid | | X | X |
| ATTRIBUTE_TYPE | INTEGER | | | AttributeType | | | X |
| LOCALIZED_ATTRIBUTE_KEY | VARCHAR | (255) | | LocalizedAttributeKey | | | X |
| SHORT_TEXT_VALUE | VARCHAR | (255) | | ShortTextValue | | | |
| LONG_TEXT_VALUE | LONGVARCHAR | (65535) | | LongTextValue | | | |
| INTEGER_VALUE | INTEGER | | | IntegerValue | | | |
| DECIMAL_VALUE | DECIMAL | (19,2) | | DecimalValue | | | |
| BOOLEAN_VALUE | INTEGER | | 0 | BooleanValue | | | |
| DATE_VALUE | DATE | | | DateValue | | | |
| CUSTOMER_UID | BIGINT | | | CustomerUid | | X | |

## TCUSTOMERDELETED

This table represents the audit of a deleted Customer.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CUSTOMER_UID | BIGINT | | | CustomerUid | | | X | |
| DELETED_DATE | DATE | | | DeletedDate | | | X | |

## TADDRESS

This table represents a Customer Address.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| LAST_NAME | VARCHAR | (100) | | LastName | | | | |
| FIRST_NAME | VARCHAR | (100) | | FirstName | | | | |
| PHONE_NUMBER | VARCHAR | (50) | | PhoneNumber | | | | |
| FAX_NUMBER | VARCHAR | (50) | | FaxNumber | | | | |
| STREET_1 | VARCHAR | (200) | | Street1 | | | | |
| STREET_2 | VARCHAR | (200) | | Street2 | | | | |
| CITY | VARCHAR | (200) | | City | | | | |
| SUB_COUNTRY | VARCHAR | (200) | | SubCountry | | | | |
| ZIP_POSTAL_CODE | VARCHAR | (50) | | ZipPostalCode | | | | |
| COUNTRY | VARCHAR | (200) | | Country | | | | |
| COMMERCIAL | INTEGER | | 0 | Commercial | | | | |
| GUID | VARCHAR | (64) | | Guid | | | X | |
| CUSTOMER_UID | BIGINT | | | CustomerUid | | X | | |

## TCATEGORYTYPE

This table represents the type of a Category, which determines the set of attributes that it has. An example of a category type would be Electronics.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| TEMPLATE | VARCHAR | (255) | | Template | | | X | |
| GUID | VARCHAR | (64) | | Guid | | | X | |

## TCATEGORY

This table represents a collection of related Products.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| START_DATE | DATE | | | StartDate | | | X | |
| END_DATE | DATE | | | EndDate | | | | |
| LAST_MODIFIED_DATE | TIMESTAMP | | | LastModifiedDate | | | X | |
| ORDERING | INTEGER | | | Ordering | | | | |
| HIDDEN | INTEGER | | 0 | Hidden | | | | |
| CATEGORY_TYPE_UID | BIGINT | | | CategoryTypeUid | | X | X | |
| PARENT_CATEGORY_UID | BIGINT | | | ParentCategoryUid | | X | | |
| CODE | VARCHAR | (64) | | Code | | | X | |

## TCATEGORYDELETED

This table represents a deleted category.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CATEGORY_UID | BIGINT | | | CategoryUid | | | X | |
| DELETED_DATE | DATE | | | DeletedDate | | | X | |

## TCATEGORYATTRIBUTEVALUE

This table represents the Attribute Values for a given Category.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| ATTRIBUTE_UID | BIGINT | | | AttributeUid | | X | X |
| ATTRIBUTE_TYPE | INTEGER | | | AttributeType | | | X |
| LOCALIZED_ATTRIBUTE_KEY | VARCHAR | (255) | | LocalizedAttributeKey | | | X |
| SHORT_TEXT_VALUE | VARCHAR | (255) | | ShortTextValue | | | |
| LONG_TEXT_VALUE | LONGVARCHAR | (65535) | | LongTextValue | | | |
| INTEGER_VALUE | INTEGER | | | IntegerValue | | | |
| DECIMAL_VALUE | DECIMAL | (19,2) | | DecimalValue | | | |
| BOOLEAN_VALUE | INTEGER | | 0 | BooleanValue | | | |
| DATE_VALUE | DATE | | | DateValue | | | |
| CATEGORY_UID | BIGINT | | | CategoryUid | | X | |

## TCATEGORYLDF

This table contains locale-dependent information about a Category.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| CATEGORY_UID | BIGINT | | | CategoryUid | X | X | X | |
| URL | VARCHAR | (255) | | Url | | | | |
| KEY_WORDS | VARCHAR | (255) | | KeyWords | | | | |
| DESCRIPTION | VARCHAR | (255) | | Description | | | | |
| TITLE | VARCHAR | (255) | | Title | | | | |
| DISPLAY_NAME | VARCHAR | (255) | | DisplayName | | | | |
| LOCALE | VARCHAR | (20) | | Locale | X | | X | |

## TCATEGORYTYPEATTRIBUTE

This table represents a mapping of Category Attributes to a CategoryType

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| ORDERING | INTEGER | | | Ordering | | | | |
| ATTRIBUTE_UID | BIGINT | | | AttributeUid | | X | X | |
| CATEGORY_TYPE_UID | BIGINT | | | CategoryTypeUid | | X | X | |

## TCMUSER

This table represents a person with an account in the system for accessing the Commerce Manager or web services.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| USER_NAME | VARCHAR | (255) | | UserName | | | X | |
| EMAIL | VARCHAR | (255) | | Email | | | X | |
| FIRST_NAME | VARCHAR | (100) | | FirstName | | | | |
| LAST_NAME | VARCHAR | (100) | | LastName | | | | |
| PASSWORD | VARCHAR | (255) | | Password | | | X | |
| CM_ACCESS | INTEGER | | 0 | CmAccess | | | | |
| WS_ACCESS | INTEGER | | 0 | WsAccess | | | | |
| CREATION_DATE | DATE | | | CreationDate | | | | |
| LAST_LOGIN_DATE | DATE | | | LastLoginDate | | | | |
| GUID | VARCHAR | (64) | | Guid | | | X | |

## TUSERROLE

This table represents a user''s role.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| DESCRIPTION | VARCHAR | (255) | | Description | | | | |
| GUID | VARCHAR | (64) | | Guid | | | X | |

## TCMUSERROLEX

This table represents the UserRoles assigned to a Customer

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| CM_USER_UID | BIGINT | | | CmUserUid | X | X | X | |
| USER_ROLE_UID | BIGINT | | | UserRoleUid | X | X | X | |

## TCUSTOMERGROUP

This table represents a customer group.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| GUID | VARCHAR | (64) | | Guid | | | X | |

## TCUSTOMERGROUPROLEX

This table represents the Role of a CustomerGroup.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| CUSTOMER_GROUP_UID | BIGINT | | | CustomerGroupUid | | X | X | |
| CUSTOMER_ROLE | VARCHAR | (255) | | CustomerRole | | | | |

## TCUSTOMERGROUPX

This table represents the Customers assigned to a CustomerGroup.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| CUSTOMER_UID | BIGINT | | | CustomerUid | | X | X | |
| CUSTOMERGROUP_UID | BIGINT | | | CustomergroupUid | | X | X | |

## TCUSTOMERSESSION

This table represents information about customers who may not be logged in.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CREATION_DATE | DATE | | | CreationDate | | | X | |
| LAST_ACCESSED_DATE | TIMESTAMP | | | LastAccessedDate | | | X | |
| CUSTOMER_UID | BIGINT | | | CustomerUid | | X | | |
| EMAIL | VARCHAR | (255) | | Email | | | | |
| LOCALE | VARCHAR | (255) | | Locale | | | | |
| CURRENCY | VARCHAR | (3) | | Currency | | | | |
| GUID | VARCHAR | (64) | | Guid | | | X | |
| IP_ADDRESS | VARCHAR | (255) | | IpAddress | | | | |

## TCUSTOMERCREDITCARD

This table represents a credit card stored by a Storefront customer.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CARD_TYPE | VARCHAR | (50) | | CardType | | | X | |
| CARD_HOLDER_NAME | VARCHAR | (100) | | CardHolderName | | | X | |
| CARD_NUMBER | VARCHAR | (255) | | CardNumber | | | X | |
| EXPIRY_YEAR | VARCHAR | (4) | | ExpiryYear | | | X | |
| EXPIRY_MONTH | VARCHAR | (2) | | ExpiryMonth | | | X | |
| START_YEAR | VARCHAR | (4) | | StartYear | | | | |
| START_MONTH | VARCHAR | (2) | | StartMonth | | | | |
| ISSUE_NUMBER | INTEGER | | | IssueNumber | | | | |
| DEFAULT_CARD | INTEGER | | | DefaultCard | | | | |
| CUSTOMER_UID | BIGINT | | | CustomerUid | | X | X | |
| BILLING_ADDRESS_UID | BIGINT | | | BillingAddressUid | | X | | |

## TIMPORTJOB

This table represents an import job.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| CSV_FILE_NAME | VARCHAR | (255) | | CsvFileName | | | X | |
| COL_DELIMETER | CHAR | (1) | | ColDelimeter | | | | |
| TEXT_QUALIFIER | CHAR | (1) | | TextQualifier | | | | |
| DATA_TYPE_NAME | VARCHAR | (255) | | DataTypeName | | | X | |
| IMPORT_TYPE | INTEGER | | | ImportType | | | X | |
| MAX_ALLOW_ERRORS | INTEGER | | | MaxAllowErrors | | | X | |
| GUID | VARCHAR | (64) | | Guid | | | X | |

## TIMPORTMAPPINGS

This table represents the column to field mapping for an import job.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| IMPORT_JOB_UID | BIGINT | | | ImportJobUid | X | X | X | |
| COL_NUMBER | INTEGER | | | ColNumber | | | X | |
| IMPORT_FIELD_NAME | VARCHAR | (255) | | ImportFieldName | X | | X | |

## TORDERADDRESS

This table represents a copy of a customer Address created with an Order.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| LAST_NAME | VARCHAR | (100) | | LastName | | | | |
| FIRST_NAME | VARCHAR | (100) | | FirstName | | | | |
| PHONE_NUMBER | VARCHAR | (50) | | PhoneNumber | | | | |
| FAX_NUMBER | VARCHAR | (50) | | FaxNumber | | | | |
| STREET_1 | VARCHAR | (200) | | Street1 | | | | |
| STREET_2 | VARCHAR | (200) | | Street2 | | | | |
| CITY | VARCHAR | (200) | | City | | | | |
| SUB_COUNTRY | VARCHAR | (200) | | SubCountry | | | | |
| ZIP_POSTAL_CODE | VARCHAR | (50) | | ZipPostalCode | | | | |
| COUNTRY | VARCHAR | (200) | | Country | | | | |
| COMMERCIAL | INTEGER | | 0 | Commercial | | | | |
| GUID | VARCHAR | (64) | | Guid | | | X | |

## TORDER

This table represents a customer Order.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X |
| LAST_MODIFIED_DATE | TIMESTAMP | | | LastModifiedDate | | | |
| CREATED_DATE | DATE | | | CreatedDate | | | X |
| IP_ADDRESS | VARCHAR | (255) | | IpAddress | | | |
| ORDER_BILLING_ADDRESS_UID | BIGINT | | | OrderBillingAddressUid | | X | |
| TOTAL | DECIMAL | (19,2) | | Total | | | |
| SUBTOTAL | DECIMAL | (19,2) | | Subtotal | | | |
| BEFORE_TAX_TOTAL | DECIMAL | (19,2) | | BeforeTaxTotal | | | |
| BEFORE_TAX_SUBTOTAL | DECIMAL | (19,2) | | BeforeTaxSubtotal | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SUBTOTAL_DISCOUNT | DECIMAL | (19,2) | | SubtotalDiscount | | | |
| STATUS | INTEGER | | 0 | Status | | | |
| ORDER_NUMBER | VARCHAR | (64) | | OrderNumber | | | X |
| CUSTOMER_UID | BIGINT | | | CustomerUid | | X | |
| LOCALE | VARCHAR | (5) | | Locale | | | X |
| CURRENCY | VARCHAR | (3) | | Currency | | | |
| INCLUSIVE_TAX_CALCULATION | INTEGER | | 0 | InclusiveTaxCalculation | | | |

## TORDERNUMBERGENERATOR

This table represents the next available order number.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | 1 | Uidpk | X | | X | |
| NEXT_ORDER_NUMBER | VARCHAR | (100) | | NextOrderNumber | | | X | |

## TORDERNOTE

This table represents a note made on an order by a CSR.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CREATED_DATE | DATE | | | CreatedDate | | | X | |
| CREATED_BY | BIGINT | | | CreatedBy | | X | X | |
| NOTE | VARCHAR | (255) | | Note | | | | |
| ORDER_UID | BIGINT | | | OrderUid | | X | | |

## TORDERPAYMENT

This table represents customer payments made against an Order.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CREATED_DATE | TIMESTAMP | | | CreatedDate | | | X | |
| CARD_TYPE | VARCHAR | (50) | | CardType | | | | |
| CARD_HOLDER_NAME | VARCHAR | (100) | | CardHolderName | | | | |
| CARD_NUMBER | VARCHAR | (255) | | CardNumber | | | | |
| EXPIRY_YEAR | VARCHAR | (4) | | ExpiryYear | | | | |
| EXPIRY_MONTH | VARCHAR | (2) | | ExpiryMonth | | | | |
| START_DATE | DATE | | | StartDate | | | | |
| ISSUE_NUMBER | VARCHAR | (100) | | IssueNumber | | | | |
| PAYMENT_GATEWAY | VARCHAR | (100) | | PaymentGateway | | | | |
| AMOUNT | DECIMAL | (19,2) | | Amount | | | | |
| REFERENCE_ID | VARCHAR | (50) | | ReferenceId | | | | |
| REQUEST_TOKEN | VARCHAR | (255) | | RequestToken | | | | |
| AUTHORIZATION_CODE | VARCHAR | (50) | | AuthorizationCode | | | | |
| TRANSACTION_TYPE | VARCHAR | (20) | | TransactionType | | | | |
| CURRENCY | VARCHAR | (10) | | Currency | | | | |
| EMAIL | VARCHAR | (100) | | Email | | | | |
| STATUS | INTEGER | | 0 | Status | | | | |
| ORDER_UID | BIGINT | | | OrderUid | | X | | |

## TORDERSHIPMENT

This table represents a customer"s order shipment.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X |
| STATUS | INTEGER | | 1 | Status | | | |
| LAST_MODIFIED_DATE | TIMESTAMP | | | LastModifiedDate | | | |
| CREATED_DATE | DATE | | | CreatedDate | | | X |
| SHIPMENT_DATE | DATE | | | ShipmentDate | | | |
| CARRIER | VARCHAR | (255) | | Carrier | | | |
| SERVICE_LEVEL | VARCHAR | (255) | | ServiceLevel | | | |
| TRACKING_CODE | VARCHAR | (255) | | TrackingCode | | | |
| SHIPPING_COST | DECIMAL | (19,2) | | ShippingCost | | | |
| BEFORE_TAX_SHIPPING_COST | DECIMAL | (19,2) | | BeforeTaxShippingCost | | | |
| ORDER_ADDRESS_UID | BIGINT | | | OrderAddressUid | | X | |
| ORDER_UID | BIGINT | | | OrderUid | | X | |

## TORDERRETURN

This table represents a customer"s order return.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X |
| RETURN_REASON | INTEGER | | 1 | ReturnReason | | | |
| CREATED_DATE | DATE | | | CreatedDate | | | X |
| RMA_CODE | VARCHAR | (255) | | RmaCode | | | |
| RETURN_COMMENT | VARCHAR | (2000) | | ReturnComment | | | |
| BEFORE_TAX_RETURN_TOTAL | DECIMAL | (19,2) | | BeforeTaxReturnTotal | | | |
| RETURN_TOTAL | DECIMAL | (19,2) | | ReturnTotal | | | |
| ORDER_UID | BIGINT | | | OrderUid | | X | |

## TORDERTAX

This table represents the tax paid on a customers Order.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| TAX_CATEGORY_NAME | VARCHAR | (255) | | TaxCategoryName | | | X |
| TAX_CATEGORY_DISPLAY_NAME | VARCHAR | (255) | | TaxCategoryDisplayName | | | X |
| VALUE | DECIMAL | (19,2) | | Value | | | |
| ORDER_UID | BIGINT | | | OrderUid | | X | |
| ORDER_RETURN_UID | BIGINT | | | OrderReturnUid | | X | |

## TPRODUCTTYPE

This table represents the type of a Product, which determines the set of attributes that it has. An example of a product type would be shoes.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| WITH_MULTIPLE_SKUS | INTEGER | | 0 | WithMultipleSkus | | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| TEMPLATE | VARCHAR | (255) | | Template | | | X | |
| GUID | VARCHAR | (64) | | Guid | | | X | |
| TAX_CODE_UID | BIGINT | | | TaxCodeUid | | X | X | |

## TBRAND

This table represents product manufacturer/brand information.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CODE | VARCHAR | (255) | | Code | | | X | |
| IMAGE_URL | VARCHAR | (255) | | ImageUrl | | | | |

## TPRODUCT

This table represents represents a merchandise product. A product must have at least 1 ProductSku associated in order to be sold.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| LAST_MODIFIED_DATE | TIMESTAMP | | | LastModifiedDate | | | X | |
| START_DATE | DATE | | | StartDate | | | X | |
| END_DATE | DATE | | | EndDate | | | | |
| IMAGE | VARCHAR | (255) | | Image | | | | |
| PRODUCT_TYPE_UID | BIGINT | | | ProductTypeUid | | X | X | |
| BRAND_UID | BIGINT | | | BrandUid | | X | | |
| DEFAULT_SKU_UID | BIGINT | | | DefaultSkuUid | | | | |
| DEFAULT_CATEGORY_UID | BIGINT | | | DefaultCategoryUid | | X | | |
| CODE | VARCHAR | (64) | | Code | | | X | |
| MIN_QUANTITY | INTEGER | | 1 | MinQuantity | | | X | |
| HIDDEN | INTEGER | | 0 | Hidden | | | | |
| SALES_COUNT | INTEGER | | 0 | SalesCount | | | | |
| TAX_CODE_UID | BIGINT | | | TaxCodeUid | | X | | |

## TORDERSKU

This table represents an order for a quantity of SKUs.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| LAST_MODIFIED_DATE | TIMESTAMP | | | LastModifiedDate | | | | |
| CREATED_DATE | TIMESTAMP | | | CreatedDate | | | X | |
| SKUCODE | VARCHAR | (255) | | Skucode | | | X | |
| TAXCODE | VARCHAR | (255) | | Taxcode | | | X | |
| PRODUCT_UID | BIGINT | | | ProductUid | | | | |
| PRODUCT_SKU_UID | BIGINT | | | ProductSkuUid | | | | |
| ORDER_UID | BIGINT | | | OrderUid | | X | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ORDER_SHIPMENT_UID | BIGINT | | | OrderShipmentUid | | X | | |
| QUANTITY | INTEGER | | | Quantity | | | | |
| DISPLAY_NAME | VARCHAR | (255) | | DisplayName | | | X | |
| DISPLAY_NAME_CSR | VARCHAR | (255) | | DisplayNameCsr | | | | |
| AMOUNT | DECIMAL | (19,2) | | Amount | | | | |
| TAX | DECIMAL | (19,2) | | Tax | | | | |
| LIST_PRICE | DECIMAL | (19,2) | | ListPrice | | | | |
| UNIT_PRICE | DECIMAL | (19,2) | | UnitPrice | | | | |
| DISPLAY_SKU_OPTIONS | VARCHAR | (255) | | DisplaySkuOptions | | | | |
| IMAGE | VARCHAR | (255) | | Image | | | | |
| WEIGHT | INTEGER | | 0 | Weight | | | | |
| DIGITAL_ASSET_UID | BIGINT | | | DigitalAssetUid | | X | | |

## TORDERRETURNSKU

This table represents the return of a quantity of SKUs for an order.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| ORDER_SKU_UID | BIGINT | | | OrderSkuUid | | X | | |
| ORDER_RETURN_UID | BIGINT | | | OrderReturnUid | | X | | |
| QUANTITY | INTEGER | | | Quantity | | | | |
| RETURN_AMOUNT | DECIMAL | (19,2) | | ReturnAmount | | | | |

## TPRODUCTATTRIBUTEVALUE

This table represents the value of a Product Attribute.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| ATTRIBUTE_UID | BIGINT | | | AttributeUid | | X | X |
| ATTRIBUTE_TYPE | INTEGER | | | AttributeType | | | X |
| LOCALIZED_ATTRIBUTE_KEY | VARCHAR | (255) | | LocalizedAttributeKey | | | X |
| SHORT_TEXT_VALUE | VARCHAR | (255) | | ShortTextValue | | | |
| LONG_TEXT_VALUE | LONGVARCHAR | (65535) | | LongTextValue | | | |
| INTEGER_VALUE | INTEGER | | | IntegerValue | | | |
| DECIMAL_VALUE | DECIMAL | (19,2) | | DecimalValue | | | |
| BOOLEAN_VALUE | INTEGER | | 0 | BooleanValue | | | |
| DATE_VALUE | DATE | | | DateValue | | | |
| PRODUCT_UID | BIGINT | | | ProductUid | | X | |

## TPRODUCTCATEGORY

This table represents the association between a Category and it''s contained Products.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| PRODUCT_UID | BIGINT | | | ProductUid | | X | X | |
| CATEGORY_UID | BIGINT | | | CategoryUid | | X | X | |
| FEAT_PRODUCT_ORDER | INTEGER | | 0 | FeatProductOrder | | | | |

## TPRODUCTLDF

This table contains locale-dependent information about a Product.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| PRODUCT_UID | BIGINT | | | ProductUid | X | X | X | |
| URL | VARCHAR | (255) | | Url | | | | |
| KEY_WORDS | VARCHAR | (255) | | KeyWords | | | | |
| DESCRIPTION | VARCHAR | (255) | | Description | | | | |
| TITLE | VARCHAR | (255) | | Title | | | | |
| DISPLAY_NAME | VARCHAR | (255) | | DisplayName | | | | |
| LOCALE | VARCHAR | (20) | | Locale | X | | X | |

## TPRODUCTPRICE

This table contains a set of ProductPriceTiers for each configured currency.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CURRENCY | VARCHAR | (255) | | Currency | | | | |
| PRODUCT_UID | BIGINT | | | ProductUid | | X | | |

## TPRODUCTPRICETIER

This table contains a set of Product Prices Tiers.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| LIST_PRICE | DECIMAL | (19,2) | | ListPrice | | | X | |
| SALE_PRICE | DECIMAL | (19,2) | | SalePrice | | | | |
| MIN_QUANTITY | INTEGER | | | MinQuantity | X | | X | |
| PRODUCT_PRICE_UID | BIGINT | | | ProductPriceUid | X | X | X | |

## TINVENTORY

This table represents the Inventory information for a ProductSku.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X |
| QUANTITY_ON_HAND | INTEGER | | | QuantityOnHand | | | |
| INFINITE_QUANTITY_ON_HAND | INTEGER | | 0 | InfiniteQuantityOnHand | | | |
| RESERVED_QUANTITY | INTEGER | | | ReservedQuantity | | | |
| REORDER_MINIMUM | INTEGER | | 0 | ReorderMinimum | | | |
| VISIBLE_WHEN_OUT_OF_STOCK | INTEGER | | 0 | VisibleWhenOutOfStock | | | |
| RESTOCK_DATE | DATE | | | RestockDate | | | |

## TINVENTORYAUDIT

This table contains an audit of inventory adjustment events.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| INVENTORY_UID | BIGINT | | | InventoryUid | | | X | |
| QUANTITY | INTEGER | | | Quantity | | | | |
| LOG_COMMENT | LONGVARCHAR | (65535) | | LogComment | | | | |
| CMUSER_UID | BIGINT | | | CmuserUid | | | X | |
| LOG_DATE | TIMESTAMP | | | LogDate | | | | |

## TPRODUCTSKU

This table represents a variation of a merchandise product.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| START_DATE | DATE | | | StartDate | | | X | |
| END_DATE | DATE | | | EndDate | | | | |
| SKUCODE | VARCHAR | (255) | | Skucode | | | X | |
| IMAGE | VARCHAR | (255) | | Image | | | | |
| INVENTORY_UID | BIGINT | | | InventoryUid | | X | | |
| PRODUCT_UID | BIGINT | | | ProductUid | | X | | |
| SHIPPABLE | INTEGER | | 1 | Shippable | | | | |
| WEIGHT | DECIMAL | (19,2) | 0 | Weight | | | | |
| HEIGHT | DECIMAL | (19,2) | 0 | Height | | | | |
| WIDTH | DECIMAL | (19,2) | 0 | Width | | | | |
| LENGTH | DECIMAL | (19,2) | 0 | Length | | | | |
| DIGITAL_ASSET_UID | BIGINT | | | DigitalAssetUid | | X | | |

## TPRODUCTASSOCIATION

This table represents ProductAssociations between two products. Example of product associations are Cross-Sells, Up-Sells, Accessories.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X |
| ASSOCIATION_TYPE | INTEGER | | | AssociationType | | | X |
| SOURCE_PRODUCT_UID | BIGINT | | | SourceProductUid | | X | X |
| TARGET_PRODUCT_UID | BIGINT | | | TargetProductUid | | X | X |
| TARGET_SKU_UID | BIGINT | | | TargetSkuUid | | X | |
| START_DATE | DATE | | | StartDate | | | X |
| END_DATE | DATE | | | EndDate | | | |
| DEFAULT_QUANTITY | INTEGER | | 1 | DefaultQuantity | | | X |
| SOURCE_PRODUCT_DEPENDENT | INTEGER | | 0 | SourceProductDependent | | | |
| ORDERING | INTEGER | | 0 | Ordering | | | |

## TPRODUCTSKUATTRIBUTEVALUE

This table represents the Attribute Values for a given Product.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| ATTRIBUTE_UID | BIGINT | | | AttributeUid | | X | X |
| ATTRIBUTE_TYPE | INTEGER | | | AttributeType | | | X |
| LOCALIZED_ATTRIBUTE_KEY | VARCHAR | (255) | | LocalizedAttributeKey | | | X |
| SHORT_TEXT_VALUE | VARCHAR | (255) | | ShortTextValue | | | |
| LONG_TEXT_VALUE | LONGVARCHAR | (65535) | | LongTextValue | | | |
| INTEGER_VALUE | INTEGER | | | IntegerValue | | | |
| DECIMAL_VALUE | DECIMAL | (19,2) | | DecimalValue | | | |
| BOOLEAN_VALUE | INTEGER | | 0 | BooleanValue | | | |
| DATE_VALUE | DATE | | | DateValue | | | |
| PRODUCT_SKU_UID | BIGINT | | | ProductSkuUid | | X | |

## TPRODUCTSKUPRICE

This table contains a set of SkuPriceTiers for each configured currency.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CURRENCY | VARCHAR | (255) | | Currency | | | | |
| PRODUCT_SKU_UID | BIGINT | | | ProductSkuUid | | X | | |

## TSKUPRICETIER

This table contains a set of Sku Prices Tiers.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| LIST_PRICE | DECIMAL | (19,2) | | ListPrice | | | | |
| SALE_PRICE | DECIMAL | (19,2) | | SalePrice | | | | |
| MIN_QUANTITY | INTEGER | | | MinQuantity | | | | |
| SKU_PRICE_UID | BIGINT | | | SkuPriceUid | | X | | |

## TPRODUCTTYPEATTRIBUTE

This table represents a mapping of Product Attributes to a ProductType.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| ORDERING | INTEGER | | | Ordering | | | | |
| ATTRIBUTE_UID | BIGINT | | | AttributeUid | | X | X | |
| PRODUCT_TYPE_UID | BIGINT | | | ProductTypeUid | | X | X | |

## TPRODUCTTYPESKUATTRIBUTE

This table represents a mapping of ProductSKU Attributes to a ProductType.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| ORDERING | INTEGER | | | Ordering | | | | |
| ATTRIBUTE_UID | BIGINT | | | AttributeUid | | X | X | |
| PRODUCT_TYPE_UID | BIGINT | | | ProductTypeUid | | X | X | |

## TRULESET

This table represents a set of promotion rules.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| LAST_MODIFIED_DATE | TIMESTAMP | | | LastModifiedDate | | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| SCENARIO | INTEGER | | | Scenario | | | X | |

## TRULE

This table represents a Rule that can be applied by the Rules Engine.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| START_DATE | DATE | | | StartDate | | | | |
| END_DATE | DATE | | | EndDate | | | | |
| ELIGIBILITY_OPERATOR | INTEGER | | 0 | EligibilityOperator | | | | |
| CONDITION_OPERATOR | INTEGER | | 0 | ConditionOperator | | | | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| PROMO_CODE | VARCHAR | (100) | | PromoCode | | | | |
| SINGLE_USE | INTEGER | | 0 | SingleUse | | | | |
| RULE_SET_UID | BIGINT | | | RuleSetUid | | X | | |

## TRULEELEMENT

This table represents the component of a Rule. For example a condition or an action.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| TYPE | VARCHAR | (255) | | Type | | | X | |
| KIND | VARCHAR | (255) | | Kind | | | X | |
| RULE_UID | BIGINT | | | RuleUid | | X | | |

## TRULEEXCEPTION

This table represents an exception of either a Rule action or condition.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| TYPE | VARCHAR | (255) | | Type | | | X | |
| RULE_ELEMENT_UID | BIGINT | | | RuleElementUid | | X | | |

## TRULEPARAMETER

This table represents a parameter of a rule condition, such as the category that a product must belong to to qualify for a promotion.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| PARAM_KEY | VARCHAR | (255) | | ParamKey | | | X | |
| PARAM_VALUE | VARCHAR | (255) | | ParamValue | | | X | |
| DISPLAY_TEXT | VARCHAR | (255) | | DisplayText | | | | |
| RULE_ELEMENT_UID | BIGINT | | | RuleElementUid | | X | | |
| RULE_EXCEPTION_UID | BIGINT | | | RuleExceptionUid | | X | | |

## TSHOPPINGCART

This table represents the state of a customers shopping cart.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| GUID | VARCHAR | (100) | | Guid | | | X | |

## TCARTITEM

This table represents a quantity of SKUs in a shopping cart, saved cart, wish list, etc.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| TYPE | VARCHAR | (255) | | Type | | | X | |
| SKU_UID | BIGINT | | | SkuUid | | X | | |
| QUANTITY | INTEGER | | | Quantity | | | | |
| SHOPPING_CART_UID | BIGINT | | | ShoppingCartUid | | X | | |
| PARENT_CART_ITEM_UID | BIGINT | | | ParentCartItemUid | | | | |

## TUSERROLEPERMISSIONX

This table represents the level of permission assigned to a given UserRole.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| ROLE_UID | BIGINT | | | RoleUid | | X | X | |
| USER_PERMISSION | VARCHAR | (255) | | UserPermission | | | | |

## TSKUOPTION

This table represents a SKU option that can be configured. For example size or color.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| OPTION_KEY | VARCHAR | (100) | | OptionKey | | | X | |

## TSKUOPTIONVALUE

This table represents an available option value for a SKU option. Example option values include red, green, small, large, etc.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| OPTION_VALUE_KEY | VARCHAR | (255) | | OptionValueKey | | | X | |
| ORDERING | INTEGER | | | Ordering | | | | |
| SKU_OPTION_UID | BIGINT | | | SkuOptionUid | | X | X | |
| IMAGE | VARCHAR | (255) | | Image | | | | |

## TPRODUCTTYPESKUOPTION

This table represents the mapping of SKU options to a ProductType.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| PRODUCT_TYPE_UID | BIGINT | | | ProductTypeUid | | X | X | |
| SKU_OPTION_UID | BIGINT | | | SkuOptionUid | | X | X | |
| ORDERING | BIGINT | | 0 | Ordering | | | | |

## TPRODUCTSKUOPTIONVALUE

This table represents the mapping of a ProductSKU to a SKUOptionValue.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| PRODUCT_SKU_UID | BIGINT | | | ProductSkuUid | | X | X | |
| OPTION_KEY | VARCHAR | (100) | | OptionKey | | | X | |
| OPTION_VALUE_UID | BIGINT | | | OptionValueUid | | X | X | |

## TSHIPPINGREGION

This table represents a region that will be associated with one or more shipping services.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| NAME | VARCHAR | (255) | | Name | | | X | |
| REGION_STR | VARCHAR | (2000) | | RegionStr | | | | |
| GUID | VARCHAR | (64) | | Guid | | | X | |

## TSHIPPINGCOSTCALCULATIONMETHOD

This table represents a method to be used for shipping cost calculation; for example Fixed Price.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| TYPE | VARCHAR | (255) | | Type | | | X | |

## TSHIPPINGCOSTCALCULATIONPARAM

This table represents a parameter of a shipping cost calculation method, such as the dollar value of the fix base shipping cost.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| PARAM_KEY | VARCHAR | (255) | | ParamKey | X | | X | |
| VALUE | VARCHAR | (255) | | Value | | | X | |
| DISPLAY_TEXT | VARCHAR | (255) | | DisplayText | | | X | |
| SCCM_UID | BIGINT | | | SccmUid | X | X | X | |

## TSHIPPINGSERVICELEVEL

This table represents a ShippingOption, for example Next Day assoicated with a ShippingRegion.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| GUID | VARCHAR | (64) | | Guid | | | X | |
| SHIPPING_REGION_UID | BIGINT | | | ShippingRegionUid | | X | X | |
| SCCM_UID | BIGINT | | | SccmUid | | X | X | |
| CARRIER | VARCHAR | (255) | | Carrier | | | | |

## TPRODUCTDELETED

This table represents an audit of deleted products.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| PRODUCT_UID | BIGINT | | | ProductUid | | | X | |
| DELETED_DATE | DATE | | | DeletedDate | | | X | |

## TLOCALIZEDPROPERTIES

This table represents a list of localized properties. For example Brand names.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| OBJECT_UID | BIGINT | | | ObjectUid | X | | X | |
| LOCALIZED_PROPERTY_KEY | VARCHAR | (255) | | LocalizedPropertyKey | X | | X | |
| VALUE | VARCHAR | (255) | | Value | | | X | |

## TDIGITALASSETAUDIT

This table represents the audit of a DigitalAsset download attempt.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| ORDERSKU_UID | BIGINT | | | OrderskuUid | | | X | |
| DIGITALASSET_UID | BIGINT | | | DigitalassetUid | | | X | |
| DOWNLOAD_TIME | DATE | | | DownloadTime | | | X | |
| IP_ADDRESS | VARCHAR | (255) | | IpAddress | | | | |

## TAPPLIEDRULE

This table represents a rule that has been applied to an order.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| ORDER_UID | BIGINT | | | OrderUid | | | X | |
| RULE_UID | BIGINT | | | RuleUid | | | X | |
| RULE_NAME | VARCHAR | (255) | | RuleName | | | X | |
| RULE_CODE | LONGVARCHAR | (65535) | | RuleCode | | | X | |

## TTOPSELLER

This table represents a category of top selling products.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X | |
| CATEGORY_UID | BIGINT | | | CategoryUid | | | X | |

## TTOPSELLERPRODUCTS

This table represents a rank of top selling products in the store.

| Name | Type | Size | Default | JavaName | PK | FK | not null | Description |
|---|---|---|---|---|---|---|---|---|
| TOP_SELLER_UID | BIGINT | | | TopSellerUid | | X | X | |
| PRODUCT_UID | BIGINT | | | ProductUid | | | X | |
| SALES_COUNT | INTEGER | | | SalesCount | | | X | |

## TSFSEARCHLOG

This table represents a log of searches performed on the storefront.

| Name | Type | Size | Default | JavaName | PK | FK | not null |
|---|---|---|---|---|---|---|---|
| UIDPK | BIGINT | | | Uidpk | X | | X |
| SEARCH_TIME | TIMESTAMP | | | SearchTime | | | |
| KEYWORDS | VARCHAR | (255) | | Keywords | | | |
| RESULT_COUNT | INTEGER | | | ResultCount | | | X |
| SUGGESTIONS_GENERATED | INTEGER | | 0 | SuggestionsGenerated | | | |
| CATEGORY_RESTRICTION | BIGINT | | 0 | CategoryRestriction | | | X |

# *Miscellaneous*

## Generated Keys

AquaLogic Commerce Services uses the Hibernate Hilo algorithm to generate primary keys. When generating keys, hibernate grabs the hi portion from the HIBERNATE_UNIQUE_KEYS table and creates an incrementing lo portion; eg., 13 (hi) + 0001 (lo) = 13001, 13 + 0002 = 13002 and so on until the hi value needs to be updated to 14 in the db.

## Database Compatibility Issues

### Oracle

- Query Length Limitatin
  Since Oracle only allow 1000 expressions in one query, we have to separate big query over 1000 expressions like "select ... from ... where ... in (..., ...)" to multiple queries. The following code from ProductServiceImpl.findByUids() is the recommended way to do it.

```
    final List queries =
    this.getUtility().composeQueries(Criteria.PRODUCT_BY_UIDS,
    Criteria.PLACE_HOLDER_FOR_LIST, productUids);
       final List result = getPersistenceEngine().retrieve(queries);
```

- Ampersand(&) cannot be used in sql scripts.

```
       INSERT INTO TBRAND (UIDPK,CODE,GUID)  VALUES (1110,'D&H',
       'D&H');
```

must be changed to :

```
       INSERT INTO TBRAND (UIDPK,CODE,GUID)  VALUES
       (1110,concat('D', concat(CHR(38), 'H')), concat('D',
       concat(CHR(38), 'H')));
```

- Literal date value like '2006-11-11 11:11:11' cannot be used in sql scripts.
  It must be changed to :

```
to_date('2005-06-10','YYYY-MM-DD')
```

  That means if you have literal date value in a torque file, the generated sql script for Oracle won't work.
  You'll have to manually change it.

# API Reference

The complete API reference is available here: http://edocs.bea.com/alcs/docs51/javadoc.

# AquaLogic Commerce Services Connect – Web Services Interface

AquaLogic Commerce Services Connect is a module included in AquaLogic Commerce Services which allows you to integrate your AquaLogic Commerce Services store with external systems.

AquaLogic Commerce Services Connect integrates Apache Axis (an open source Web Services engine for Java) along with Dozer and XDoclet to expose services from the AquaLogic Commerce Services service layer as Web Services to external systems. Once you define which services to include in the Web Services API (by editing configuration files) AquaLogic Commerce Services Connect will build your Web Services API for you.

## *What are Web Services?*

Web Services is a term used to define a set of technologies that exposes business functionality over the Web as a set of declarative interfaces. These interfaces allow applications to discover and integrate with other applications over the web using standard Internet protocols (i.e. HTTP) and data format (i.e. XML).

A web service:

- Is a programmable application, accessible as a component via standard Web protocols.
- Uses standard Web protocols like HTTP, XML and SOAP.
- Works through existing proxies and firewalls.
- Can take advantage of HTTP authentication.
- Supports encryption for free with SSL.
- Supports easy incorporation with existing XML messaging solutions.
- Takes advantage of XML messaging schemas and easy transition from XML RPC solutions.
- Combines the best aspects of component-based development and the Web.
- Is available to a variety of clients (platform independent).

Web Services is being considered the next "big thing" in software development. It represents an important evolutionary step in building distributed applications. Typical application areas are business-to-business integration, business process integration and management, content management, and design collaboration. Most business will eventually become both suppliers and consumers of Web Services. Web Services take what HTML and TCP/IP started, and add the element of XML to enable task-focused services that come together dynamically over the Web.

## *Web Services Security*

AquaLogic Commerce Services Connect currently uses SSL (HTTPS) in conjunction with HTTP Basic Authorization to provide security for web services. Acegi is used (as in the SF and CM) to provide authentication. If your EP Connect application will be exposed to the internet, please ensure that HTTPS is enabled since Basic information essentially sent unencrypted over the network.

Currently, EP Connect users must be configured as a Web Services User (to allow WS access) through the CM user admin pages. Out of the box, users with the super user role can access all web services (ie; product, customer, inventory, order). WS-level permissions can be configured by creating new user role with only access (for example) to the product web service. Users can then be configured with this new product web service role. For finer-grained (method-level control), please enable the methodSecurityInterceptor in serviceWS.xml and acegi.xml to determine permissions necessary for a specific WS method. Please note that method-level control should only be used sparingly and the default web service level access control should be used whenever possible.

## *AquaLogic Commerce Services Web Services infrastructure*

The AquaLogic Commerce Services web services layer provides programmatic access to the application using a simple, powerful, and secure Application Programming Interface (API). Using a Web Service-enabled development environment, one can construct Web Service client applications that use standard Web Service protocols to access all the operation available in the AquaLogic Commerce Services application's service layer. For example,

- query the application's information
- create, update, and delete the application's data

### Web Services Standards

The relevant standards for describing and using Web Services are the Web Services Description Language (WSDL), which is a standardized XML schema-like language that is used for specifying a Web Service and the Simple Object Access Protocol (SOAP), which is the actual communication protocol for Web Services.

### SOAP binding in WSDL

AquaLogic Commerce Services Web Services uses a document style binding format with literal representation. The Document/Literal style rules are less rigid and many enhancements and changes can be made to the XML schema without breaking the interface. Also since everything

within the soap body is defined in the schema, the message can be validate with any XML validator. However, since Document/Literal Web Services style relies on the natural tree structures to represent data objects, it lacks support for complicated data models like cyclic data graphs and polymorphism.

## SOAP Engine

AquaLogic Commerce Services Connect uses Apache Axis as the SOAP engine. Axis is an implementation of the SOAP ("Simple Object Access Protocol") submission to W3C. It is a well-architected SOAP engine that provides the following features.

- Speed - Axis uses the Simple API for XML (SAX) for greater processing speed than SOAP implementations that use Document Object Model (DOM) parsing. DOM parsing uses a tree-based navigation model that requires creation of an internal tree based on the XML document structure. SAX parsing uses an event-based navigation model that requires no internal representation of the XML document. A SAX parser scans XML data and calls handler functions when certain parts of the document are found.
- Extensibility - Extend Axis to provide additional features such as custom header processing and system management.
- Component-oriented architecture - Define reusable message handler classes, processors that can be reused and combined in handler chains. Chains are reusable networks of message handlers you can use to implement common patterns of processing.
- Transport framework - Use a simple abstraction to design transports, senders and listeners for SOAP over various protocols such as SMTP, FTP, and message-oriented middleware. The engine core is transport-independent.
- Declarative configuration - Configure Axis for specific web services or custom functionality by editing a web service deployment descriptor (WSDD) file, server-config.wsdd, in a web application's WEB-INF directory.

## *AquaLogic Commerce Services Web Services Domain Models*

Since Document/Literal web services style lacks support for complicated data model, the web services layer uses a simplified version of the AquaLogic Commerce Services rich domain models. This approach would take advantage of the supported XML primitive data type. For each domain model within AquaLogic Commerce Services application, there will a corresponding simplified web service domain model with the following relationships:

- keep all the primitive types,
- convert all the data of type java.util.List to a detailed Object array, i.e. AddressWsImpl[],

The basic mappings between Java types and XML schema primitive data types in Axis are listed below.

```
Standard mappings from WSDL to Java
xsd:base64Binary byte\[\]
xsd:boolean boolean
```

```
xsd:byte byte
xsd:dateTime java.util.Calendar
xsd:decimal java.math.BigDecimal
xsd:double double
xsd:float float
xsd:hexBinary byte\[\]
xsd:int int
xsd:integer java.math.BigInteger
xsd:long long
xsd:QName javax.xml.namespace.QName
xsd:short short
xsd:string java.lang.String
```

Note that the XML Schema does not define a 'char', java.util.Locale, or java.util.Currency type. Currently web service domain objects created by XDoclet will convert char types to String and Dozer will map primitive types to Strings on the fly. Locale and Currency types are converted to and from Strings with a custom Dozer bean converter; see dozerBeanMappings.xml for more details.

## *Data conversion with Dozer*

Since the web services use simplified domain models, conversion is required between the simplified domain models and the AquaLogic Commerce Services rich domain models, and vice-versa. AquaLogic Commerce Services web services uses the Dozer mapper to copy data from one object to another. Dozer is a simple bean to bean mapper. Dozer supports mapping between attribute names as well as converting between types. Many conversion types are supported out of the box, but Dozer also allows you to specify custom conversions via XML (e.g. dozerBeanMapping.xml). Mapping is bi-directional so only one relationship between objects needs to be defined in the XML. If any property names on both objects are the same Dozer maps this automatically without needing to explicity define the mapping in the xml file.

For more details, refer to http://dozer.sourceforge.net.

Here is an example Dozer mapping.

```
<mapping map-null="false">
 <class-a>com.elasticpath.ws.domain.impl.CustomerWsImpl</class-a>
 <class-b>com.elasticpath.domain.impl.CustomerImpl</class-b>

 <field>
  <a>addresses</a>
  <b>addresses</b>
  <a-hint>com.elasticpath.ws.domain.impl.AddressWsImpl</a-hint>
  <b-hint>com.elasticpath.domain.impl.CustomerAddressImpl</b-hint>
```

```
 </field>

 <field>
  <a>customerGroups</a>
  <b>customerGroups</b>
  <a-hint>com.elasticpath.ws.domain.impl.CustomerGroupWsImpl</a-hint>
  <b-hint>com.elasticpath.domain.impl.CustomerGroupImpl</b-hint>
 </field>

 <field-exclude type="one-way">
  <a>uidPk</a>
  <b>uidPk</b>
 </field-exclude>

</mapping>
```

## *Using Web Services*

AquaLogic Commerce Services Connect uses Apache Axis 1.4 to provide web services. For more information on Axis please refer to http://ws.apache.org/axis/. This page will give a brief description of our Axis implementation and our exposed web services.

### Where are the WSDLs?

The Axis page showing all the web services with links to their WSDL's can be accessed at the following url: http://localhost:7001/connect/services for AquaLogic Commerce Services Connect deployed locally on port 7001 with a root context of connect. A HTTP Basic user/pass prompt should appear for the 'Connect' realm. Enter the username and password of a Web Services User with the SUPERUSER user role (ie; the admin user by default). If the correct username and password are entered a page displaying the various web services and links to WSDL's will appear. The AdminService and Version web services are enabled by default by Axis. Click on the WSDL link(s) to download and view the individual WSDLs. With the downloaded WSDL's one can construct web service client code to call AquaLogic Commerce Services Connect web services.

### Web Services Permissions

By default only SUPERUSER Web Services Users can access all the web services. In order to create finer-grained control additional user roles and users must be created through the CM. Individual permissions for each web service already exist, so for example, to create a user capable of only using the Product Web Service:

1. Log into the CM.
2. Click the Admin tab.
3. Click User Roles on the left (under User Config).

4. Click Add User Role button.
5. Enter a role name, description and ensure than the Product WS permission check box is checked. The new user role should appear in the list.
6. Click Users on the left.
7. Click Add User button. Ensure that all the required information is filled in on the User tab and that Web Services User is checked.
8. Click the Roles tab. Select the new Product WS user role and move to the right.
9. Click save.

This same process can be used to create users that can access more than one web service. For more detail on user config please consult the Commerce Manager User Manual.

## Client Testing

Download the WSDL for the web service you'd like to test and use your favorite language/web services toolkit to create a web services client; ie, java's Axis framework can generate web services client code and skeleton junit test cases based on WSDLs.

# *Services and Methods*

This section will list and briefly describe the various web services and their exposed methods.

## ProductServiceWS

This is a web service wrapper class for ProductServiceImpl.

The following method(s) are exposed:

- findByBrandUid - returns an array of products based on the uidPk of the product's brand.

## OrderServiceWS

This is a web service wrapper class for OrderServiceImpl.

The following method(s) are exposed:

- findOrderByCriteria - returns an array of orders that match the order criteria parameter. Results are limited by maxReturnNumber in commerce-config.xml.

## InventoryServiceWS

This is a web service wrapper class for InventoryServiceImpl.

The following method(s) are exposed:

- updateBySkuCode - updates database inventory row with supplied inventory parameter. The skuCode parameter is used to find row to update, uidPk of inventory parameter is ignored.
- update - updates database inventory row with supplied inventory parameter. The uidPk of the inventory parameter is used to find row to update.

### CustomerServiceWS

This is a web service wrapper class for CustomerServiceImpl.

The following method(s) are exposed

- findCustomerByCriteria - returns an array of customers that match the customer criteria parameter. Results are limited by maxReturnNumber in commerce-config.xml.
- add - adds a customer into the database. Password will be encrypted and a customer object with updated uidPk will be returned.

# Exception Handling

This document reviews exception handling practices in Java as well as how they are handled in the Web tier, AJAX, and AquaLogic Commerce Services Connect web services.

## *Exception handing practices*

The following exception handling tips and practices should be observed.

- Never throw java.lang.RuntimeException or java.lang.Exception. Use a more specific exception that indicates the nature of the problem.
- Never catch an exception just to log it and then ignore it or rethrow the same exception to a higher application layer. In the AquaLogic Commerce Services applications, all exceptions will be caught and logged in the web layer.
- You may catch a checked exception, wrap it with a general Elastic Pate exception, e.g. EpServiceException, and throw it to a higher layer.
- If you create a new exception in a method, remember to put it in the method declaration header. This is especially important for runtime exceptions.
- Abstraction - "Throw exceptions appropriate to the abstraction. In other words, exceptions thrown by a method should be defined at an abstraction level consistent with what the method does, not necessarily with the low-level details of how it is implemented." (Javaworld 2004)
- Fail early; Fail loudly - If you have an error let it be thrown and don't hide it. It is better to throw it early than find out later.
- Be sure to throw the proper type of exception, e.g. do not throw an EpDomainException in service layer code.
- Make the exception message well versed since these messages may be exposed to end-user in the Commerce Manager or through a web service.

The following code snippet shows examples of preferred and discouraged error handling practices.

```
//The error handling approach in this method is discouraged.
//Objects of the wrong type are ignored, potentially hiding
//a serious error.
public int compare(Object o1, Object o2) {
    int result = 0;
    String time1 = "";
    String time2 = "";
    if (o1 instanceof String && o2 instanceof String) {
        final String time1 = (String)o1;
        final String time2 = (String)o2;
        result =
Timestamp.valueOf(time1).compareTo(Timestamp.valueOf(time2));
    }
    return result;
}

//This method is preferred. In this example, the code is less complex
//and passing an object of the wrong type will throw an exception,
//potentially uncovering a serious bug.
public int compare(Object o1, Object o2) {
    final String time1 = (String)o1;
    final String time2 = (String)o2;
    return
Timestamp.valueOf(time1).compareTo(Timestamp.valueOf(time2));
}
```

## Generic AquaLogic Commerce Services exceptions by layer

When a new exception type is not warranted, throw an existing exception corresponding to the application layer. The following exception types are available.

- EpPersistenceException
- EpServiceException
- EpDomainException
- EpSfWebException

## Uncaught exceptions in the Web layer

A list of ordered HandlerExceptionResolver objects can be configured in the web layer with Spring MVC. Make sure the most generic one, namely EpSystemExceptionHandler.java, has the

lowest order number. This handler will catch all uncaught exceptions in the web layer and redirect the user to the general error page where the exception is displayed.

### AJAX Exceptions

When using DWR to remotely invoke service layer methods, any exceptions will be passed back to the client. The client-side application can then choose to display an appropriate message to the user.

### Web Service Exception Handling

For Web Service Exception Handling, AquaLogic Commerce Services uses the SOAPFaultException technique. This means that when any exception happens at server side, the AquaLogic Commerce Services web service will compose a SOAPFaultException to wrap the exception.

This technique is recommened in the following article.

- Exception Handling in Web Services
  http://www.developer.com/net/csharp/article.php/10918_3088231_1

Also, this is the second most preferred technique mentioned in the following article.

- Web services programming tips and tricks: Exception Handling with JAX-RPC
  http://www-128.ibm.com/developerworks/xml/library/ws-tip-jaxrpc.html

The preferred technique according to this article, checked exceptions, seems only useful when you want to send a lot of infomation back to the client in your exception.

## Search Engine and Indexing

This section explains how to interface with the Lucene search engine which underlies all search capabilities in AquaLogic Commerce Services. Lucene search is used to search for products in the Storefront. In the Commerce Manager, Lucene is used to search for products as well as other kinds of entites such as categories, customers and orders.

### How to use Lucene search

1. Create and populate a SearchCriteria bean. There are 4 types of search criteria: ProductSearchCriteria, CategorySearchCriteria, CustomerSearchCriteria, OrderSearchCriteria
2. Inject an indexSearchService and call the following method of the service.

```
/**
 * Searches the index with the given search criteria.
 *
 * @param searchCriteria the search criteria
 * @return a list of object uids which match the given search
criteria
```

```
     */
       List search(SearchCriteria searchCriteria);
```

3. Load objects based on your needs

The search will return the UIDs of the objects that can be used to load them from Hibernate. However, you many only need to load the top 100 products of the returned 5000 product UID list.

You can use the following method in ProductService to load products by their UIDs.

```
/**
 * Returns a list of <code>Product</code> based on the given uids.
 * The returned products will be populated based on the given load
tuner. If a
 * given product Uid is not found, it won't be included in the return
list.
 *
 * @param productUids a collection of product uids
 * @param loadTuner the load tuner
 * @return a list of <code>Product</code>s
 */
List findByUids(Collection productUids, ProductLoadTuner loadTuner);
```

## *Search fields and index fields*

| Object | Search Fields | Index Fields | Locale | Comment |
|---|---|---|---|---|
| product(SF) | keyword | product display name; brand display name; sku code; attribute value which are defined indexable | all locales | If you give multiple keywords, like: keyword1 keyword2. The search result will products whose index fields contain BOTH keyword1 and keyword2. |
| product(SF) | active only | start date & end date | n/a | product is active if : start date <= now <= end date |
| category(CM) | keyword | category display name, code | system default locale | |
| category(CM) | active only | start date & end date | n/a | category is active if : start date <= now <= end date |
| category(CM) | inactive only | start date & end date | n/a | category is inactive if : now <= start date or now >= end date |

| | | | | |
|---|---|---|---|---|
| product(CM) | keyword | product display name, code, sku code | system default locale | |
| product(CM) | active only | start date & end date | n/a | product is active if : start date <= now <= end date |
| product(CM) | inactive only | start date & end date | n/a | product is active if : now <= start date or now >= end date |
| product(CM) | brand code | brand code | n/a | |
| product(CM) | category uid | all ancestor categories uids | n/a | |
| customer(CM) | first name | customer first name | n/a | |
| customer(CM) | last name | customer last name | n/a | |
| customer(CM) | customer number | customer number(uidPk) | n/a | |
| customer(CM) | email | customer email | n/a | |
| customer(CM) | phone number | phone number | n/a | |
| order(CM) | order number | order number | n/a | |
| order(CM) | order status | order status code | n/a | |
| order(CM) | sku code | all sku codes in an order | n/a | |
| order(CM) | order from date | order create date | n/a | where start date <= create date |
| order(CM) | order to date | order create date | n/a | where create date <= to date |
| order(CM) | order shipment zipcode | all order shipments zipcode | n/a | |
| order(CM) | order shipment status | all order shipments status code | n/a | |
| order(CM) | first name | customer first name | n/a | |

| | | | | |
|---|---|---|---|---|
| order(CM) | last name | customer last name | n/a | |
| order(CM) | customer number | customer number(uidPk) | n/a | |
| order(CM) | email | customer email | n/a | |
| order(CM) | phone number | customer phone number | n/a | |

For an object search, if multiple search fields are given, it's always an AND relationship among all of them.

## Index builder

The index builder constructs the index that Lucene uses to execute searches. The index builder is invoked by a Quartz scheduled job configured in quartz.xml. The index builder uses properties from buildIndex.properties, which is described in the following section. There are several index builders that build indices for different entities and their index files are named according to the pattern xBuildIndex.properties where x is the name of the entity being indexed such as a customer or product.

## Key classes and files

The following files and Java classes are also related to the Lucene search index support.

buildIndex.properties

- Contains the following two properties.
    - Rebuild - set to true to create a new index. This provides a way to create a new index when the server is already started. The default value is false and the value is also set to false after a successful index build/update.
    - LastBuildDate - the date when the last build was executed successfully.

IndexBuildService

- This service class handles the building of the Lucene index as well as updating and deleting documents from the index.
- provides two key methods.
    - void buildIndex(final boolean rebuild) - If rebuild is true, creates a new index, if rebuild is false, updates the existing index.
    - void buildIndex() - Checks the following conditions to determine whether to create a new index or update an existing index.
        - if the Rebuild property in buildIndex.properties is set to true, calls buildIndex(true)

    - if LastBuildDate property in buildIndex.properties is null or empty, calls buildIndex(true)
    - else calls buildIndex(false)

PropertiesDao

- This DAO reads and saves properties files.
- getPropertiesFile(buildIndex) - Retrieves buildIndex.properties.
- storePropertiesFile(buildIndexProperties, buildIndex) - saves the Rebuild and LastBuildDate properties to buildIndex.properties

ProductService

- This service class handles product retrieval.
- list() - Returns list of all products, this is used to create a new index.
- findByModifiedDate(lastBuildDate) - Returns a list of products whose modified date is after the last build date, this is used to update index.
- findByDeletedDate(lastBuildDate) - Returns a list of deleted products whose deleted date is after the last build date, this is used to update index.

IndexDao

- This DAO creates new indices and updates/deletes documents from existing indices.
- createIndex(documentList, locale) - Create a new index with the passed in list of documents.
- updateDocumentInIndex(documentList, product uid key, locale) - Update a document in an existing index.
- deleteDocumentInIndex(termsList, locale) - Delete a document in an existing index.

### *Related Code*

The following packages contain code related to the Lucene search.

- com.elasticpath.domain.search.*
- com.elasticpath.service.index.*
- com.elasticpath.persistence.IndexWriter
- com.elasticpath.persistence.Searcher

# Internationalization and Localization

This section describes various mechanisms in AquaLogic Commerce Services that support multiple languages and currencies. The AquaLogic Commerce Services Storefront can display content in multiple languages and currencies. The Commerce Manager operates in English only,

but includes support for specifying the multiple languages and currencies that are to be displayed in the Storefront.

## *Languages*

This document describes several issues and implementation mechanisms for multi-language support.

## Multi-language support mechanisms

There are four mechanisms used to support multiple languages in the Storefront. Each mechanism has characteristics that make it most appropriate for specific situations.

### Properties files

Properties files are used as the source of language-specific text. The properties files contain simple key value pairs where the key is a description of the String and the value is the corresponding text in a particular language. Multiple properties files with file names that indicate the language are used to support multiple languages. The language-independent keys in the properties files are used to retrieve the corresponding text in Velocity templates using a call to a Velocity macro as shown below.

```
#springMessage("productTemplate.itemsAvailable")
```

The Velocity macro is part of Spring's integration with Velocity, and it will look up the value of the key for the currently selected Locale. Properties files are generally created for each Velocity template and stored in the same location. The properties file has the same name as the template but has the extension ".properties" instead of ".vm". For Strings that are frequently used across multiple pages, the properties are stored in "globals.properties".

This multi-language mechanism is suitable only for static page content coded into Velocity templates.

### Localized Properties

The Localized Properties mechanism is a database-only solution that allows dynamic content such as a store's brands to be displayed in multiple languages. To use this mechanism, a domain object with display values in multiple languages has a reference to a LocalizedProperties object containing the language values retrieved by key and locale. All localized property data for all objects is stored as rows in the TLocalizedProperties table. It is not necessary to change the database schema to add new localized properties. See "How to add Localized Properties to a domain object" below for instructions on adding Localized Property support to a new domain object.

### Locale Dependent Fields

Locale Dependent Fields is used programatically in a simlar way to Localized Properties. An object has a reference to a LocaleDependantFields (LDF) object, from which it can retrieve localized Strings. In this mechanism, the localized properties are stored in columns in the

database table and the table can be joined directly with the parent entity table. This means that performance is better than Localized Properties and the field values can be accessed using methods instead of a map key. However, the disadvantage of this approach is that a schema change is required for each new kind of localized field and a new table is required to support Locale Dependent Fields for each new object that requires them. For this reason, Locale Dependent Fields is only used for performance-critical domain objects such as products and categories.

### Attributes

The attribute system in AquaLogic Commerce Services also supports multiple languages. Attributes have the advantage that they can be created and maintained by a business user through the Commerce Manager user interface. Attributes are relatively slower than Locale Dependent Fields and are only supported for Products, Categories, and SKUs. More information on the attribute system can be found in the attribute technical feature design section (coming soon).

## How to add Localized Properties to a domain object

1. Add a pair of setter and getter to your Java code, see Brand.java and BrandImpl.java for an example.
2. Map LocalizedProperties correctly in your Hibernate mapping file, see Brand.hbm.xml for details.

See "Localized Properties" for information on when to use this multi-language mechanism.

## How to add new properties files

Spring framework provides messaging (i18n or internationalization) functionality by using MessageSource. When an ApplicationContext gets loaded, it automatically searches for a MessageSource bean named "messageSource" defined in the context.

In the Storefront, messageSource bean is defined in /WEB-INF/conf/spring/views/velocity/velocity.xml. The MessageSource implementation used in AquaLogic Commerce Services, ResourceBundleMessageSource, supports a hierarchical structure that defines where properties files should be defined. See the anotated XML code below.

```
<bean id="messageSource"
    class="org.springframework.context.support.
          ReloadableResourceBundleMessageSource">
  <!-- Define the parent message source bean -->
  <property name="parentMessageSource"> <ref
bean="globalMessageSource"/>
  </property>
  <property name="basenames">
    <list>
      <!-- Define .properties files here (omitt the extension) -->
      <value>/WEB-INF/templates/velocity/account/create</value>
```

```
      <value>/WEB-INF/templates/velocity/address/create</value>
      . . .
    </list>
  </property>
</bean>

<bean id="globalMessageSource"
    class="org.springframework.context.support.
           ReloadableResourceBundleMessageSource">
  <property name="basenames">
  <list>
    <!-- Global .properties files are defined here -->
    <value>/WEB-INF/templates/velocity/globals</value>
    <value>org/acegisecurity/messages</value>
  </list>
  </property>
</bean>
```

With this hierarchical structure, the property definition in the files associated with messageSource bean will overwrite the definiton in the files associated with globalMessageSource, if there is any.

To avoid property key duplication among the files in the same hierarchy, property keys are prefixed with the file name. For example, the content of /WEB-INF/templates/velocity/catalog/product/productTemplate.properties will look like the following.

```
productTemplate.zoom=Zoom In
productTemplate.addToWishlist=Add to Wishlist
productTemplate.qty=Qty
productTemplate.itemno=ITEM NO
productTemplate.accessories=Optional Accessories
productTemplate.warranties=Protect this item
productTemplate.upSells=Upgrade to
...
```

## *Currencies*

AquaLogic Commerce Services supports multiple currencies, which are configured in commerce-config.xml. The currently selected currency is stored in the CustomerSession object, which is stored in the web application session.

## *Character Set Encoding*

AquaLogic Commerce Services supports multiple character set encodings. This document describes basic encoding defaults and outlines the changes that are necessary when changing the character set encoding.

### Default Encoding

The default encoding for various components of AquaLogic Commerce Services are as follows.

- Database data (UTF-8)
- Content (UTF-8)
    - html pages
    - emails
- Data files (UTF-16)
    - import data files
    - report files
- URL (UTF-8)
    - catalog browsing URLs
    - search URLs

### Changing the encoding

The following subsections describe how to change the encoding.

#### How to change the encoding for database

You will need to specify the encoding when you create a database and give the same encoding in your JDBC URL.

For example, in MySql you will need to create a Database as shown below to use the Big5 encoding.

```
create database DB_NAME character set big5;
```

The corresponding JDBC URI is shown below.

```
jdbc:mysql://127.0.0.1/alcs?AutoReconnect=true&amp;
        useUnicode=true&amp;characterEncoding=big5
```

> **Note**
>
> You may encounter problems when using some Asian encodings that use a double wide character set, which makes some fields require twice as much storage space. In this case, the schema may need to be updated to accommodate the longer strings.

### How to change the encoding for content

1. Change the configuration "content.encoding" value in elasticpath.xml.
2. Change the following settings in velocity.xml

```
<prop key="template.encoding">UTF-8</prop>
<prop key="input.encoding">UTF-8</prop>
   <prop key="output.encoding">UTF-8</prop>
```

3. Change the following settings in web.xml

```
<!-- Encoding filter -->
  <filter>
    <filter-name>Encoding Filter</filter-name>
    <filter-class>
        com.elasticpath.commons.filter.impl.EncodingFilter
    </filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
     </filter>
```

### How to change the encoding for data files

Change the configuration "datafile.encoding" value in elasticpath.xml.

### How to change the encoding for URLs

Changing the encoding for URLs is not recommended. However, the encoding is defined as a constant URL_ENCODING in WebConstants.java.

# Rules Engine

At the core of the promotion rule system is the JBoss Rules (formerly Drools Rules) library. JBoss Rules is a third-party rules engine that uses a fast algorithm to evaluate rule conditions and execute their actions. The input to the JBoss Rules engine is a set of objects used in the condition evaluation and action execution as well as the set of rules, which we express as text in the proprietary Drools language.

The representation of a Rule in AquaLogic Commerce Services is an object model of the components of a rule such as conditions, actions, and parameters used by various rule elements. The object model is persisted in the database directly with one table corresponding to one class in the rule object model. This allows the graph of rule objects to be easily stored, retrieved and modified, and stored again. The objects in the rule object model are responsible for generating Drools language code that is passed to JBoss Rules. The generated code is not persisted and it is not possible to re-create the object model representation of a rule given the drools code.

The role of the Commerce Manager promotion editor is to allow the user to compose rules from rule elements and then store those rules in the database. The Storefront then retrieves the rules from the database as object graphs, requests the corresponding drools code from the rule objects, and passes the rule code to the JBoss Rules engine. JBoss Rules will then determine which rules' actions should be executed on the Java objects that are passed to it.

JBoss rules supports basic evaluation of objects' properties within the engine itself. However, more complex operations are not supported. In AquaLogic Commerce Services, nearly all conditions are evaluated in Java and the actions are also executed in Java. This allows the condition and action code to be easily debugged and unit tested. The PromotionRuleDelegate is the class that is responsible for computing conditions and executing actions as required by JBoss Rules.

For more detailed information regarding the promotion rule system, see the promotion rule engine technical feature design (coming soon).

# Scheduling

AquaLogic Commerce Services uses the Quartz scheduler to execute scheduled jobs such as computing product recommendations or building a Lucene index. The scheduled jobs are configured in Spring via the quartz.xml configuration file.

## *Adding a new job in quartz.xml*

To add a new scheduled job, perform the following steps.

### Define a job and set its properties

Define a job bean as shown in the code below and set the following properties.

- **targetObject** - the class that contains the logic for the scheduled job.
- **targetMethod** - the method name in the targetObject to execute.
- **concurrent** - set to false to prevent jobs from executing concurrently.

```
<bean id="newJob"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailF
actoryBean">
  <property name="targetObject">
```

```
    <ref bean="newJobService"/>
  </property>
  <property name="targetMethod">
    <value>executeMethod</value>
  </property>
  <property name="concurrent">
    <value>false</value>
  </property>
</bean>
```

## Define the time/trigger details and link the job

Define the trigger bean as shown below and set the following properties.

- **jobDetail** - the job to execute
- **cronExpression** - the cron expression to set how often the job should be executed

```
<bean id="newJobTrigger"
    class="org.springframework.scheduling.quartz.CronTriggerBean">
  <property name="jobDetail">
    <ref bean="newJob"/>
  </property>
  <property name="cronExpression">
    <value>0 0 0/1 * * ?</value>
  </property>
</bean>
```

## Add the trigger to the scheduler factory

Add the new trigger to the scheduler factory bean as shown below.

```
<bean id="schedulerFactory"

class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
  <property name="triggers">
    <list>
      <!-- add the reference to the new trigger here -->
      <ref bean="newJobTrigger" />
    </list>
  </property>
</bean>
```

# Acegi – Security Framework

AquaLogic Commerce Services uses the Acegi security system to handle authentication, authorization, and HTTP/HTTPS switching. Acegi integrates with Spring to provide these services, which are configured in acegi.xml. Acegi is implemented as a chain of web application request filters that perform various security tasks. Acegi is therefore integrated with AquaLogic Commerce Services in web.xml, where the filters are declared. In case you're wondering, Acegi is pronounced "Ah-see-gee" the name has no meaning – it is just the #1, #3, #5, #7, and #9 letters of the alphabet.

For more information on acegi.xml configuration options, see acegi.xml in the deployment guide.

## *Acegi filters*

Following list of available Acegi filters are used in AquaLogic Commerce Services. The bean declaration and configuration for each filter can be found in acegi.xml.

- **channelProcessingFilter** - Determines which transport protocol to be used (HTTP or HTTPS).

The property filterInvocationDefinitionSource is used to specify which pages REQUIRES_SECURE_CHANNEL and which pages REQUIRES_INSECURE_CHANNEL as follows.

```
CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
\A/billing-and-review.ep\Z=REQUIRES_SECURE_CHANNEL
\A/receipt.ep\Z=REQUIRES_SECURE_CHANNEL
\A/.password..ep.*\Z=REQUIRES_SECURE_CHANNEL
\A/sign-in.ep.*\Z=REQUIRES_SECURE_CHANNEL
\A/j_acegi_security_check.ep\Z=REQUIRES_SECURE_CHANNEL
\A/.*\Z=REQUIRES_INSECURE_CHANNEL
```

Note that the order of the entries is critical. The channelProcessingFilter will work from the top of the list down to the **first** pattern that matches the request URL. Accordingly, you should place the most specific expressions first (e.g. a/b/c/d.*), with the least specific (e.g. a/.) expressions last.

- **httpSessionContextIntegrationFilter** - Responsible for storing the SecurityContextHolder contents between invocations.
- **logoutFilter** - This filter processes logout requests, being triggered by a match with a URL configured in the filterProcessesUrl property (e.g. /sign-out.ep). This filter takes several constructor args - the first being the URL of the page to redirect to on successful logout, and the second being a list of logout handlers - an Acegi provided handler (SecurityContextLogoutHandler) performs a logout by clearing the security context and an EP logout handler clears the shopping cart from the session.
- **authenticationProcessingFilter** - This filter processes requests from the sign in form.

- **exceptionTranslationFilter** - Handles any AccessDeniedExceptions or AuthenticationExceptions thrown within the filter chain.
- **filterInvocationInterceptor** - Responsible for handling the security of HTTP resources. It requires an AuthenticationManager and an AccessDecisionManager.
    - AuthenticationManager - Retrieves UserDetails and validates the username and password.
    - AccessDecisionManager - Responsible for making the access control decision based on role voting.

The objectDefinitionSource property is used the configure the role based access control configuration as follows.

```
 CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
\A/manage-account.ep\Z=ROLE_CUSTOMER
\A/edit-account.ep\Z=ROLE_CUSTOMER
\A/create-address.ep\Z=ROLE_CUSTOMER
\A/edit-address.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
\A/update-email.ep\Z=ROLE_CUSTOMER
\A/update-password.ep\Z=ROLE_CUSTOMER
\A/address-preference.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
\A/order-details.ep\Z=ROLE_CUSTOMER
\A/checkout.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
\A/shipping-address.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
\A/checkout-address.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
\A/delivery-options.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
\A/billing-and-review.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
\A/receipt.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
\A/printReceipt.ep\Z=ROLE_CUSTOMER, ROLE_ANONYMOUS_CUSTOMER
```

Note, again, that the order of the entries is critical. The filterInvocationInterceptor will work from the top of the list down to the **first** pattern that matches the request URL. Accordingly, you should place the most specific expressions first (e.g. a/b/c/d.*), with the least specifiec (e.g. a/.) expressions last.

Also note that pages in the customer self-service area cannot be accessed by anonymous customers because of the above role requirements.

## *Authentication process*

When a user signs in, Acegi performs the authentication. The AquaLogic Commerce Services sign-in HTML form sends the user's login information (a username and password) by posting to the /j_acegi_security_check.ep URL (configured in acegi.xml), which is intercepted by the filter, authenticationProcessingFilter. Acegi reads the username and password as form parameters with the pre-defined attribute names, j_username and j_password. The authenticationProcessingFilter

references an instance of CustomerAuthenticationDaoImpl, which is an AquaLogic Commerce Services object wired into Acegi in acegi.xml. CustomerAuthenticationDaoImpl implements the Acegi interface, UserDetailsService, whose loadUserByUsername method is invoked by Acegi to retrieve customer information about the user who is signing in. The CustomerAuthenticationDaoImpl then retrieves and returns the Customer object corresponding to the given username (the email address). The Customer object implements the Acegi UserDetails interface, which Acegi uses to query the password and account status for the user who is attempting to log in. If the password matches, Acegi will update the user's authentication status. Acegi uses a ThreadLocal SecurityContextHolder to store the SecurityContext between web requests. The SecurityContext contains a single getter/setter for the Authentication object that stores the user's authentication status.

EpAuthenticationProcessingFilter extends AuthenticationProcessingFilter to perform additional processing upon a successful login. The EpAuthenticationProcessingFilter notifies the WebCustomerSession it references, which in turn retrieves the customer's CustomerSession object and stores it in the web application session.

### *Authorization process*

When a signed-in user attempts to access a page, the Acegi filterInvocationInterceptor filter will intercept the request and ensure that the user has sufficient permissions. The required permissions for a given page are specified in acegi.xml in as part of the filterInvocationInterceptor configuration. Acegi references the user's UserDetails object to call the getAuthorities() method on it and check that the user has the required authorities to access the page. An "authority" is an implementation of Acegi's GrantedAuthority interface, which has a getAuthority() method that simply returns a String. This String corresponds with the Strings in the filterInvocationInterceptor Spring configuration in acegi.xml. In AquaLogic Commerce Services, CustomerRole objects implement the GrantedAuthority interface and are wired with String authorities in the domainModel.xml Spring configuration file. However, Customer, which implements UserDetails, isn't directly associated with these roles. Customers have a collection of CustomerGroups that they belong to, and CustomerGroups have collections of CustomerRoles. Therefore the getAuthorities() method in CustomerImpl iterates over customer groups to get the collection of Roles (authorities) to return to Acegi.

## Plug-In Architecture

Using this lightweight plug-in architecture based on some Spring functionality, you can easily create a plug-in that will work seamlessly with existing AquaLogic Commerce Services modules.

### *Spring out of the box*

You should already be familiar with how you can just change configuration in Spring bean context files. However, this alone does not make your module completely pluggable - changes still need to be made manually in the Storefront and/or Commerce Manager configuration files.

## *Auto Discovery*

Spring provides some functionality that makes it relatively easy to create pluggable components that can be auto-discovered. Auto-discovery of the code itself is easy - just add your code .jar file to the `WEB-INF/lib` directory of any application that requires it and the code will be automatically added to the classpath. But what about the configuration files?

The following statement in the spring configuration file elastic-path-servlet.xml of the Storefront module will allow it to auto-discover the configuration file of any plug-in added to the classpath

```
<import resource="classpath*:META-INF/conf/plugin.xml"/>
```

Now any plug-in jar which includes a `META-INF/conf/plugin.xml` file will have that file auto-discovered!

## *Self Configuration*

The second addition required is the ability for the plug-in to self-configure - i.e. in a tight integration we will often want to change configuration parameters of existing bean definitions. Fortunately there is now a way to dynamically wire together beans without modifying the original configuration files.

### Introducing PluginBeanFactoryPostProcessor

The key piece of infrastructure which allows modules to be self configuring is a new class in the AquaLogic Commerce Services core module, `com.elasticpath.commons.util.impl.PluginBeanFactoryPostProcessor`. This is an implementation of the Spring `BeanFactoryPostProcessor` interface, which Spring invokes after all of the configuration has been discovered and loaded into an in-memory representation, but **before the actual objects are created**.

For each property/bean you wish to override/extend, you would add a `PluginBeanFactoryPostProcessor` bean definition in your plugin.xml

Suppose we have the following bean definition in the original module

```
<bean id="authenticationProcessingFilter"

class="com.elasticpath.sfweb.filters.EpAuthenticationProcessingFilter
">
  <property name="webCustomerSessionService">
    <ref bean="webCustomerSessionService"/>
  </property>
  <property name="authenticationManager">
    <ref bean="authenticationManager"/>
  </property>
  <property name="authenticationFailureUrl">
```

```
    <value>/sign-in.ep?login_failed=1</value>
  </property>
  <property name="defaultTargetUrl">
    <value>/manage-account.ep</value>
  </property>
  <property name="filterProcessesUrl">
    <value>/j_acegi_security_check.ep</value>
  </property>
</bean>
```

Here's an example of the `PluginBeanFactoryPostProcessor` class in use in a plug-in module's plugin.xml file to override the original bean definition

```
<!-- Override the authenticationProcessingFilter to use the WLS
filter
     and add a customerService property -->
<bean
class="com.elasticpath.commons.util.impl.PluginBeanFactoryPostProcess
or">
  <property name="extensionBeanName">
    <value>"authenticationProcessingFilter</value>
  </property>
  <property name="extensionClassName">

<value>com.elasticpath.plugins.wls_authenticator.filters.WlsAuthentic
ationProcessingFilter</value>
  </property>
  <property name="propertyName">
    <value>customerService</value>
  <property name="propertyValue">
    <ref bean="customerService"/>
  </property>
</bean>
```

The properties are as follows

| Property | Description |
|---|---|
| extensionBeanName | The name of the bean to find an override |
| extensionClassName | (optional) The class to set the overridden bean to use. This property can be omitted to leave the class as assigned in the original bean definition |
| propertyName | The property of the bean to override or add |
| propertyValue | The value for the above property |

Effectively, the above example has the same effect as manually editing the original configuration file like this

```
<bean id="authenticationProcessingFilter" class=
"com.elasticpath.plugins.wls_authenticator.filters.WlsAuthenticationP
rocessingFilter"
>
  <property name="webCustomerSessionService">
    <ref bean="webCustomerSessionService"/>
  </property>
  <property name="authenticationManager">
    <ref bean="authenticationManager"/>
  </property>
  <property name="authenticationFailureUrl">
    <value>/sign-in.ep?login_failed=1</value>
  </property>
  <property name="defaultTargetUrl">
    <value>/manage-account.ep</value>
  </property>
  <property name="filterProcessesUrl">
    <value>/j_acegi_security_check.ep</value>
  </property>
  <property name="customerService">
    <ref bean="customerService"/>
  </property>
</bean>
```

## Adding Functionality

Additional functionality can be easily added to this architecture by adding to the processing done in the `PluginBeanFactoryPostProcessor.postProcessBeanFactory()` method, for example you could easily add an `extendList` property to optionally allow a property value to add to a bean's list property rather than overwrite the list, or provide support for multiple properties in a <props> element.

### *Example*

For an example of a fully functional plug-in including self-configuration, see the Weblogic Authentication Plug-In, com.elasticpath.plugins.wls-authenticator

### *Summary*

As you can see, using the above functionality to add auto-discovery and self-configuration means you can now create a plug-in module where all the classes and configuration are contained within a single jar file which can just be dropped in to the classpath - no additional configuration required! The plug-in module's `plugin.xml` file will configure any new beans and use `PluginBeanFactoryPostProcessor` beans to override classes/properties of beans already defined in the base modules.

# 4 - Building AquaLogic Commerce Services Applications

This section lists a few of the more commonly used ANT tasks. A full listing of Ant tasks is available in each build.xml file.

**Assumptions** : your current directory is the working directory of AquaLogic Commerce Services, and you have setup Ant according to the instructions in Setting up Ant.

**Tip**
To determine the possible targets after successfully running the setup you should be able to use 'ant -p' in any project directory.

## Environment configuration

Ant needs some properties describing your development environment.

- Copy 'env.config.for.developer' in the AquaLogic Commerce Services working directory to 'env.config' and modify env.config based on your environment settings.
    - Typical **development** settings are:
        - ep.log.everything=false
        - ep.log.level=DEBUG
        - ep.log.to=CONSOLE
        - ep.hibernate.showsql=true
        - ep.dwr.debug=true
        - ep.velocity.library.autoreload=true
        - ep.velocity.cache.seconds=3
        - ep.hibernate.2nd.lvl.cache.enable=false
        - ep.disable.index.build=true
        - ep.disable.rule.compile=true
        - ep.disable.top.seller.build=true
        - ep.disable.product.recommendation.build=true
    - Typical **production** settings are:
        - ep.log.everything=false
        - ep.log.level=INFO

- ep.log.to=FILE
- ep.hibernate.showsql=false
- ep.dwr.debug=false
- ep.velocity.library.autoreload=false
- ep.velocity.cache.seconds=-1
- ep.hibernate.2nd.lvl.cache.enable=true
- ep.disable.index.build=false
- ep.disable.rule.compile=false
- ep.disable.top.seller.build=false
- ep.disable.product.recommendation.build=false

## Creating the database

- To create an AquaLogic Commerce Services database for the dbms you are using, run the appropriate command:

```
#> ant \-f database/build.xml create-mysql
#> ant \-f database/build.xml create-mssql
#> ant \-f database/build.xml create-postgresql
#> ant \-f database/build.xml create-oracle
#> ant \-f database/build.xml create-db2
```

To create SQL insert scripts, run the appropriate Ant target

- MySQL example: #> ant -f database/build.xml sql-mysql

## Building the applications

- To do a complete build, enter your AquaLogic Commerce Services working directory, and run command
    - #>ant all
- To build the com.elasticpath.sf project
    - #>ant -f com.elasticpath.sf/build.xml build
- To build the com.elasticpath.cm project
    - #>ant -f com.elasticpath.cm/build.xml build

**Developer Tip**
Notice: if your application server is started in debugging mode, you can change code in com.elasticpath.core project dynamically. Your changes will take effect (hot swapped) immediately unless you changed the interface/method signature (ex. such as adding/removing methods).
However, if you stop your application server, you will still need to run the above ant tasks to apply the changes statically.
If you changed the interface, you need to run above tasks and restart your

application server.

When you add/change/delete **hibernate mapping** files and the **spring configuration** file - domainModel.xml in either com.elasticpath.core, com.elasticpath.sf or com.elasticpath.cm projects, you may have to rebuild the SF or CM projects for the changes to take effect.

## Testing Your Code

You can run testing tasks for each of the projects independently.

Running the Cobertura test coverage Ant task requires that you:

- download the the JavaNCSS version 21.41 zip file: http://www.kclee.de/clemens/java/javancss/javancss21.41.zip
- extract ccl.jar, rename it to javancss-ccl-21.41.jar and place it into /LIBS/javancss/ccl
- extract javancss.jar, rename it to javancss-javancss-21.41.jar and place it into /LIBS/javancss/javancss

When you add/change/delete java code in a project, from that project's root directory you can run various checks and testing tasks:

- To run junit tests
    - #>ant junit
- To run pmd (To check for good coding practices)
    - #>ant pmd
- To run checkstyle (To check your code for formatting and style)
    - #>ant checkstyle

**Tip**
Normally you don't need to run pmd and checkstyle ant tasks if you correctly setup pmd and checkstyle in Eclipse.
The plugins can report the same warnings and errors in Eclipse. In addition, you can run junit tests from within Eclipse by selecting a Test class and choosing "Run as... Junit Test".

## Torque Data Changes

When you add/change/delete **torque schema** and **data files** in the database/src directory

- To apply changes to your database
    - $ant -f database/build.xml create-DBMS_NAME

## New Libraries / Jar files

When libraries are added to the $EP_LIB directory, the appropriate Ant scripts have to be made aware of the change. See the Customizing and extending AquaLogic Commerce Services applications for more information on how to do that.

Assuming your Ant scripts have been updated so that they're aware of the new libraries, you can let Eclipse know about them with:

1. #>ant -f ant/setup/setup.xml setup
2. #>ant eclipse-setup-all
3. Refresh the projects in Eclipse.

## Source Included

By default, the .JAR file or .WAR files generated by Ant scripts don't contain the source code. If you want to include the source files, please use the Ant property similar to these examples:

$ ant -f com.elasticpath.core/build.xml **jar** -Dswitch.archive.source="yes"

$ ant -f com.elasticpath.sf/build.xml **war** -Dswitch.archive.source="yes"

## Cleaning

From time to time things may get out of sync, such as old JAR files which are no longer used but are still lingering around in the system and causing problems. When this happens, you should trying **cleaning**.

- ant clean-all
    - cleans all sub-projects
- ant -f com.elasticpath.PROJECT_NAME/build.xml clean
    - cleans one project

## Updating the dojo library

The dojo library is needed in template-resources/js/dojo to get the Commerce Manager running. Also, some customized dojo code is saved in template-resources/js/ep-dojo - the custom code overwrites some dojo files. To get it all set up, run

- #>ant -f com.elasticpath.cm/build.xml **dojo**