ORACLE®

# ATG WEB COMMERCE

Version 10.2

# Upgrade and Migration Guide

**10.1.2 to 10.2**

**Oracle ATG**
**One Main Street**
**Cambridge, MA 02142**
**USA**

## Oracle ATG Web Commerce Upgrade and Migration Guide 10.1.2 to 10.2

### Document Version

ATG10.2 MIGRATIONv2 4/26/13

### Copyright

# Contents

# 1 Introduction

This guide describes how to upgrade from Oracle ATG Web Commerce 10.1.2 to 10.2. It is written for programmers, DBAs, site administrators, and ATG partners.

Before you begin migration:

1. Review the New in ATG 10.2 chapter for information on new features.
2. Check the Supported Environments information on the My Oracle Support Web site. Make sure you are running on a supported version of your application server, JDK, database server, and JDBC driver.
3. Review the Migration Steps Overview chapter.

This guide uses the convention `<ATGdir>` to represent the root directory for ATG products. By default, the Windows installation creates the root directory `C:\ATG\ATG10.2`, but its actual location can vary according to your installation.

For detailed information about ATG products, see the ATG documentation on the Oracle Technology Network Web site.

**Important:** You must upgrade your entire Oracle ATG Web Commerce installation; you cannot upgrade only part of it. Oracle does not support running different versions of Oracle ATG Web Commerce products together. For example, using the 10.2 platform with an older version of Commerce Service Center is not supported. The platform and all applications must have the same version number.

# 2 New in ATG 10.2

This chapter describes new features available in the ATG platform and applications.



## Oracle ATG Web Commerce Platform

This section describes new platform features.

### Site Administration

Profile realms provide multisite profile management support. For more information, see the *ATG Multisite Administration Guide*.

### REST Web Services

A new extensible REST *architecture* (the REST MVC Web Services API) is available in addition to the legacy REST API. For more information, see the *ATG Web Services Guide*.

### ATG Content Administration

The following features are new to ATG Content Administration. For more information, see the *ATG Content Administration Programming Guide*.

- An extensible asset purge function removes unneeded repository items.
- Emergency workflows are now available for situations where an emergency project must be updated within minutes, bypassing standard deployment.

### Oracle ATG Web Commerce

Cross-channel inventory visibility adds in-store pickup functionality, and enhances the current ATG Inventory system to accommodate multiple locations for any given SKU. For more information, see the *ATG Commerce Guide to Setting Up a Store* and the *ATG Commerce Programming Guide*.

### Oracle ATG Web Commerce Merchandising

The following features are new to Oracle ATG Web Commerce Merchandising for version 10.2. For more information about these features, see the *ATG Business Control Center Administration and Development Guide* and the *ATG Merchandising Guide for Business Users*.

- New roles and access rights help control the content that business users can access in the Business Control Center.
- Configurable filters can now restrict how promotions are applied to qualifiers and target items.
- In Oracle ATG Web Merchandising layout view, Promotions Visibility features analyze how promotions behave when preview users shop for items. You can also edit promotions directly from layout view.
- Promotions now support non-discountable items.
- The workbench pane that appears at the top of the Merchandising window bookmarks assets for organization and editing. (Previously, the workbench could be used only for Multi Edit operations.)
- New UI for working with internal, preview, and external users.
- New Store Location assets contain information about physical "brick and mortar" stores, such as addresses and hours.
- The existing rich text editor has been replaced with FCKeditor HTML editor. FCKeditor is installed automatically with the Oracle ATG Web Commerce applications that require it.
- In multisite environments, administrators can limit Merchandising users' access to catalog assets and promotions based on their site affiliation.
- All personalization items can be configured to use site groups.

## Oracle ATG Web Commerce Reference Store

The following features are new to Oracle ATG Web Commerce Reference Store for version 10.2. For more information about these features, see the *ATG Commerce Reference Store Overview*.

- Updated code to improve checkout performance.
- Example of Oracle ATG Web Commerce returns functionality.
- Optional integration with Oracle RightNow Knowledge Cloud Service, which displays context-sensitive knowledgebase and support information.
- Integration with Oracle Endeca Commerce. All search facilities use Oracle Endeca Commerce and the Commerce Reference Store category pages and search results pages are Endeca-driven.
- A build environment allows you to quickly and easily rebuild the Commerce Reference Store modules using the Apache Ant build tool.

**Note:** There is no migration path from older versions of Commerce Reference Store to version 10.2. To use version 10.2, install and configure a new instance of Commerce Reference Store using the instructions in the *ATG Commerce Reference Store Installation and Configuration Guide*.

# Commerce Reference Store iOS Universal Application

ATG Web Commerce 10.2 includes the first release of the Commerce Reference Store iOS Universal Application (CRS-IUA) that serves both iPhone and iPad. For more, see the *CRS-IUA Overview*.

# Oracle ATG Web Commerce Business Intelligence

The following features are new to Oracle ATG Web Commerce Business Intelligence. For more information, see the *ATG Data Warehouse Guide* and the *ATG Reports Guide*.

- Data aggregation to improve reporting performance in the Business Intelligence data warehouse.
- Usability improvements in Business Intelligence reports.

# Oracle ATG Web Commerce Service Center

The following features are new to Oracle ATG Web Commerce Service Center. For more information, see the *ATG Commerce Service Center Installation and Programming Guide* and the *ATG Commerce Service Center User Guide*.

- Optional integration with Oracle Endeca Commerce for catalog search.
- Commerce Service Center can be configured to use in-store pickup, which allows the customer to go to a store near them and pickup the merchandise. Returns with in-store pickup are also supported.
- Support for profile realms, which provide multisite profile management support.
- Optional integration with Oracle Live Help On Demand Click to Call.

.

# 3 Migration Steps Overview

Migrating from ATG 10.1.2 to ATG 10.2 can be divided into the following stages:

**Pre-Migration Tasks**

**ATG installation**

**Database Schema Migration**

**Application-Specific Migration Tasks**

**Post-Migration Tasks**

**Migration Testing**

## Pre-Migration Tasks

**Note:** In addition to the tasks listed here, a number of applications have their own pre-migration requirements. Before starting the migration, check the instructions for each installed application, under Migration Notes.

Before starting migration, complete the following tasks:

1. Back up your ATG 10.1.2 database.
2. Back up all of your `Publishing` and `PublishingAgent` directories—-for example, these CIM-generated directories under `<ATG10dir>/home/servers/`:
   - `atg_publishing_lockserver/Publishing`
   - `atg_production_lockserver/PublishingAgent`
3. If your installation includes ATG Content Administration, complete all projects.

   **Note:** Solution workflow projects can remain open during the migration process.
4. Shut down any Oracle ATG Web Commerce Outreach campaigns and Campaign Optimizer tests that are running.
5. Process all reporting event logs. On the data warehouse load server, execute the `loadAllAvailable` method on all loaders in `/atg/reporting/datawarehouse/loaders/`.
6. Shut down all ATG servers.
7. Shut down any search engines your environment uses.
8. Remove ATG artifacts, such as server instances, EAR files, and data sources, from the application server.

# ATG Installation

1. Install ATG 10.2 as directed in the *ATG Installation and Configuration Guide*.
2. Download the migration kits necessary for your products. Create the following directories as necessary and unpack the migration kits to the new directories:
   - Oracle ATG Web Commerce platform:

     `<ATGdir>/migration/101to102sql`
   - Oracle ATG Web Commerce Service Center:
     `<ATGdir>/`
3. Copy your existing configuration files and application modules to the ATG 10.2 installation. Update application code as necessary to conform to new ATG 10.2 functionality (see the Migration Notes chapter).
4. Copy the directories listed from your ATG 10.1.2 installation into your ATG 10.2 installation:

   ```
   /home/Publishing/versionFileStore
   /home/servers/atg_svcagent_lockserver/publishing
   /home/servers/atg_production_lockserver/PublishingAgent
   /home/servers/atg_svcagent_lockserver /PublishingAgent
   /home/servers/atg_staging/PublishingAgent
   ```
5. If you use a Publishing Web Agent server, copy `/home/PublishingWebAgent` from your ATG 10.1.2 installation to your ATG 10.2 installation.

# Database Schema Migration

Several migration kits are available for upgrading the schemas of supported databases: Oracle, MSSQL, and DB2. Migration kits are available for download from My Oracle Support.

The following migration kits are available for ATG products:

- Oracle ATG Web Commerce platform
- Oracle ATG Web Commerce Service Center

Each migration kit contains three database-specific directories:

```
/db_components/oracle
/db_components/db2
/db_components/mssql
```

Run the scripts in these directories as directed in the Database Migration chapter.

## Batch Files versus SQL Scripts

The batch files/shell scripts run all required DDL scripts in the correct order. Instead of running batch files/shell scripts, you can execute individual SQL scripts. Be sure to run these in the same order as they appear in the batch file or shell script.

## Before Running Migration Scripts

Review the migration scripts and modify where necessary. Make sure the scripts do not overwrite custom database changes. In order to run these scripts, your database login must be the same as the one used to create the original ATG 10.2 schemas.

Before you run the migration batch files or shell scripts, include a dot (`.`) in the PATH environment variable to enable execution of DDL script commands that may be internally called.

## Running Batch Files

Use the syntax shown in the following sections to run batch files or shell scripts.

### *Oracle*

*filename*.{bat|sh} *user-acct password tns-alias*

- *user-acct*: Name of the schema user account
- *password*: Password to the user account
- *tns-alias*: TNS name for the database

### *MSSQL*

*filename*.bat *user-acct password hostname db-schema*

- *user-acct*: Name of the schema user account
- *password*: Password to the user account
- *hostname*: Host name of the database server
- *db-schema*: Name of the database schema

### *DB2*

*filename*.{bat|sh} *user-acct password DB2-alias*

- *user-acct*: Name of the schema user account
- *password*: Password to the user account
- *DB2-alias*: Alias for the DB2 database

## Running DDL Scripts

Use the syntax shown in the following sections to run DDL scripts.

### *Oracle*

```
sqlplus –S user-acct/password@tns-alias < ddl-pathname > logfile
```

### *MSSQL*

```
sqlcmd -U user-acct -P password -S hostname -d db-schema
 -i ddl-pathname -o logfile
```

### *DB2*

```
db2 –tvf ddl-pathname > logfile
```

# Application-Specific Migration Tasks

After completing all tasks described in previous sections, complete migration for specific ATG applications as documented under Migration Notes.

# Post-Migration Tasks

After migration is complete, perform the following tasks:

1. If using ATG Content Administration, update view mappings to receive user interface updates.
2. If using ATG Content Administration, it is good practice to launch a full deployment to all workflow targets. This ensures that content is fully synchronized between the asset management server and workflow targets.

   **Note:** Before launching a full deployment, remap the repositories for your site in the CA Console. This step is necessary because a number of repositories (such as `ProductCatalog` and `Site`) are secured by default in this release.
3. Clear browser caches.
4. Clear the application server cache.

   For example, on JBoss, empty the contents of each server directory:

   `/work/jboss.web/localhost`
5. It is good practice to restart any existing scenarios.
6. Test the upgraded platform and applications (see Migration Testing).

# Migration Testing

To verify the success of your migration on asset management and production servers, follow these steps:

1. Assemble EAR files for the asset management and production servers.

   **Important:** If you installed and configured a preview server and imported preview data from `DPS-UI/install/data/viewmapping_preview.xml`, `SiteAdmin/Versioned/install/data/viewmapping_preview.xml`, and `DCS-UI/install/data/viewmapping_preview.xml`, you must assemble the versioned preview application and the Business Control Center with the `-layer Preview` switch. For more information about setting up preview, see the *Business Control Center Administration and Development Guide*.
2. Deploy the EAR files to the application server and start the production and asset management servers.
3. From the ATG 10.2 Business Control Center, verify that you can view:
   - All internal users created in ATG 10.1.2
   - All assets created in ATG 10.1.2
4. Confirm that the ATG 10.2 production server contains all assets that were accessible on the ATG 10.1.2 platform.

5. Create projects, where you add new assets and edit existing ones. Deploy these assets and verify that the deployment is successful.
6. Confirm that deployed file assets are in the correct location and behave as expected.

# 4 Database Migration

ATG 10.2 includes a number of enhancements to the database schemas used by ATG 10.1.2. The migration kits provide the batch files/shell scripts and DDL scripts needed to upgrade existing schemas in the following migration kit directory:

- Oracle:

  `../db_components/oracle`

- DB2:

  `../db_components/db2`

- MSSQL:

  `../db_components/mssql`

**Note:** Due to the merging of B2B and B2C features in Oracle ATG Web Commerce 10.1, separate B2B migration scripts are no longer required.

The scripts for Oracle and DB2 databases are the same, but located in different directories. Use the specified scripts to migrate your Oracle ATG Web Commerce platform database. Scripts can be found in the platform migration kit unless otherwise specified.



Also see the Data Migration chapter for additional scripts that may be required, depending on your installed products.

**Note:** Some Commerce Service Center scripts drop unnecessary tables and create new ones. If your environment does not already include ATG Knowledge Manager and Self Service, the drop logs will contain errors for tables or views that do not exist.

## Production Core Schema Migration Scripts

If you do not use a switching database, run the scripts listed in both this section and the Switching Schema Migration Scripts section on your production core schema. It is strongly recommended to use a switching schema with ATG products.

| Product | Migration Script | Prerequisites |
|---|---|---|
| Oracle ATG Web Commerce Platform | `run_das_core.{bat\|sh}`<br>`run_dps_core.{bat\|sh}` | None |
| Oracle ATG Web Commerce | `run_dcs_core.{bat\|sh}` | ATG Platform |
| Oracle ATG Web Commerce Service Center | `run_service_production.{bat\|sh}` | ATG Platform; ATG Search or Endeca MDEX; B2CCommerce or Commerce Reference Store |

# Switching Schema Migration Scripts

| Product | Migration Script | Prerequisites |
|---|---|---|
| Oracle ATG Web Commerce | `run_dcs_switching_all.{bat\|sh}` | ATG Platform |
| Oracle ATG Web Commerce Service Center | `run_service_switching.{bat\|sh}` | ATG Platform |

# Publishing Schema Migration Scripts

| Product | Migration Script | Prerequisites |
|---|---|---|
| Oracle ATG Web Commerce | `run_publishing_dcs_all.{bat\|sh}` | ATG Platform |
| Oracle ATG Web Commerce Service Center | `Run_service_management.{bat\|sh}` | ATG Platform |

# Data Warehouse Schema Migration Scripts

| Product | Migration Script | Prerequisites |
|---|---|---|
| Base data warehouse | `run_arf_dw_base.{bat\|sh}` | None |
| Oracle ATG Web Commerce | `run_dcs_dw.{bat\|sh}` | Base data warehouse |

# Agent Schema Migration Scripts

| Product | Migration Script | Prerequisites |
|---|---|---|
| Oracle ATG Commerce Service Center | **Platform migration kit**: `run_publishing_all.{bat\|sh}` **Service migration kit**: `run_service_agent.{bat\|sh}` | None |

# 5 Data Migration

After updating your database schemas, you may need to perform additional steps that affect the data in your database and repositories. The sections that follow explain how to perform these tasks.

This chapter includes the following sections:



## Updating View Mappings

Many ATG user interfaces rely on view mappings. After you run the appropriate migration scripts for your environment, you must update your view mappings.

Before importing any view mappings, you must configure the data sources used by the import utility. Configure a `home/localconfig/atg/dynamo/service/jdbc/FakeXADataSource.properties` file to refer to your publishing schema, and another with the name `FakeXADataSource_production.properties` for your production schema. An example `FakeXADataSource` file that refers to the publishing schema follows:

```
$class=atg.service.jdbc.FakeXADataSource
driver=oracle.jdbc.OracleDriver
URL=jdbc:oracle:thin:@your_host_name:1521:utf8112
user=username
password=password
```

If you are using switching data sources, configure the following files:

- `FakeXADataSource_switchA.properties` to refer to your `SwitchingA` schema.
- `FakeXADataSource_switchB.properties` to refer to your `SwitchingB` schema.
- `SwitchingDataSourceA.properties` to refer to your `FakeXADataSource_switchA.properties` file.
- `SwitchingDataSourceB.properties` to refer to your `FakeXADataSource_switchB.properties` file.

For example:

```
$class=atg.service.jdbc.MonitoredDataSource
dataSource=/atg/dynamo/service/jdbc/FakeXADataSource_switchA
```

Run the following scripts on the Asset Management servers for all products:

```
/bin/startSQLRepository -m BIZUI –repository
/atg/web/viewmapping/ViewMappingRepository –import
"<ATGdir>/home/../BIZUI/install/data/viewmapping.xml"

bin/startSQLRepository -m BCC -repository
/atg/web/viewmapping/ViewMappingRepository -import
"<ATGdir>/home/../BCC/install/data/viewmapping.xml"

bin/startSQLRepository -m DPS-UI -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DPS-
UI/AccessControl/install/data/viewmapping.xml"

bin/startSQLRepository -m DPS-UI -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DPS-
UI/install/data/viewmapping.xml"

bin/startSQLRepository -m AssetUI -repository
/atg/web/viewmapping/ViewMappingRepository -import
"<ATGdir>/home/../AssetUI/install/data/viewmapping.xml"

bin/startSQLRepository -m AssetUI -repository
/atg/web/viewmapping/ViewMappingRepository -import
"<ATGdir>/home/../AssetUI/install/data/assetManagerViews.xml"

bin/startSQLRepository -m SiteAdmin.Versioned -repository
/atg/web/viewmapping/ViewMappingRepository -import
"<ATGdir>/home/../SiteAdmin/Versioned/install/data/viewmapping.xml"

bin/startSQLRepository -m DPS-UI.Versioned -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DPS-
UI/Versioned/install/data/viewmapping.xml"

bin/startSQLRepository -m DPS-UI.Versioned -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DPS-
UI/Versioned/install/data/examples.xml"

bin/startSQLRepository -m DPS-UI -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DCS-
UI/install/data/viewmapping.xml"

bin/startSQLRepository -m DPS-UI.Versioned -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DCS-
UI/Versioned/install/data/viewmapping.xml"

bin/startSQLRepository -m DCS-UI.SiteAdmin.Versioned -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DCS-
UI/SiteAdmin/Versioned/install/data/viewmapping.xml"
```

**Important:** Before running the `ImportDCSUISearchCustomCatalogs` script, comment out the following:

```
bin/startSQLRepository -m DCS.Search.CustomCatalogs.Versioned -m
DCS.Versioned -repository /atg/search/repository/RefinementRepository -
import "${DYNAMO_HOME}/../DCS-UI/Search/install/data/refinement.xml" -
workspace RefinementRepositoryImport:main -comment GlobalFacetDefinition
```

After making the change, run the following scripts:

```
<ATGdir>/home/../DCS-UI/Search/install/importDCSUISearchCustomCatalogs.sh
```

```
bin/startSQLRepository -m BIZUI -repository
/atg/web/viewmapping/ViewMappingRepository -import "${DYNAMO_HOME}/../DCS
-UI/Search/install/data/viewmapping.xml"
```

```
bin/startSQLRepository -m BIZUI –repository
/atg/web/viewmapping/ViewMappingRepository -import "${DYNAMO_HOME}/../DCS
-UI/Search/install/data/flex/viewmapping.xml"
```

```
bin/startSQLRepository -m BIZUI –repository
/atg/web/viewmapping/ViewMappingRepository –import
"${DYNAMO_HOME}/../AssetUI/Search/install/data/viewmapping.xml"
```

**Important:** Run the following three scripts only if you installed and configured a preview server on your Asset Management server. These import scripts configure some viewmapping components specifically for preview and require the preview layer. (For more information, see Migration Testing.)

```
bin/startSQLRepository -m DPS-UI -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DPS-
UI/install/data/viewmapping_preview.xml"
```

```
bin/startSQLRepository -m SiteAdmin.Versioned -repository
/atg/web/viewmapping/ViewMappingRepository -import
"<ATGdir>/home/../SiteAdmin/Versioned/install/
data/viewmapping_preview.xml"
```

```
bin/startSQLRepository -m DCS-UI -repository
/atg/web/viewmapping/ViewMappingRepository -import "<ATGdir>/home/../DCS-
UI/install/data/viewmapping_preview.xml"
```

# Updating Internal Users

This section describes how to import new roles and access rights for all existing internal users. In this release, a number of repositories, such as `ProductCatalog`, `SEORepository`, and `Site`, are secured by default using these roles and access rights in their ACLs.

Before importing any internal user data, you must configure the data sources used by the import utility. Configure a `home/localconfig/atg/dynamo/service/jdbc/FakeXADataSource.properties` file to refer to your publishing schema, and another named `FakeXADataSource_production.properties` for your production schema. An example `FakeXADataSource` file that refers to the publishing schema follows:

```
$class=atg.service.jdbc.FakeXADataSource
driver=oracle.jdbc.OracleDriver
URL=jdbc:oracle:thin:@your_host_name:1521:utf8112
user=username
password=password
```

If you are using switching data sources, configure the following files:

- FakeXADataSource_switchA.properties to refer to your SwitchingA schema.
- FakeXADataSource_switchB.properties to refer to your SwitchingB schema.
- SwitchingDataSourceA.properties to refer to your FakeXADataSource_switchA.properties file.
- SwitchingDataSourceB.properties to refer to your FakeXADataSource_switchB.properties file.

For example:

```
$class=atg.service.jdbc.MonitoredDataSource
dataSource=/atg/dynamo/service/jdbc/FakeXADataSource_switchA
```

Run the following scripts on the Asset Management servers for all products:

```
<ATGdir>/home/../DPS/InternalUsers/install/importDPSInternalUsers.bat|sh
```

```
/bin/startSQLRepository -m DPS.InternalUsers -repository
/atg/userprofiling/InternalProfileRepository –import
"<ATGdir>/home/../DPS/InternalUsers/install/data/searchadmin-security.xml"
```

```
/bin/startSQLRepository -m DCS.Versioned -repository
/atg/userprofiling/InternalProfileRepository –import
"<ATGdir>/home/../ DCS/Versioned/install/data/internal-users-security.xml"
```

**Important:** Before running users.xml, comment out the following line:

```
<set-property name="password"><![CDATA[${merchandisingPassword}]]></set-
property>
```

After making the change, run the following script:

```
/bin/startSQLRepository -m DPS-UI.Versioned -repository
/atg/userprofiling/InternalProfileRepository –import
"<ATGdir>/home/../DCS-UI/Versioned/install/data/users.xml"
```

```
/bin/startSQLRepository -m DSS.InternalUsers -repository
/atg/userprofiling/InternalProfileRepository –import
"<ATGdir>/home/../Publishing/base/install/epub-role-data.xml"
```

```
/bin/startSQLRepository -m SiteAdmin.Versioned -repository
/atg/userprofiling/InternalProfileRepository –import
"<ATGdir>/home/../SiteAdmin/Versioned/install/data/siteadmin-role-
data.xml"
```

```
/bin/startSQLRepository -m BIZUI -repository
/atg/userprofiling/InternalProfileRepository –import
"<ATGdir>/home/../BIZUI/install/data/profile.xml"
```

**Important:** Before running profile.xml, comment out the following line:

```
<set-property name="password"><![CDATA[${adminPublishingPassword}]]></set-
property>
```

After making the change, run the following script:

```
/bin/startSQLRepository -m BIZUI -repository
/atg/userprofiling/InternalProfileRepository –import
"<ATGdir>/home/../WebUI/install/data/profile.xml"
```

**Important:** Before running the dynAdminRepo.xml script, comment out the following:

```
set-property name="password"><![CDATA[${dynAdminPassword}]]></set-
property>
```

After making the change, run the following scripts:

```
/bin/startSQLRepository -m DAS -repository
/atg/dynamo/security/AdminSqlRepository -import
$DYNAMO_HOME/../DAS/install/data/dynAdminRepo.xml
```

```
/bin/startSQLRepository -m WebUI -repository
/atg/userprofiling/ProfileAdapterRepository -import
$DYNAMO_HOME/../WebUI/install/data/external_profile.xml
```

After you have imported all new roles, run the following script from Oracle ATG Web Commerce platform migration kit:

```
run_upgrade_dps_internal_user_roles.{bat|sh}
```

**Note:** To update Commerce Service Center roles, see Updating Commerce Service Center Roles.

# ATG Commerce Service Center Framework Migration

This procedure is necessary only if you have custom framework data.

1. If you are using a non-switching datasource, configure the <ATG10dir>/home/servers/agent_import101to102/localconfig/atg/svc/ui/framework/ServiceFrameworkRepository_read.properties file as shown:

   ```
   dataSource=/atg/dynamo/service/jdbc/JTDataSource_production
   ```

   If you are using a switching datasource, configure <ATGdir>/home/servers/agent_import100to101/localconfig/atg/svc/ui/framework/ServiceFrameworkRepository_read.properties as shown:

   ```
   dataSource=/atg/dynamo/service/jdbc/SwitchingDataSourceA
   ```

2. Create a <ATGdir>/home/localconfig/atg/svc/framework/ directory.
3. Change to the <ATGdir>/Service10.2/Service/DBMigration/FrameworkDataMigration/scripts directory.
4. Change to the <ATGdir>/../Service10.2/Service/DBMigration/FrameworkDataMigration/scripts directory and run the following script:

   ```
   run_csc_framework_data_migration.{bat|sh} agent_import101to102
   ```

   This script creates an <ATG10dir>/home/localconfig/atg/svc/framework/service_framework_csc.xml file that identifies differences between default databases and your customized databases.
5. Rename the output XML file that is generated by the script you run to serviceFramework.xml and add it to your customization module.

6. Back up the <ATGdir>/home/localconfig/atg/svc/framework/ directory.
7. Reassemble and redeploy your production and agent servers.

# 6 Migration Notes

Each section in this chapter focuses on applications that have their own migration-related requirements, beyond the database schema changes described in the previous chapter.

Any applications not listed here can be assumed to have no additional migration steps.

This chapter contains the following sections:



## Oracle ATG Web Commerce Notes

A major feature of ATG 10.1 was the merge of Commerce B2B and B2C features. Due to this merge, you must make changes to your Oracle ATG Web Commerce B2B applications (B2C applications do not need to make any changes). However, you did not need to make these changes when you migrated to ATG 10.1.

If you decided to postpone your application changes when you upgraded to 10.1, you must make them now as part of the 10.2 upgrade because the B2BCommerce module has been removed from this release. The rest of this section describes these changes.

First, change any `manifest.MF` entries that refer to either the B2BCommerce or B2CCommerce modules. Both types of application should refer to the DCS module directly.

Second, if you have extended any of the pipeline chains in `commercepipeline.xml`, examine your XML files to make sure they get the desired results. The merge inserts some formerly-B2B pipeline links into the chains. It also moves three links (`setStimulusMarkers`, `setSalesChannel`, `setSubmittedSite`) to an earlier position in the `processOrder` chain than they had in previous Commerce versions. If you have altered those links, you may need to do so again to preserve transitions.

Third, any B2B code you have written that expects a return type in the `atg.b2bcommerce` package must change its declaring type to the corresponding superclass in the `atg.commerce` package. All B2B-specific methods were moved into DCS and their return type changed to the superclass. Failure to make this change will result in compilation errors.

For example, the following code will result in errors:

```
import atg.b2bcommerce.order.CostCenterManager
import atg.b2bcommerce.order.CostCenter
...
CostCenterManager costCenterManager;
CostCenter newCostCenter;

costCenterManager =
Nucleus.getGlobalNucleus.resolveName("/atg/commerce/order/CostCenterManager");

newCostCenter = costCenterManager.createCostCenter("identifier");
```

The last line would result in a compile-time "incompatible types" error now that the `createCostCenter` method returns `atg.commerce.order.CostCenter` instead of`atg.b2bcommerce.order.CostCenter`.

You can fix this by changing the import to `atg.commerce.order.CostCenter`.

# Updating Commerce Service Center Roles

These steps insure that the rights and roles for Commerce Service Center are migrated correctly:

1. Run the agent schema migration with the exception of the `upgrade_svc_super_admin_role.sql` file.
2. Configure the `/atg/dynamo/service/jdbc/FakeXADataSource.properties` file to point to your agent schema in your agent import server. For example: `DYNAMO_HOME/servers/`*`agent import server`*`/localconfig/atg/dynamo/service/jdbc/FakeXADataSource.properites`).
3. Remove or rename the `JTDataSource` and `DirectJTDataSource` files if they exist on your import server. Revert these changes after you run the scripts.
4. Run the `Service10.2/migration/101to102sql/scripts/run-agent-import.bat|sh` script.
5. Run the `Service10.2/migration/101to102sql/db_components/`*`db_type`*`/run_upgrade_svc_super_admin_role.bat|sh` *`user_name password db_specific_alias`* script.

After you have migrated your database schemas and data, perform the following post-migration procedure: Create Search Environment.

## Create Search Environment

After completing the database migration tasks (see the pertinent section for your database), you must create an ATG Search live indexing search environment to perform profile and order searches. Oracle ATG Web Commerce Service Center uses an embedded search method for customer profile and order searches that are included in the `DPS.Search.Index` and `DCS.Search.Order.Index` modules on each customer-facing and asset management server. For additional information, refer to the *ATG Commerce Service Center Installation and Programming Guide*.

1. Open the Dynamo Server Admin on the agent server.
2. Open the Nucleus component `/atg/search/routing/LiveIndexingService`.

3. Create the ATGProfile search environment by entering the details of your search engine.
4. Create the ATGOrder search environment by providing the search engine information.
5. Open the component `/atg/userprofiling/search/ProfileOutputConfig` and execute the `bulkLoad` method.
6. Open the component `/atg/commerce/search/OrderOutputConfig` and execute the `bulkLoad` method.

### Framework Definition Changes

If you have any custom code that uses `ServiceFrameworkHomes`, you must modify your customizations to now use `ServiceFrameworkXMLHomes`, as the `ServiceFrameworkHomes` class has been removed.

The `ServiceFrameworkXMLHomes` component can now be accessed using the following two ways:

- Using a component reference such as:

  ```
  protected ServiceFrameworkXMLHomes mServiceFrameworkXMLHomes;
  ```

- Using a static call such as:

  ```
  atg.svc.framework.repository.beans.ServiceFrameworkXMLHomes.
  getServiceFrameworkXMLHomes()
  ```

  This method provides an instance of `ServiceFrameworkXMLHomes`.

# ATG Commerce Returns Migration Notes

The current Commerce Service Center components and class names will be inherited in core Commerce to facilitate backward compatibility.

### Java Packages

The following Java packages have been moved from the DCS-CSR module into the DCS module:

- `atg.commerce.csr.returns`
- `atg.commerce.csr.pricing.calculators`
- `atg.reporting.datawarehouse.commerce.csr`
- `atg.commerce.order.edit`

### Nucleus Name Spaces

The following directories have been moved from the DCS-CSR module into the DCS module:

- `/atg/commerce/custsvc`
- `/atg/commerce/custsvc/returns`
- `/atg/commerce/custsvc/returns/processor`
- `/atg/reporting/datawarehouse/process/`
- `/atg/reporting/datawarehouse/process/custsvc/handlers`
- `/atg/reporting/datawarehouse/process/custsvc/`

- /atg/reporting/datawarehouse/loaders/custsvc/

### *File Locations*

The following files were moved from the DCS-CSR module into the CSR module. Unless otherwise noted, all files remain in the same directory structure.

- /atg/commerce/custsvc
  - CsrRepository.properties
  - CsrRepository.xml
- /atg/commerce/custsvc/returns
  - CloneReturnWorkingOrder.properties
  - CreditCardCopier.properties
  - DispositionLookup.properties
  - GetRelatedReturnRequests.properties
  - GiftCertificateCopier.properties
  - InitializePEOProperties.properties
  - InitializeRAOProperties.properties
  - InitializeRCOProperties.properties
  - InitializeReturnWorkingOrderState.properties
  - LostPromotions.properties
  - NonReturnItemDetailsDroplet.properties
  - PaymentGroupCopyManager.properties
  - ReturnDroplet.properties
  - ReturnItemStateDescriptions.properties
  - ReturnItemStates.properties
  - ReturnManager.properties
  - ReturnReasonLookupDroplet.properties
  - ReturnStateDescriptions.properties
  - ReturnStates.properties
  - ReturnTools.properties
  - StoreCreditCopier.properties

  **Note:** The following files remain in the DCS-CSR module:
  - IsBalancingPaymentGroup.properties
  - IsReturnExchange.properties
  - ModifyRefundValuesFormHandler.properties
  - ProfileSearch.properties
  - ReturnFormHandler.properties
  - ReturnLookup.properties
  - ReturnsDataHolder.properties
  - StartReturnExchangeProcess.properties
- /atg/commerce/custsvc/pricing/calculators
  - ExchangeItemAdjustmentCalculator.properties

- ExchangeOrderAdjustmentCalculator.properties
- ExchangeOrderDiscountCalculator.properties
- ExchangePromotionEvaluationUpdateCalculator.properties

## IsReturnExchange Droplet

In the DCS module, the IsReturnExchange droplet has been refactored into the IsReturnActive and GetReturnRequest droplets. The DCS-CSR module continues to use the IsReturnExchange droplet.

## IsItemReturnable Droplet

The IsItemReturnable and OrderIsReturnable droplets have been pushed into an IsReturnable class and droplet in the DCS module.

CSC will continue to use IsItemReturnable and OrderIsReturnable. All implementations call into the ReturnTools isReturnable API. Customizations to the returnable business logic should be made by extending ReturnTools.

The following API in the IsItemReturnable and OrderIsReturnable droplet will be deprecated and no longer called:

```
protected boolean isOrderReturnable(RepositoryItem pOrder)
{
 return getReturnManager().getReturnTools().isReturnable(pOrder);
}
protected boolean isItemReturnable(RepositoryItem pItem)
{
 return getReturnManager().getReturnTools().isItemReturnable(pItem);
}
```

The new API for this is:

```
protected String getOrderReturnableState(RepositoryItem pOrder)
{
 return getReturnManager().getReturnTools().getOrderReturnableState
   (pOrder);
}
protected String getItemReturnableState(RepositoryItem pItem)
{
 return getReturnManager().getReturnTools().getItemReturnableState
   (pItem);
}
```

These IsItemReturnable and isOrderReturnable API changes require that customers who have extended this API change from an IS*x* to the GET*x* methods and use return states.

## ReturnFormHandler

The following handlers have been deprecated and are no longer supported:

- handleStartReturn
- handleStartExchange
- handleSelectReturnItems

## CRSAgentTools

The following changes have been made to the `CRSAgentsTool` API:

| Method | Description |
|---|---|
| `addBypassExpiredPromotionsCheckParameter`<br>`addBypassPaymentGroupAuthorizationParameter` | Deprecated and moved to `OrderTools`. Older versions of this method point to the new location. |
| `addBypassPromotionVetoersParameter`<br>`addItemPriceSourceParameter` | Deprecated and moved to `PricingTools`. Older versions of this method point to the new location. |
| `isOrderFulfilled`<br>`isFulfilledOrderState`<br>`getFulfilledOrderStates`<br>`createPromotionValueMap` | Deprecated and moved to `ReturnTools`. Older versions of this method point to the new location. |
| `getRemainingAmount` | Deprecated and moved to `ClaimableTools`. Older versions of this method point to the new location. |
| `setFulfilledOrderStates` | Deprecated and removed. |

## ReturnTools

The following changes have been made to the `ReturnTools` API:

| Method | Description |
|---|---|
| `isReturnable` | Remove linkage to `CSRAgentTools.isOrderSupported` |
| `repriceCurrentExchangeOrder` | No longer links to `CSRAgentTools`, `ExchangePricingModelHolder` and `EnvironmentTools`. Now calls `priceOrderTotal` directly in `PricingTools`, uses empty promotion Collections instead of the `ExchangePricingModelHolder`, gets the locale from `LocaleTools` and the customer profile from the `ReturnRequest`. For example:<br>`Map params = addExchangeOrderPricingParams(null, pReturnRequest);`<br>`getPricingTools().priceOrderTotal(replorder, Collections.EMPTY_LIST,`<br>`Collections.EMPTY_LIST,`<br>`Collections.EMPTY_LIST,`<br>`Collections.EMPTY_LIST,`<br>`localeTools.getUserLocaleHelper().getLocale(),` |

| Method | Description |
|---|---|
| | pReturnRequest.getProfile(),params); |
| repriceWorkingOrder | No longer links to CRSAgentTools. Now calls SiteContextManager directly. |
| generatePromotionValue AdjustmentMap | No longer links to CSRAgentTools. but to PromotionTools. |
| getPromotionsLost | No longer links to CSRAgentTools but to PromotionTools. |

### Schema Changes

The DDL for the Returns repository is currently defined by the DCS-CSR_ddl.xml file. The CsrRepository specific DDL is now located in /templates/DCS/sql/order_returns_ddl.xml.

The following tables have been removed from the schema:

- csr_exch_repl_item
- csr_exch_repl_itms

The following columns have been removed:

- csr_exch_item table: bonus_refund
- csr_sc_exch_method table: fixed_amount and bonus_credit

# Oracle ATG Web Commerce Reference Store Notes

ATG Commerce Reference Store (CRS) was rewritten for the 10.2 platform. To use the 10.2 version, install and configure a new CRS instance as described in the *ATG Commerce Reference Store Installation and Configuration Guide*.

If an application is based on an earlier version of CRS, follow the migration steps described elsewhere in this guide.

### Integration with Oracle Endeca Commerce

Oracle Endeca Commerce integration with CRS 10.2 requires Oracle Endeca Commerce version 3.1.2.

# Oracle ATG REST Web Services Notes

The following section provides information about migrating from the legacy REST API version and the REST MVC framework that is new in Oracle ATG Web Commerce 10.2. System integrators are encouraged to use the new framework instead of the old one. This can be done piece-meal, moving Legacy REST API calls over as needed to the REST MVC framework.

## JSP Templates

REST MVC uses JSP actors, which differ slightly from the Legacy REST JSP template. Instead of writing out JSON tags, the REST MVC JSP page writes variables to `request` attributes, which are mapped to a `ModelMap`. The `ModelMap` is output to either JSON or XML using a `ResponseGenerator`.

Legacy REST JSP templates can be re-applied using REST MVC JSP actors:

```
<jsp id="displayCart" page="/orderDetail.jsp" context="mobile"
   response-var="orderJSON" >
 <output id="orderOut" name="order" value="${orderJSON}"
   embed-for-mime-type="application/json"/>
</jsp>
```

## REST Security Configuration

The `restSecurityConfiguration.xml` file is deprecated in the REST MVC framework. The `restSecurityConfiguration.xml` file defined Legacy REST resources for `/rest/bean` components, and defined which methods and properties on the component were accessible. However, there is no security for Legacy REST JSP templates.

**Note:** The Legacy REST configuration will continue to use `RestSecurityConfiguration.xml` and `fitleringConfiguration.xml` without modification.

The following is an example of a Legacy REST `restSecurityConfiguration.xml` resource:

```
 <resource component="/atg/store/profile/RegistrationFormHandler" secure="true">
  <method name="handleCheckoutDefaults" secure="false"
    formHandlerErrorURLProperty="updateErrorURL"/>
  <method name="handleCreate" secure="false"/>
  <property name="loginEmailAddress" secure="false"/>
  <property name="dateFormat" secure="false"/>
  <property name="dateOfBirth" secure="false"/>
  <property name="emailOptIn" secure="false"/>
  <property name="previousOptInStatus" secure="false"/>
  <property name="value.email" secure="false"/>
  <property name="value.firstName" secure="false"/>
  <property name="value.lastName" secure="false"/>
  <property name="value.homeAddress.phoneNumber" secure="false"/>
  <property name="value.homeAddress.postalCode" secure="false"/>
  <property name="value.gender" secure="false"/>
  <property name="extractDefaultValuesFromProfile" secure="false"/>
  <property name="createNewUser" secure="false"/>
  <property name="confirmPassword" secure="false"/>
  <property name="value.member" secure="false"/>
  <property name="sourceCode" secure="false"/>
  <property name="value.autoLogin" secure="false"/>
  <property name="value.password" secure="false"/>
 </resource>
```

If you used the `/rest/model/` APIs, you must migrate your changes from the `/atg/rest/security/RestSecurityConfiguration.xml` file to the REST MVC actor definitions. To do this, create a separate Nucleus component and XML file for each API call. Handle methods for form handlers can be placed in a single component and XML file as different variants. Before you can use the actor components, you must register them to make them accessible to the REST MVC framework. Refer to the *ATG Web Services and Integration Framework Guide* for information on registering actor components.

In the actor-based framework, you define an actor XML file. For components and form handlers, you use a component or form handler actor and define which method you want to use per actor. Therefore, only methods that are referenced in an `actor-chain` are accessible.

Instead of defining a list of accessible input properties, you define `input` elements for each actor so only inputs defined in the actor definition will be observed. Any extra parameters that are passed will be ignored.

The following is an example of what an actor does in the REST MVC framework:

```
<actor-template>
 <actor-chain id="create">
  <form name="/atg/store/profile/RegistrationFormHandler"
    handle="create">
   <input name="loginEmailAddress" value="${param.loginEmailAddress}"/>
   <input name="dateFormat" value="${param.dateFormat}"/>
   <input name="dateOfBirth" value="${param.dateOfBirth}"/>
   <input name="emailOptIn" value="${param.emailOptIn}"/>
   <input name="previousOptInStatus"
      value="${param.previousOptInStatus}"/>
   <input name="value.email" value="${param.email}"/>
   <input name="value.firstName" value="${param.firstName}"/>
   <input name="value.lastName" value="${param.lastName}"/>
   <input name="value.homeAddress.phoneNumber"
      value="${param.phoneNumber}"/>
   <input name="value.homeAddress.postalCode"
      value="${param.postalCode}"/>
   <input name="value.gender" value="${param.gender}"/>
   <input name="extractDefaultValuesFromProfile"
      value="${param.extractDefaultValuesFromProfile}"/>
   <input name="createNewUser" value="${param.createNewUser}"/>
   <input name="confirmPassword" value="${param.confirmPassword}"/>
   <input name="value.member" value="${param.member}"/>
   <input name="sourceCode" value="${param.sourceCode}"/>
   <input name="value.autoLogin" value="${param.autoLogin}"/>
   <input name="value.password" value="${param.password}"/>
   <input name="createErrorURL" value="${param.errorURL}"/>
   <input name="createSuccessURL" value="${param.successURL}"/>
  </form>
 </actor-chain>

 <actor-chain id="checkoutDefaults" transaction="TX_SUPPORTS">
  <form name="/atg/store/profile/RegistrationFormHandler"
    handle="checkoutDefaults">
```

```
    <input name="value.shippingAddress"
      value="${param.shippingAddress}"/>
    <input name="value.defaultCreditCard"
      value="${param.defaultCreditCard}"/>
    <input name="value.defaultShippingMethod"
      value="${param.defaultShippingMethod}"/>
    <input name="updateErrorURL" value="${param.errorURL}"/>
    <input name="updateSuccessURL" value="${param.successURL}"/>
   </form>
  </actor-chain>
<actor-template>
```

All actors are securable – whether they are JSP fragments, droplets, components, form handlers, etc. The inputs for all actors are explicitly declared so it is impossible to pass in a parameter that is not expected. Furthermore, each method defines its expected inputs so that two different handle methods on a form handler may define only the inputs that they actually use.

### REST URLs

The REST MVC APIs start with `/rest/model` instead of `/rest/bean`, `/rest/repository` or `/rest/service`. Any calls to `/rest/bean`, `/rest/repository` or `/rest/service` will still be resolved by the Legacy REST `BeanProcessor`, `RepositoryProcessor` or `ServiceProcessor` respectively.

### Filtering

When using the REST MVC framework, you must also migrate your `/atg/rest/filtering/filteringConfiguration.xml` file to a new `/atg/dynamo/service/filter/bean/beanFilteringConfiguration.xml` file.

# Oracle ATG REST Web Services Migration Examples

The following are examples of existing Mobile Commerce `/rest/bean`, `/rest/repository` and `/rest/service` calls and what they would look like in the REST MVC framework.

### Migrating a REST Service to a Droplet Actor

The following is an example of a `/rest/service` migration to a `DropletActor`. This example takes a Legacy REST service that displays the billing address for an order and shows how you would create a corresponding `DropletActor` in the REST MVC API.

#### *Billing Address REST Service Example*

The Legacy REST service call contains the following:

- The `/rest/service` API call to migrate:

  ```
  curl -v -b cookies.txt -X GET
  \ http://myserver:8080/rest/service/atg/commerce/billingAddresses
  ```

- The JSP template, which renders the page that displays the data:

```
<%--
 Return a list of permitted shipping addresses for the current order
--%>
<dsp:page>
 <dsp:importbean
   bean="/atg/commerce/order/purchase/ShippingGroupDroplet"/>
 <%-Initialize shipping data objects --%>
 <dsp:droplet name="ShippingGroupDroplet">
  <dsp:param name="createOneInfoPerUnit" value="false"/>
  <dsp:param name="clearShippingInfos" value="true"/>
  <dsp:param name="clearShippingGroups" value="true"/>
  <dsp:param name="shippingGroupTypes" value="hardgoodShippingGroup"/>
  <dsp:param name="initShippingGroups" value="true"/>
  <dsp:param name="initBasedOnOrder" value="true"/>
  <dsp:oparam name="output"/>
 </dsp:droplet>
 <dsp:importbean
   bean="/atg/commerce/order/purchase/ShippingGroupContainerService" />
 <dsp:importbean bean="/atg/store/droplet/AvailableShippingAddresses" />
 <dsp:importbean bean="/atg/userprofiling/Profile" />
 <json:object>
  <dsp:getvalueof var="shippingGroupMap" vartype="java.lang.Object"
   bean="ShippingGroupContainerService.shippingGroupMap" />
  <c:if test="${not empty shippingGroupMap}">
   <dsp:droplet name="AvailableBillingAddresses">
    <dsp:param name="map" value="${shippingGroupMap}" />
    <dsp:param name="defaultId"
      bean="Profile.shippingAddress.repositoryId" />
    <dsp:param name="sortByKeys" value="true" />
    <dsp:oparam name="output">
     <dsp:getvalueof var="permittedAddresses"
       vartype="java.lang.Object"
       param="permittedAddresses" />
     <json:array name="shippingAddresses"
       items="${permittedAddresses}" var="hsg">
      <json:object>
       <json:property name="nickname" value="${hsg.key}" />
       <dsp:include page="addressProperties.jsp">
        <dsp:param name="address"
          value="${hsg.value.shippingAddress}" />
        <dsp:param name="useCountryCode" value="${true}"/>
       </dsp:include>
      </json:object>
     </json:array>
    </dsp:oparam>
   </dsp:droplet>
  </c:if>
 </json:object>
</dsp:page>
```

- The `filteringConfiguration.xml` file, which filters the results displayed to the end user:

```
<bean-filtering>
 <component name="atg.core.util.Address">
  <property name="address1"/>
  <property name="address2"/>
  <property name="address3"/>
  <property name="city"/>
  <property name="country" property-customizer=
    "/atg/rest/LocalizedCountryDisplayNamePropertyCustomizer" />
  <property name="firstName"/>
  <property name="lastName"/>
  <property name="middleName"/>
  <prpoerty name="nickname"/>
  <property name="phoneNumber"/>
  <property name="postalCode"/>
  <property name="state"/>
 </component>
</bean-filtering>
```

### *Billing Address REST Model Example*

The REST MVC framework model call that uses the actor-based API contains the following:

- The `/rest/model` API call:

```
curl -v -b cookies.txt -X GET
\ http://myserver:8080/rest/model/atg/commerce/billingAddressesActor
```

- The `DropletActor` that invokes the droplets used to send the output to a `ModelMap`:

```
/atg/commerce/billingAddressesActor.properties
$class=atg.service.actor.ActorChainService
$scope=global
definitionFile=/atg/commerce/billingAddressesActor.xml
<!-/atg/commerce/billingAddressesActor.xml -->
<actor-chain>
 <droplet name="/atg/commerce/order/purchase/ShippingGroupDroplet">
  <input name="createOneInfoPerUnit" value="false"/>
  <input name="clearShippingInfos" value="true"/>
  <input name="clearShippingGroups" value="true"/>
  <input name="shippingGroupTypes" value="hardgoodShippingGroup"/>
  <input name="initShippingGroups" value="true"/>
  <input name="initBasedOnOrder" value="true"/>
 <droplet>
 <droplet name="/atg/droplet/Switch">
  <input name="value" value="${availableBillingAddresses not empty}"/>
  <oparam name="true">
   <droplet name="/atg/store/droplet/AvailableBillingAddresses"
     var="availableBillingAddresses">
    <input name="map"
      value="${nucleus['/atg/commerce/order/purchase/
```

```
      ShippingGroupContainerService'].shippingGroupMap}" />
    <input name="defaultId"
      value="${nucleus['/atg/userprofiling/Profile.shippingAddress']
      .repositoryId}" />
    <input name="sortByKeys" value="true" />
    <!-output droplet permittedAddresses to model as
      billingAddresses -->
    <oparam name="output">
     <output name="billingAddresses"
       value="${availableBillingAddresses.permittedAddresses}"/>
    </oparam>
   </droplet>
  </oparam>
 </droplet>
</actor-chain>
```

- The beanFilterConfiguration.xml file, which filters the results displayed to the end user:

```
<bean-filtering>
 <bean type="atg.core.util.Address">
  <filter id="default">
   <property name="address1"/>
   <property name="address2"/>
   <property name="address3"/>
   <property name="city"/>
   <property name="country"
     property-customizer="/atg/dynamo/service/actor/
     LocalizedCountryDisplayNamePropertyCustomizer" />
   <property name="firstName"/>
   <property name="lastName"/>
   <property name="middleName"/>
   <property name="nickname"/>
   <property name="phoneNumber"/>
   <property name="postalCode"/>
   <property name="state"/>
  </filter>
 </bean>
</bean-filtering>
```

## Migrating a Legacy REST Service to a JSP Actor

The following is an example of a /rest/service migration to a JSPActor. This example takes a Legacy REST service that returns available shipping methods and cost based upon the location of the customer and shows how you would create a corresponding JSPActor in the REST MVC API.

### *Shipping Method Legacy REST Service Example*

The Legacy REST service call contains the following:

- The /rest/service API call to migrate:

```
curl -v -b cookies.txt -X GET
 \ http://myserver:8080/rest/service/atg/commerce/pricing/
 AvailableShippingMethods?getPrices=true
```

- The JSP template, which renders the page that displays the data:

```
<%--
 Return available shipping methods, optionally with prices
 -
 Optional parameters:
  getPrices if true, will return prices with the shipping methods
 --%>
<dsp:page>
 <dsp:importbean bean="/atg/commerce/pricing/AvailableShippingMethods" />
 <dsp:importbean bean="/atg/commerce/pricing/CurrencyCodeDroplet"/>
 <dsp:importbean
   bean="/atg/commerce/order/purchase/ShippingGroupFormHandler" />
 <dsp:importbean bean="/atg/commerce/ShoppingCart" />
 <dsp:importbean bean="/atg/userprofiling/Profile"/>
 <dsp:getvalueof var="shippingGroup"
   bean="ShippingGroupFormHandler.
   firstNonGiftHardgoodShippingGroupWithRels" />
 <dsp:getvalueof var="getPrices" param="getPrices" />
 <json:object>
  <json:array name="shippingMethods">
   <%--
    Iterates over the list of available shipping methods and determine
      default one.
    Input parameters:
     shippingGroup shipping group. We pass the first one from the
             list of shipping groups in the current order
             (shopping cart). We always have at least one
             hardgood shipping group in the cart.
    Output parameters:
     availableShippingMethods a list of shipping method codes
   --%>
   <dsp:droplet name="AvailableShippingMethods">
    <dsp:param name="shippingGroup" value="${shippingGroup}" />
    <dsp:oparam name="output">
     <dsp:getvalueof var="availableShippingMethods"
        vartype="java.lang.Object"
      param="availableShippingMethods" />
     <c:forEach var="availableShippingMethod"
       items="${availableShippingMethods}">
      <json:object>
       <json:property name="name"
         value="${availableShippingMethod}" />
       <c:if test="${getPrices == 'true'}">
    <%-Determine shipping price for the current shipping method --%>
        <dsp:droplet
          name="/atg/store/pricing/PriceShippingMethod">
```

```
            <dsp:param name="shippingMethod"
               value="${availableShippingMethod}" />
            <dsp:oparam name="output">
             <dsp:getvalueof var="shippingPrice"
               param="shippingPrice" />
            </dsp:oparam>
           </dsp:droplet>
           <json:property name="price" value="${shippingPrice}" />
          </c:if>
         </json:object>
        </c:forEach>
       </dsp:oparam>
      </dsp:droplet>
     </json:array>
     <%-Add the currencyCode if we're returning prices --%>
     <c:if test="${getPrices == 'true'}">
      <dsp:getvalueof var="priceListLocale" vartype="java.lang.String"
       bean="Profile.priceList.locale" />
      <%--
        This droplet calculates a currency code for the locale specified.
        Input parameters:
         locale Specifies a locale to calculate the code from.
        Output parameters:
         currencyCode The resulting currency code.
        Open parameters:
          output Always rendered.
      --%>
      <dsp:droplet name="CurrencyCodeDroplet">
       <dsp:param name="locale" value="${priceListLocale}" />
       <dsp:oparam name="output">
        <dsp:getvalueof var="currencyCode" vartype="java.lang.String"
          param="currencyCode" />
        <json:property name="currencyCode" value="${currencyCode}" />
       </dsp:oparam>
      </dsp:droplet>
     </c:if>
    </json:object>
   </dsp:page>
```

This example does not use a `filteringConfiguration.xml` file.

### *Shipping Method REST Method Example*

The REST MVC framework model call contains the following:

- The `/rest/model` API call:

  ```
  curl -v -b cookies.txt -X GET \
  http://myserver:8080/rest/model/atg/commerce/pricing/
  AvailableShippingMethodsActor?getPrices=true
  ```

- The `JSPActor` that will invoke the JSP template:

```
/atg/commerce/pricing/AvailableShippingMethodsActor.properties
$class=atg.service.actor.ActorChainService
$scope=global
definitionFile=/atg/commerce/availableShippingMethodsActor.xml
<!-/atg/commerce/availableShippingMethodsActor.xml -->
<actor-chain id="shippingMethods">
 <jsp url="/atg/commerce/pricing/shippingMethodsActor.jsp" context="DCS">
 <!-if true, getPrices will return prices with the shipping methods -->
  <input name="getPrices" value="${param.getPrices}">
  <output name="shippingMethods" value="${shippingMethods}"/>
  <output name="currencyCode" value="${currencyCode}"/>
 </jsp>
</actor-chain>
```

- The JSP template that will execute the JSP page and output objects to the `ModelMap`. Note that the JSP page will not write directly to the HTPP response. The output of the JSP page is available in the `response-var` attribute that allows you to map to a property in the `ModelMap`.

```
<%--
 Return available shipping methods, optionally with prices
 -
 Optional parameters:
  getPrices if true, will return prices with the shipping methods
 -
 Model output:
  shippingMethods
  currencyCode
 --%>
<dsp:page>
 <dsp:importbean bean="/atg/commerce/pricing/AvailableShippingMethods" />
 <dsp:importbean bean="/atg/commerce/pricing/CurrencyCodeDroplet"/>
 <dsp:importbean
   bean="/atg/commerce/order/purchase/ShippingGroupFormHandler" />
 <dsp:importbean bean="/atg/commerce/ShoppingCart" />
 <dsp:importbean bean="/atg/userprofiling/Profile"/>
 <dsp:getvalueof var="shippingMethodsMap" class="java.core.util.HashMap"
   scope="request"/>
 <dsp:getvalueof var="shippingGroup"
   bean="ShippingGroupFormHandler.
   firstNonGiftHardgoodShippingGroupWithRels" />
 <dsp:getvalueof var="getPrices" param="${getPrices}" />
 <dsp:droplet name="AvailableShippingMethods">
  <dsp:param name="shippingGroup" value="${shippingGroup}" />
  <dsp:oparam name="output">
   <dsp:getvalueof var="availableShippingMethods"
     vartype="java.lang.Object" param="availableShippingMethods" />
   <c:forEach var="availableShippingMethod"
     items="${availableShippingMethods}">
    <c:set var="shippingPrice" value="null"/>
    <c:if test="${getPrices == 'true'}">
```

```
    <%-Determine shipping price for the current shipping method --%>
     <dsp:droplet name="/atg/store/pricing/PriceShippingMethod"
       var="priceShippingMethod">
      <dsp:param name="shippingMethod"
        value="${availableShippingMethod}" />
      <dsp:oparam name="output">
       <c:set target="${shippingMethodsMap}
         property="${availableShippingMethod}"
         value="${priceShippingMethod.shippingPrice}"/>
      </dsp:oparam>
     </dsp:droplet>
    </c:if>
   </c:forEach>
  </dsp:oparam>
  <!-Sets property in the ActorContext -->
  <c:set name="shippingMethods" value="${shippingMethodsMap}"
    scope="request"/>
 </dsp:droplet>
 <%-Add the currencyCode if we're returning prices --%>
 <c:if test="${getPrices == 'true'}">
  <dsp:getvalueof var="priceListLocale" vartype="java.lang.String"
   bean="Profile.priceList.locale" />
  <dsp:droplet name="CurrencyCodeDroplet" var="currencyCodeDroplet">
   <dsp:param name="locale" value="${priceListLocale}" />
   <dsp:oparam name="output">
    <!-Sets property in the ActorContext -->
    <c:set name="currencyCode"
      value="${currencyCodeDroplet.currencyCode}" scope="request"/>
   </dsp:oparam>
  </dsp:droplet>
 </c:if>
</dsp:page>
```

- The `modelFilterConfiguration.xml` file, which filters the results displayed to the end user:

## Migrating a Legacy REST Service to an Actor Referencing Nucleus Components

The following is an example of a `/rest/service` migration to an actor that contains the Nucleus component `/atg/commerce/ShoppingCart`.

### *Shopping Cart Legacy REST Service Example*

The Legacy REST service call contains the following:

- The `/rest/service` call:

```
curl -v -b cookies.txt -X GET \
 http://myserver:8080/rest/service/atg/commerce/ShoppingCart
```

- The `restSecurityConfiguration.xml` file:

```
<resource component="/atg/commerce/ShoppingCart" secure="false">
</resource>
```

- The `filteringConfiguration.xml` file:

```
<bean-filtering>
 <!-Shopping cart filters -->
 <!-reprice the cart with ORDER_TOTAL operation before returning it
   this is the default and used for retrieving order confirmation data
   regular cart retrievals should price with ORDER_SUBTOTAL, using the
   cartRepriceSubtotal template below -->
 <component name="/atg/commerce/ShoppingCart">
  <property name="cart"
    property-customizer="/atg/rest/filtering/customizers/
    CartTotalRepricer"/>
 </component>
 <component name="/atg/commerce/ShoppingCart/totalCommerceItemCount">
   <property name="totalCommerceItemCount"
     target="current.totalCommerceItemCount"
     component="/atg/commerce/ShoppingCart"/>
 </component>
 <!-applied to the shopping cart -->
 <component name="atg.projects.store.order.StoreOrderImpl">
  <property name="shipping" target="priceInfo.shipping"/>
  <property name="tax" target="priceInfo.tax"/>
  <property name="discount" target="priceInfo.discountAmount"/>
  <property name="storeCredit"
     property-customizer="/atg/rest/filtering/customizers/
     StoreCreditPropertyCustomizer"/>
  <property name="subtotal" target="priceInfo.rawSubtotal"/>
  <property name="currencyCode" target="priceInfo.currencyCode"/>
  <property name="total" target="priceInfo.amount"/>
  <property name="appliedPromotions" target="priceInfo.adjustments"
    property-customizer="/atg/rest/filtering/customizers/
    RemoveNullPromotions"/>
  <property name="commerceItems"/>
  <property name="totalCommerceItemCount"/>
  <property name="containsGiftWrap"/>
  <property name="shippingGroupCount"/>
  <property name="couponCode" target="currentCouponCode"
    component="/atg/store/order/purchase/CouponFormHandler"
    writable="false"/>
 </component>
 <component name="atg.commerce.order.CommerceItem">
  <property name="commerceItemId" target="id"/>
  <property name="prodId" target="auxiliaryData.productId"/>
  <property name="thumbnailImage"
    target="auxiliaryData.productRef.thumbnailImage.url"/>
  <property name="sku" target="auxiliaryData.catalogRef"/>
  <property name="qty" target="quantity"/>
  <property name="price" target="priceInfo.amount"/>
```

```
 <property name="listPrice" target="priceInfo.listPrice"/>
 <property name="salePrice" target="priceInfo.salePrice"/>
 <property name="onSale" target="priceInfo.onSale"/>
 <property name="unitPrices"
   property-customizer="/atg/rest/filtering/customizers/
   UnitPriceDetail"/>
 <property name="appliedPromotions" target="priceInfo.adjustments"
   property-customizer="/atg/rest/filtering/customizers/
   RemoveNullPromotions"/>
</component>
<component name="atg.commerce.pricing.PricingAdjustment">
 <property name="promotion" target="pricingModel.displayName"/>
</component>
<component name="atg.commerce.pricing.UnitPriceBean">
 <property name="unitPrice"/>
 <property name="quantity"/>
</component>
</bean-filtering>
```

### *Shopping Cart REST Method Example*

The REST MVC model call contains the following:

- The `/rest/model` API call

  ```
  curl -v -b cookies.txt -X GET \
   http://myserver:8080/rest/model/atg/commerce/ShoppingCartActor
  ```

- The actor that references the `/atg/commerce/shoppingCart` Nucleus component:

  ```
  /atg/commerce/modelproduer/ShoppingCartActor.properties
  $class=atg.service.actor.ActorChainService
  $scope=global
  definitionFile=/atg/commerce/shoppingCartActor.xml
  <actor-template>
   <!-/atg/commerce/shoppingCartActor.xml -->
   <actor-chain id="ShoppingCart" transaction="TX_SUPPORTS">
    <!-output the shopping cart component as property 'cart' -->
    <component name="/atg/commerce/ShoppingCart" component-var="cart">
     <output name="cart" value="${cart}"/>
    </component>
   </actor-chain>
  </actor-template>
  ```

- The `beanFilteringConfiguration.xml` that filters the results:

  ```
  <bean-filtering>
   <!-Shopping cart filters -->
   <!-reprice the cart with ORDER_TOTAL operation before returning it
     this is the default and used for retrieving order confirmation data
     regular cart retrievals should price with ORDER_SUBTOTAL, using the
     cartRepriceSubtotal template below -->
   <bean type="/atg/commerce/ShoppingCart">
    <filter id="default">
  ```

```
  <property name="cart" property-customizer="/atg/rest/filtering/
    customizers/CartTotalRepricer"/>
 </filter>
</bean>
<bean type="/atg/commerce/ShoppingCart/totalCommerceItemCount">
 <filter id="default">
  <property name="totalCommerceItemCount"
    target="current.totalCommerceItemCount"
    component="/atg/commerce/ShoppingCart"/>
 </filter>
</bean>
<!-applied to the shopping cart -->
<bean type="atg.projects.store.order.StoreOrderImpl">
 <filter id="default">
  <property name="shipping" target="priceInfo.shipping"/>
  <property name="tax" target="priceInfo.tax"/>
  <property name="discount" target="priceInfo.discountAmount"/>
  <property name="storeCredit" property-customizer="/atg/rest/
    filtering/customizers/StoreCreditPropertyCustomizer"/>
  <property name="subtotal" target="priceInfo.rawSubtotal"/>
  <property name="currencyCode" target="priceInfo.currencyCode"/>
  <property name="total" target="priceInfo.amount"/>
  <property name="appliedPromotions" target="priceInfo.adjustments"
    property-customizer="/atg/rest/filtering/customizers/
    RemoveNullPromotions"/>
  <property name="commerceItems"/>
  <property name="totalCommerceItemCount"/>
  <property name="containsGiftWrap"/>
  <property name="shippingGroupCount"/>
  <property name="couponCode" target="currentCouponCode"
    component="/atg/store/order/purchase/CouponFormHandler"/>
 </filter>
</bean>
<bean type="atg.commerce.order.CommerceItem">
 <filter id="default">
  <property name="commerceItemId" target="id"/>
  <property name="prodId" target="auxiliaryData.productId"/>
  <property name="thumbnailImage"
     target="auxiliaryData.productRef.thumbnailImage.url"/>
  <property name="sku" target="auxiliaryData.catalogRef"/>
  <property name="qty" target="quantity"/>
  <property name="price" target="priceInfo.amount"/>
  <property name="listPrice" target="priceInfo.listPrice"/>
  <property name="salePrice" target="priceInfo.salePrice"/>
  <property name="onSale" target="priceInfo.onSale"/>
  <property name="unitPrices" property-customizer="/atg/rest/
    filtering/customizers/UnitPriceDetail"/>
  <property name="appliedPromotions" target="priceInfo.adjustments"
    property-customizer="/atg/rest/filtering/customizers/
    RemoveNullPromotions"/>
```

```
  </filter>
 </bean>
 <bean type="atg.commerce.pricing.PricingAdjustment">
  <filter>
   <property name="promotion" target="pricingModel.displayName"/>
  </filter>
 </bean>
 <bean type="atg.commerce.pricing.UnitPriceBean">
  <filter id="default">
   <property name="unitPrice"/>
   <property name="quantity"/>
  </filter>
 </bean>
</bean-filtering>
```