ORACLE®

ATG WEB COMMERCE

Commerce Service Center

Version 10.2

Installation and Programming Guide

Oracle ATG
One Main Street
Cambridge, MA 02142
USA

ATG Commerce Service Center Installation and Programming Guide

Product version: 10.2
Release date: 04-30-13
Document identifier: CSCInstallationAndProgrammingGuide1403311801

Table of Contents

1 Introduction

Oracle ATG Web Commerce Service Center is a customizable and deployable customer service application that enables an agent to perform the following tasks for a Commerce site:

- Create and manage customer profiles
- Create and manage orders
- Issue refunds and exchanges
- Process returned items
- Research customer activity

For information about Oracle ATG Web Commerce, see the *ATG Commerce Guide to Setting Up a Store* and the *ATG Commerce Programming Guide*.

Audience

This manual is intended for System Administrators, Site Administrators and Programmers responsible for installing, configuring and customizing Commerce Service Center.

Documentation Conventions

The following conventions are used in this manual:

- Installation Directory

  `<ATG10dir>` — The installation directory for ATG 10. For example, the default location for installations is `/ATG/ATG10.2`

- Menu Navigation

  The “ > “ (greater than) symbol indicates menu choices. For example, File > Save means you should select the Save option on the File menu.

Related Documents

The following documentation provides additional reference information:

Document	Description
ATG Commerce Service Center User Guide	Describes Commerce Service Center concepts and tasks for agents and end users.
ATG Service Center UI Programming Guide	Describes the architecture and customization of the Service Center UI.
ATG Ticketing User Guide	Describes the architecture and implementation of ATG Ticketing.
ATG API Reference for Commerce Service Center	Contains information from the Commerce Service Center API
ATG Installation and Configuration Guide	Describes how to install and configure ATG applications running on different Web applications.
ATG Personalization Programming Guide	Describes programming tasks for the ATG Personalization and Scenarios module. Includes information on setting up profile repositories, creating targeting rules and services, configuring scenario servers, and adding custom scenario events and actions.
ATG Commerce Guide to Setting Up a Store	Describes how to use Oracle ATG Web Commerce to create an online store. Intended for business users and page developers.
ATG Commerce Programming Guide	Describes how to install and customize Oracle ATG Web Commerce. Intended for programmers and site administrators.
ATG Business Control Center User's Guide	Describes how to use Oracle ATG Web Commerce Business Control Center's Web interface to manage user profiles and organizations; create and assign roles; segment site visitors; and define rules for personalizing site content. Also includes setup and configuration information. Intended for all audiences.
ATG Repository Guide	Describes the ATG platform's repository API. Presents programming concepts for advanced users, including SQL repositories, LDAP repositories, secured repositories, and composite repositories. Includes examples and reference information to help programmers develop applications using the repository API.

Before You Begin

This section provides a high-level description of the tasks you need to perform before running Commerce Service Center:

1. Ensure that supported versions of your Web application software is installed.
2. Install a supported application server. Consult the *ATG Installation and Configuration Guide* for your application server.
3. Install ATG 10.2.
4. Install and configure Oracle ATG Web Commerce 10.2
5. Install a supported database.

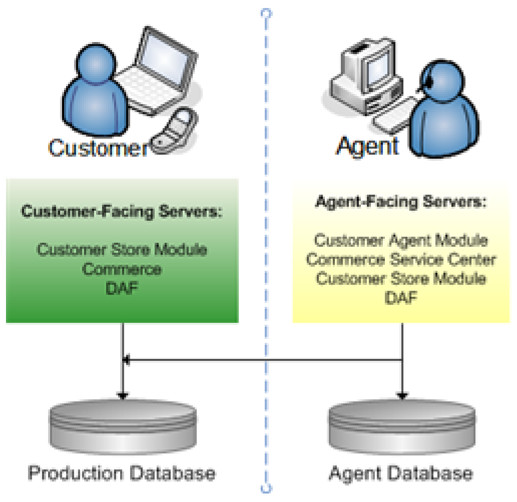
Browser and Environment Requirements

For information about the supported browsers, environments, and configurations, refer to the Oracle ATG Web Commerce Supported Environments Matrix document in the My Oracle Support knowledge base.

Users must enable cookies and scripting in their browsers to access Commerce Service Center.

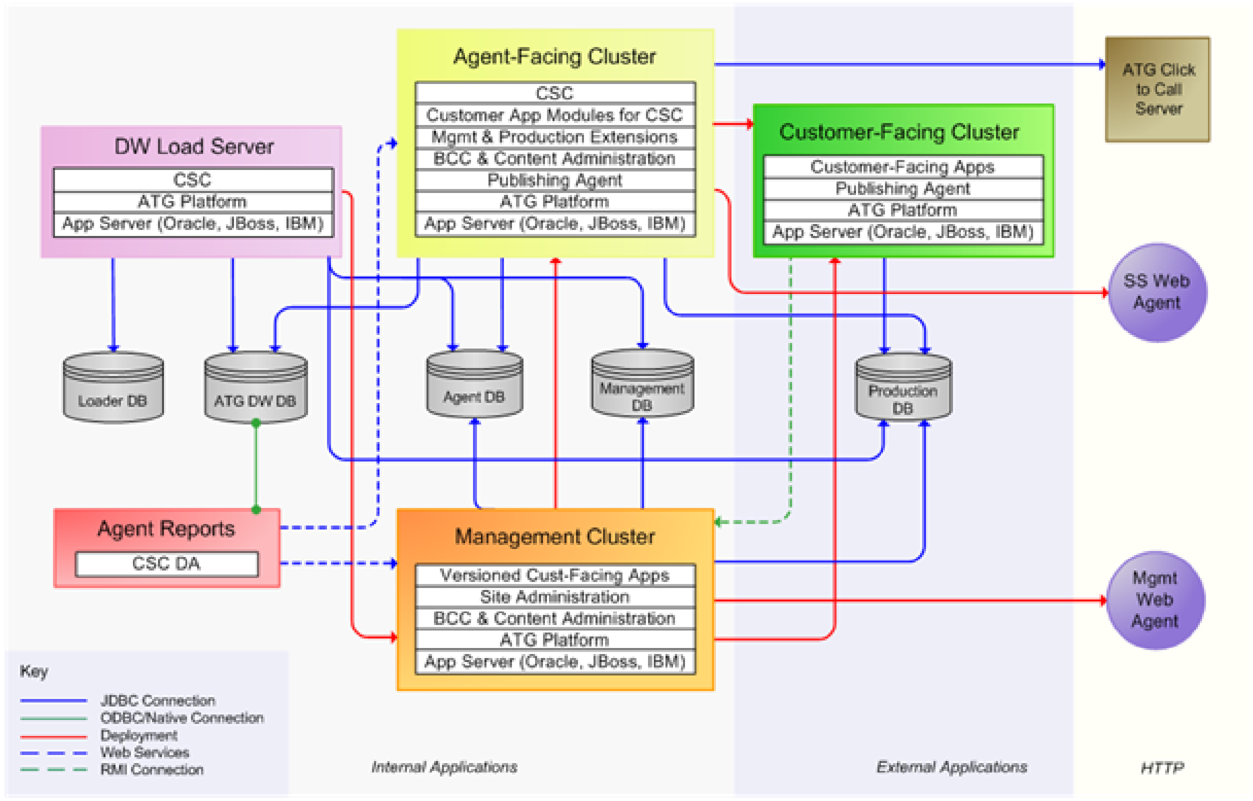
2 Commerce Service Center Server Architecture

Commerce Service Center is comprised of both customer-facing and agent-facing clusters. The customer-facing cluster contains such applications as Oracle ATG Web Commerce stores and other components that customers use. The agent-facing cluster contains application such as Commerce Service Center and other components that agents use.



Commerce Service Center Environment

Commerce Service Center works in tandem with Commerce and other ATG applications. The following diagram displays a complete architectural schematic of Commerce Service Center within a typical environment. Note that your environment may or may not look similar based upon the applications that you have installed.

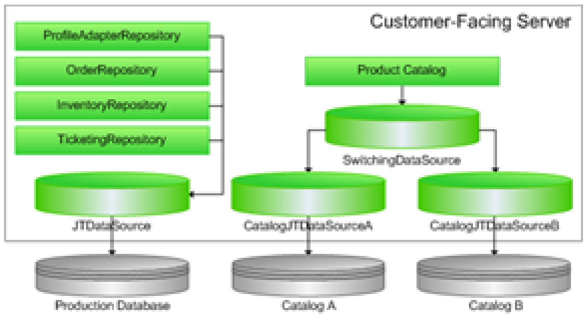


Commerce Service Center Architecture

Customer-Facing Server Configuration

Customer-facing servers display the sites that customers log into. These servers contain versioned information that is deployed using standard Commerce Publishing. Repository information is accessed through data sources that connect to production or switching databases.

In the customer-facing server configuration, the `JTDataSource` component is used for all repositories other than the product catalog.



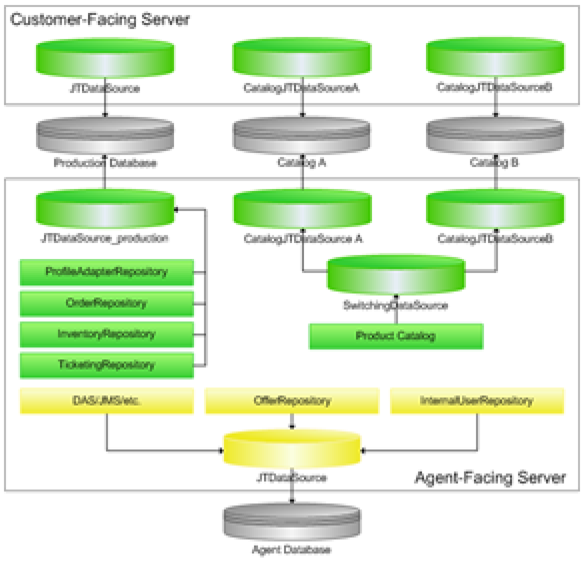
Customer-Facing Server Configuration

Agent-Facing Server Configuration

The agent-facing server has access to the same customer-facing information, with the addition of agent-specific repositories.

With agent-facing server configurations the `JTDataSource` references its own schema rather than the one used by the customer-facing server. In addition to the `JTDataSource`, agent-facing servers use a `JTDataSource_production` data source. This `JTDataSource_production` data source references all operational data, which includes profiles, orders, tickets, and inventory data.

The catalog may still continue to uses the `ProductCatalogSwitchingDataSource` component.



Customer and Agent-Facing Server Configuration

For additional information on Commerce architecture, refer to the *ATG Commerce Programming Guide*.

Note: The features available with Commerce's B2B module will not function correctly within Commerce Service Center.

3 Installing and Configuring the Commerce Service Center Server

This section discusses the installation of Commerce Service Center. It is strongly suggested that you install Commerce Service Center using the Configuration and Installation Manager (CIM).

Requirements for Commerce Service Center

Before you proceed, you must have the following installed or configured:

- Java JDK 1.6.0_25 or the latest supported version. Refer to the Oracle ATG Web Commerce Supported Environments Matrix document in the My Oracle Support knowledge base. Ensure that the `JAVA_HOME` and `JAVA_HOME/bin` variables are set in your path correctly
- A supported application server, such as Oracle WebLogic. Consult the *ATG Installation and Configuration Guide* for your application server
- A supported database, such as Oracle. Create two database users for your database to use, for example *admin* and *svcagent* users
- The Oracle ATG Web Commerce Platform 10.2
- Oracle ATG Web Commerce 10.2 or Oracle ATG Web Commerce Reference Store 10.2

  Note: Commerce Service Center does not enable the Commerce B2B features. Any orders that contain cost center, purchase order, invoice requests, order approvals or organization information will be read-only and cannot be modified by an agent.

Database and Schema Requirements

Commerce Service Center requires two different database user accounts for database configuration. Create the following accounts before configuring the database:

Database Schema	Contents
`svcagent` (Agent)	Versioned repository tables. Referred to in this document as the agent schema. This schema is accessed from the agent-facing servers.

Database Schema	Contents
svcadmin (Production)	Unversioned repository tables. Referred to in this document as the production schema. This schema is accessed from the customer-facing servers.

Databases can be configured with their tables on separate machines, separate table spaces or partitions, or in other configurations.

The following table outlines the template files used by the CIM installation process, the SQL files that are called from that template, and the schema to which the SQL files will be pointed:

Template	SQL Files	Schema
`production-ddl-template`	`DCS-CSR_ddl.sql`, `DCS-CSR_ticketing_ddl.sql` `unversioned_DCS-CSR_site_ddl.sql`	Production
`agent-ddl-template`	`DCS-CSR_logging_ddl.sql`, `DCS-CSR_profile_ddl.sql`, `DCS-CSR_approvals_ddl.sql`	Agent
`management-ddl-template`	`versioned_DCS-CSR_site_ddl.sql`	Management

Installing with the Configuration and Installation Manager

CIM simplifies product configuration by providing scripts that configure Commerce Service Center. The scripts allow you to identify the components used within your environment, as well as to add on additional applications. Using CIM ensures that all necessary steps are completed and are performed in the correct order.

Note: It is best to install Commerce Service Center using CIM.

CIM handles the following configuration steps:

- Creates data sources according to the database connection information you supplied, including those needed for applications you may add
- Creates database tables and imports initial data
- Creates and configures ATG application servers, including a dedicated indexing server, a lock manager and required loader servers
- Assembles your application EAR files for each ATG server, including modules for the Agent, Production and Data Warehouse load servers, as well as DCS-CSR, Fulfillment and UI modules
- Deploys EAR files to your application server and allows you to start up the agent-facing, customer-facing and load servers

- Allows you to add custom modules

Refer to the CIM script help and the *ATG Installation and Configuration Guide* for additional information on CIM.

To install Commerce Service Center using CIM, do the following:

1. Install your application server.
2. Install your application files.
3. To start CIM, go to `<ATG10dir>/home/bin` and launch the CIM script:

   ```
   ./cim.sh | bat
   ```

4. Select the products you want to install.
5. Select the add-ons that you want to install.
6. Follow the CIM script according to the prompts. You can type H at any prompt for additional information.

For detailed information on Commerce Service Center CIM installations, refer to the Appendix C, *CIM Configuration Components* (page 173)

Repositories

Repositories are configured using the CIM Database Configuration menu. The Commerce Service Center database tables are used by several repositories. Depending on your needs, you may need to change the configuration of these repositories, such as their data sources. For information on configuring repositories, refer to the *ATG Repository Guide*.

The following repositories are shared between your customer-facing server and Commerce Service Center:

- `/atg/commerce/catalog/ProductCatalog`
- `/atg/commerce/claimable/ClaimableRepository`
- `/atg/commerce/contracts/Contracts`
- `/atg/commerce/custsvc/CsrRepository`
- `/atg/commerce/custsvc/approvals/ApprovalsRepository`
- `/atg/commerce/gifts/Giftlists`
- `/atg/commerce/inventory/InventoryRepository`
- `/atg/commerce/locations/LocationRepository`
- `/atg/commerce/order/OrderRepository`
- `/atg/commerce/pricing/priceLists/PriceLists`
- `/atg/userprofiling/PersonalizationRepository`

CIM configures all repositories to use the `/atg/dynamo/service/jdbc/JTDataSource_production` data source, which should reference the production schema.

For each repository, CIM configures the following properties by default. If you create new repositories, you should ensure that these properties are configured:

Property	Suggested Value
`dataSource`	`/atg/dynamo/service/jdbc/JTDataSource_production`
`idGenerator`	`/atg/dynamo/service/IdGenerator_production`
`lockManager`	`/atg/dynamo/service/ClientLockManager_production`
`eventServer`	`/atg/dynamo/server/SQLRepositoryEventServer_production`
`subscriberRepository`	`/atg/dynamo/service/jdbc/SQLRepository_production`

Using IDGenerators

CIM ensures that all repositories share the same IDGenerator component. By default, any repository defined in the customer-facing cluster should use the `/atg/dynamo/service/IDGenerator_production` IDGenerator component on the agent-facing server.

For detailed information on the IDGenerator component, refer to the *ATG Platform Programming Guide*.

Understanding Lock Management

CIM allows you to configure dedicated lock manager servers during the installation process.

Lock servers synchronize caches among ATG servers to maintain data integrity, even if an item is modified at the same time by different servers. CIM configures the `ClientLockManager.properties` and the `ServerLockManager.properties` files to ensure all servers are using the correct ports. For additional information on `ClientLockManager` and `ServerLockManager` properties, refer to the SQL Repository Caching section in the *ATG Repository Guide*.

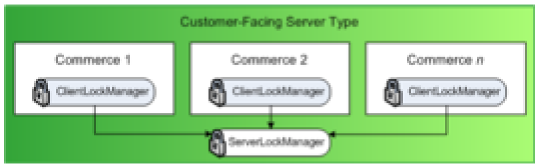
CIM locates the default client lock manager to `LockManager=/atg/dynamo/service/ClientLockManager`. By default, the `ClientLockManager` component has its `useLockServer` property set to `false`, which disables the lock server. To use locked mode repository caching, this property must be set to `true`. For example:

```
$class=atg.service.lockmanager.ClientLockManager
lockServerAddress=tartini,corelli
lockServerPort=9010,9010
useLockServer=true
```

ServerLockManager on the Customer-Facing Server

The customer-facing server cluster defines a primary `ServerLockManager` instance. Additionally, there is a defined `ClientLockManager`, which points to the primary `ServerLockManager`.

You can configure backup `ServerLockManager` instances for redundancy if needed. The following diagram shows a typical customer-facing configuration with three commerce servers that have `ClientLockManagers` that each point to the primary `ServerLockManager`:



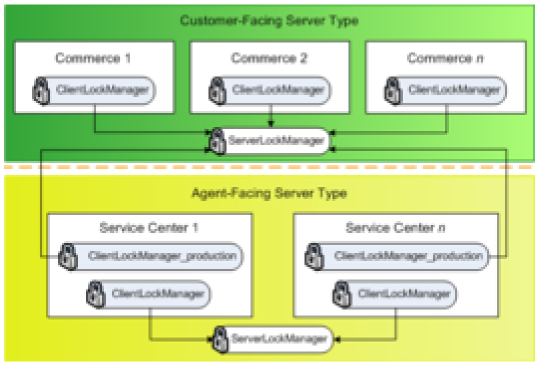
Customer-Facing Lock Management

For information on setting up the `ServerLockManager`, refer to the *ATG Installation and Configuration Guide* and the *ATG Repository Guide*.

ClientLockManagers on the Agent-Facing Server

Agent-facing servers use the customer-facing server lock manager for any shared repositories, and individually scheduled services.

The agent-facing clusters define a `ClientLockManager_production`, which points to the `ServerLockManager` used by the customer-facing server. The following diagram displays both the customer-facing server described above and the agent-facing server configuration. The agent-facing configuration displays the two Service Center instances that each contain a `ClientLockManager_production` that points to the `ServerLockManager` used by the three customer-facing commerce servers. The two Service Center instances contain their own `ClientLockManagers` that each point to the agent-facing `ServerLockManager`:



Customer and Agent-Facing Lock Management

For detailed information on configuring `LockManagers`, refer to the *ATG Repository Guide*.

Accessing Commerce Service Center

Before you can access Commerce Service Center, you must start the servers by entering the appropriate run command in your application server. Refer to your application server documentation for further information.

Once your servers are running in accordance with your application server, you can access the Commerce Service Center application using the following URL:

```
http://hostname:port/agent
```

The `hostname` is the name of the machine on which Commerce Service Center is running. The `port` is the port number that your application server uses to listen for requests; see the *ATG Installation and Configuration Guide* for your application server for the default port number.

Working with Multiple Sites

If you have multiple customer-facing sites created, you can configure Commerce Service Center, using Site Administration, to share data between the sites. Sites can be used for such things as providing localized store information, branded segments, or creating running promotions.

Multiple sites within Commerce Service Center allow the agent to select a specific site, search throughout sites, and to set site context. Site context ensures that configuration of the current site will affect the Commerce Service Center display and control the availability of assets such as products, SKUs, catalogs, and price lists. When an agent changes from one site to another the site context and potentially the available assets will also change.

When configuring your system, you identify the assets that are available to each site using Site Administration or Commerce Merchandising. The sites that can be accessed by Commerce Service Center are configured using Site Administration in the Business Control Center. These settings include:

- Enabled Site – Sites that are available to ATG applications such as Commerce Service Center
- Commerce Site – Sites that are displayed in a Commerce application
- Site Icon – An icon that identifies each site. The pixel limit for icons is 16x16
- Default Site – Sets the site context when the agent logs in. This information is obtained from the site repository
- Default Catalog – The catalog to use with a site unless other selection logic is available
- Default Price List and Sale Price List – Price lists to use with a site unless other selection logic is available
- Site Priority – Identifies the priority of the site if it is a member of multiple sites

For additional information on configuring Commerce Service Center using Site Administration, refer to the *ATG Multisite Administration Guide*.

Enabling Multisite

Commerce Service Center can be configured to recognize and use multiple sites with the `siteManager multiSiteEnabled` property. The `isMultiSiteEnabled` method allows you to conditionally render the Commerce Service Center UI for multiple sites. Refer to the *ATG Multisite Administration Guide* for additional information.

Configuring the Default Site

To configure the default site, use the `/atg/commerce/custsvc/util/CSRConfigurator` component to set the `defaultSiteId` property. This sets the default current site when agents log into Commerce Service Center.

Refer to the Using the CSRConfigurator Component (page 47) section for additional information. To load an agent-specific default site, override the `getAgentDefaultSiteId` method in `CSRAgentTools`.

Configuring the Default Site Icon

Commerce Service Center can display a default site icon, should a site not have an associated icon. Use the `/atg/commerce/custsvc/util/CSRConfigurator` component to set the `defaultSiteIconURL` property to display the default icon. Refer to the Using the CSRConfigurator Component (page 47) section for additional information.

Configuring a Site Icon

You can configure site icons to display on specific pages. The images are configured for each site within the site repository and must be downloaded to their appropriate location. To set a site icon, you must update the `siteConfiguration` item descriptor and add a `siteIcon` property with the location. The following example sets a `siteIcon` property value for Site A and for Site B:

```
<update-item item-descriptor="siteConfiguration" id="SiteA">
  <set-property name="siteIcon">
    <![CDATA[/DCS-CSR/images/icons/icon_site_a.gif]]>
    </set-property>
</update-item>

<update-item item-descriptor="siteConfiguration" id="SiteB">
  <set-property name="siteIcon"><![CDATA[/DCS-CSR/images/icons/icon_site_b.gif]]>
    </set-property>
</update-item>
```

Configuring Shareables

When you configure Commerce, you identify the shareables that are used in both Commerce and Commerce Service Center. For information on configuring these shareables, refer to the *ATG Commerce Programming Guide* and the *ATG Multisite Administration Guide*.

4 Configuring Order and Profile Search

This chapter describes the following Commerce Service Center configurations for configuring order and profile searches.

Setting Up Order and Profile Search

Commerce Service Center uses an embedded search method for orders and customer profile searches that does not require the installation and set up of a Search Administration server. Instead, this search method is configured and administered using the Dynamo Server Admin.

Note: This search process is not used for ticket, product or catalog searches.

To run order and profile searches, add the `DCS.Search.Order.Index` module to your EAR file on each customer-facing and management server when you run the CIM scripts. The `DPS.Search.Index` module, which is included when you install `DCS.Search.Order`, runs profile searches. `DCS.Search.Order.Index` runs order searches. These modules capture changes that are made from the storefront. If these modules are not installed, profiles or orders that are added or updated will not be indexed and will not be searchable.

Using Live Indexing and Endeca MDEX Catalog Search

When using both live indexing and the Endeca MDEX catalog search, your live indexing instances must not run the ATG Endeca MDEX modules. If you have ATG Search configuration changes in the storefront module that are required on the live indexing server, you must package and install those configurations in a separate module. This allows them tobe included on the live indexing instance without including additional store front configuration information.

Order and Profile Search Overview

The order and profile search configuration is comprised of the following:

- A Routing Component – The routing component allows you to create and administer search environments, as well as handle the live updates of the index
- Indexing Output Configurations – An indexing output configuration (IOC) is the `/atg/userprofiling/search/ProfileOutputConfig` or the `/atg/commerce`

`/search/OrderOutputConfig` component with an associated definition file. The definition file is a standard XML document that defines the repository items, such as fields, that create the search index. Separate indices are created for orders and profiles, and, as such, a definition document is created for each index: `profile-output-config.xml` and `order-output-config.xml`. For detailed information on Indexing Output Configurations, refer to the *Creating XHTML Documents from Repository Items* section of the *ATG Search Administration Guide*

- An Indexing Service – The indexing service racks updates in a particular environment and passes information to the routing component to ensure the index is updated
- Indexed Items Groups – The indexed items group defines the type of items that will be included in the index. By default, only submitted orders will be indexed

Indexing Methods

Order and profile search support both incremental and bulk indexing. Incremental indexing occurs in real time, and as such, does not require an index deployment.

- Incremental Indexing – Enabled by default, it is configured using the indexing service to check for modifications to orders and profiles every five seconds from multiple environments. During some operations, such as modification, backup or restoration of an environment, incremental indexing is not available. During these operations, incremental updates are queued until incremental indexing is restored
- Bulk Indexing – Started from the `/atg/userprofiling/search/ProfileOutputConfig` or the `/atg/commerce/search/OrderOutputConfig` component, bulk indexing recreates the complete index and is performed in a temporary staging environment. When the bulk indexing job is completed, the temporary staging environment is swapped with the live environment. During the time that the bulk index is running, searches are performed in the live environment. However, incremental updates are queued and not applied until the bulk indexing job has completed and the environments are swapped

Order and Profile searches require separate Search projects. This allows both orders and profiles to be indexed concurrently, as indexing jobs from separate search environments can be run in parallel.

Note: You cannot search for incomplete orders. Incomplete orders, or orders that have yet to be submitted, are not indexed, and as such, are not contained within the search index.

Order and Profile Search Components

The following components from `DAF.Search.Index` are used for order and profile searches. For additional information on these components, refer to the *ATG Search Installation and Configuration Guide*.

- `/atg/search/repository/IncrementalItemQueue`
- `/atg/search/repository/IncrementalItemQueueRepository`
- `/atg/search/repository/ConfigStatePersister`
- `/atg/search/repository/ConfigAndRepositoryPersister`
- `/atg/search/repository/IncrementalLoader` – `DPS.Search.Index` appends the `ProfileOutputConfig` to the `monitoredOutputConfig` property. `DCS.Search.Order.Index` appends the `OrderOutputConfig` to the `monitoredOutputConfig` property. When the `IndexingPeriodicService` runs, the `IncrementalLoader` will process queued indexing requests for each of the `/atg/userprofiling/search/ProfileOutputConfig` or the `/atg/commerce/search/OrderOutputConfig` components.
- `/atg/search/repository/BulkLoader`

- `/atg/search/repository/IndexingPeriodicService` – Configured in `DPS.Search.Index` to run every two seconds by default. All servers are configured with the `IndexingPeriodicService` enabled but they only index any `/atg/userprofiling/search/ProfileOutputConfig` or `/atg/commerce/search/OrderOutputConfig` component that has the component `enableIncrementalLoading=true` and `incrementalUpdateSeconds` > `0`. As such, you should only set the `incrementalUpdateSeconds` > `0` on servers that will be used for indexing.

  The `OrderOutputConfig.incrementalUpdateSeconds` is set to 30 by default to ensure that at least one server will process the queued incremental indexing requests. In a production environment, it is best to set `incrementalUpdateSeconds=-1` on the agent-facing server and configure `incrementalUpdateSeconds` > `0` on the management server. Incremental indexing can be enabled on any background server that is running `DPS.Search.Index`, and `DCS.Search.Order.Index` for servers.

  The `IndexingPeriodicService` clears any expired configuration claim locks each time it runs using the `checkExpiredConfigurationClaimsIntervalSeconds` property, which checks for expired configuration claim locks every *n* seconds. Each time the `IndexingPeriodicService` runs, it checks to see if the time elapsed has exceeded this interval.

The following components are also used in the order and profile search process:

Component	**Description**
`/atg/userprofiling/search/ProfileOutputConfig`	The profile Indexing Output Component. Defined in the `DPS.Search.Index` module so that an indexing request can be queued wherever a profile is added or updated. The component creates the `profile-output-config.xml` definition file.
`/atg/commerce/search/OrderOutputConfig`	The order Indexing Output Component. Defined in the `DCS.Search.Order.Index` module so that an order indexing request can be queued wherever an order is added or updated. The component creates the `order-output-config.xml` definition file.
`/atg/search/repository/LiveDocumentSubmitter`	A special document submitter that updates the index in near real time. This component is defined in the `DAF.Search.Index` module.
`/atg/commerce/search/OrderProfileIdPropertyAccessor`	Uses the `atg.repository.search.indexing.accessor.ItemIdPropertyAccessor` class that retrieves properties from an item where the item is referenced by ID. In this case, the order item references the user item by Profile ID.
`/atg/userprofiling/search/AddressPropertyAccessor`	Uses the `atg.repository.search.indexing.accessor.ConcatenatePropetyAccessor` class to concatenate `address1`, `address2` and `address3` into a single indexed address meta property.
`/atg/search/repository/AlphaNumericPropertyAccessor`	Removes non-alpha-numeric characters from the indexed meta property. Used to strip white space and punctuation from `phoneNumber` to promote better search consistency.

Component	Description
`/atg/search/routing/respository/SearchConfigurationRepository`	The `shardConfig` and `shard` properties store configuration information on how to shard your search environment, where to store the logical partitions and what information is stored in each shard.

Configuring Live Indexing for Oracle ATG Web Commerce Search

Important: When configuring a live indexing server, please note that it must run on a separate dedicated server. Live indexing cannot run on your Commerce Service Center servers or storefront servers, which use simple caching mode for items within both the Order and Profile Adapter repositories. Indexing servers must disable caching. If the indexing server does not disable caching, updates to orders and profiles will not be indexed.

For every server that creates, updates or deletes profiles or orders you must perform the following:

1. Add `DPS.Search.Index` (which includes the `DCS.Search.Order.Index` module) to the EAR file on the agent-facing servers, if not already present. Also, add these modules to the EAR file for your customer-facing and management servers.

2. During your CIM installation, you identified an internal server that is your live indexing server. To create a full index, the indexing engine requires a clean partition. The clean partition is a file from which all indexes are created. As such, you need to identify the location of the clean partition by creating a `/localconfig/atg/search/routing/Configuration.properties` file. Use the `cleanPhysicalPartitionPath` property to identify the full path to the clean partition.

   There is a copy of the clean partition located at `<Searchdir>/SearchEngine/`*`operatingsystem`*`/data/initial.index.` To resolve the path correctly, use a relative path to identify the clean partition location as a local copy. For example:

   ```
   cleanPhysicalPartitionPath =../data/initial.index
   ```

Creating Search Indexing Environments

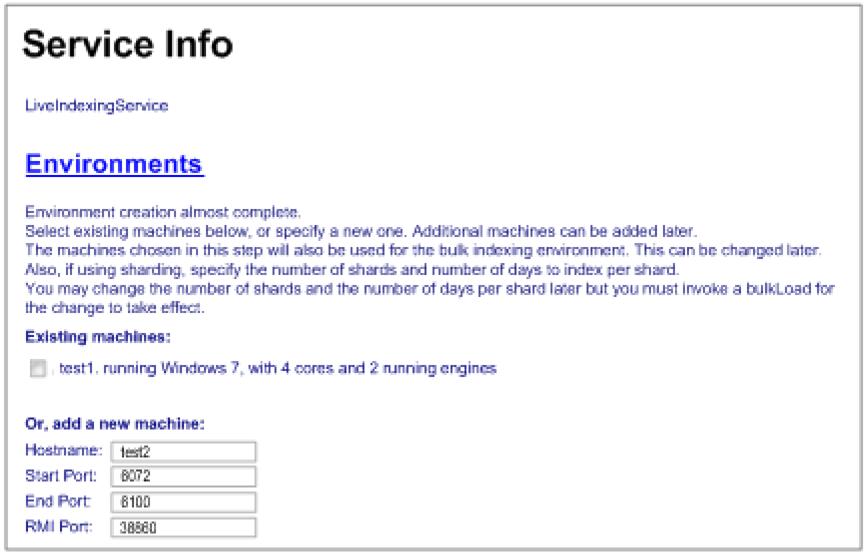
Each Search project creates an indexing environment on the local machine. Configuration of order and profile search is done using the Dynamo Server Admin. For additional information on search environments, refer to the *Managing Search Environments* chapter of the *ATG Search Administration Guide*.

1. On your Search server, start the remote server by running the `/Search/Search10.2/SearchAdmin/bin/startRemoteLauncher` script.

2. Open the Dynamo Server Admin at `http://hostname: port/dyn/admin/nucleus/atg/search/routing/LiveIndexingService/`

3. Click the Create a New Live Indexing Environment link. Two buttons appear for creating order and profile search environments.

4. Click the buttons to create one of the environments and enter the details of your search engine. You can either select the checkbox for your existing machine or enter the address of another machine.


Service Info

LiveIndexingService

Environments

Environment creation almost complete.
Select existing machines below, or specify a new one. Additional machines can be added later.
The machines chosen in this step will also be used for the bulk indexing environment. This can be changed later.
Also, if using sharding, specify the number of shards and number of days to index per shard.
You may change the number of shards and the number of days per shard later but you must invoke a bulkLoad for the change to take effect.

Existing machines:

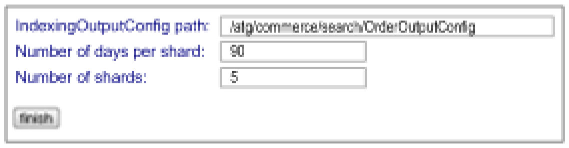
☐ test1, running Windows 7, with 4 cores and 2 running engines

Or, add a new machine:

Hostname:	test2
Start Port:	8072
End Port:	8100
RMI Port:	38860

5. Repeat steps 3 and 4 to create additional indexing environments.

6. Enter the `IndexingOutputConfig` path to use.



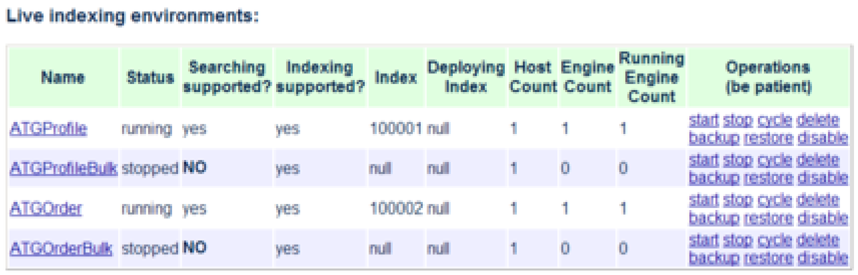
IndexingOutputConfig path:	/atg/commerce/search/OrderOutputConfig
Number of days per shard:	90
Number of shards:	5

finish

7. If creating shards, enter the number of days per shard and the number of shards to create. For information on shards, refer to the Purging Older Orders (page 22)section. This enables you to add hosts to a logical partition or shard.

   Note: Sharding can be enabled by setting the `/atg/commerce/search/OrderOutputConfig.shardingEnabled` property to `true` on the live indexing server.

8. Click the Environments link to display the current default environments. Two environments, a live indexing environment and a bulk indexing environment, are created for both Profile and Order processes.



Live indexing environments:

Name	Status	Searching supported?	Indexing supported?	Index	Deploying Index	Host Count	Engine Count	Running Engine Count	Operations (be patient)
ATGProfile	running	yes	yes	100001	null	1	1	1	start stop cycle delete backup restore disable
ATGProfileBulk	stopped	**NO**	yes	null	null	1	0	0	start stop cycle delete backup restore disable
ATGOrder	running	yes	yes	100002	null	1	1	1	start stop cycle delete backup restore disable
ATGOrderBulk	stopped	**NO**	yes	null	null	1	0	0	start stop cycle delete backup restore disable

Note: If you choose different environment names than the default `ATGProfile` and `ATGOrder`, you must edit the `/atg/userprofiling/search/ProfileSearchConfiguration` and `/atg/commerce/search/OrderSearchConfiguration` components to reflect the search environment names and corresponding logical partition names.

9. You can select a specific environment to manage from the list.

   For additional information, refer to the *Configuring Remote Indexing Engines* section in the *ATG Search Installation and Configuration Guide*.

Preloading the Index

If you would like to preload the index with existing orders and/or profiles, perform the following steps:

1. Using the Dynamo Server Admin, open the `/atg/commerce/search/OrderOutputConfig` component to preload orders, or the `/atg/userprofiling/search/ProfileOUtputConfig` component to preload profiles.

2. Invoke the `bulkload` method by selecting it from the Methods list. This will recreate the index with orders and/or profiles that are already in the repository.

Enhancing Performance of Bulk Loads

In environments where you are working with large data volumes, you can enhance the performance of the `bulkLoad` indexing process. By default, the `/atg/commerce/search/OrderOutputConfig/threadedItemQueueBatchSize` is set to 4, which is equal to twenty thousand orders retrieved for batch processing. This results in a high number of unconstrained executions of the SQL query used to initiate the `bulkLoad` indexing process.

To reduce the number of unconstrained SQL query requests and improve `bulkLoad` performance, increase the `threadedItemQueueBatchSize` parameter to no more than 20. Setting the value higher than 20 could result in Out of Memory exceptions and database time out exceptions.

Purging Older Orders

You can purge older orders from the index using sharding, which horizontally partitions orders into separate logical partitions. Each of these partitions is represented by a single shard, which defines an indexed date range of orders by order creation date.

For example, if you have more than a full year of orders, you could divide the order into five separate shards that each contain 90 days of orders. The five shards are assigned date ranges based upon the date that the index was created. The date range for each shard is fixed, which keeps orders in a known logical partition, allowing orders to be updated quickly.

Continuing with the example, you would have the following shards and logical partitions configured:

Shard/Logical Partition	Date Ending	Date Starting
LP5	4/5/2011	1/5/2011
LP4	1/5/2011	10/7/2010
LP3	10/7/2010	7/9/2010
LP2	7/9/2010	4/10/2010

Shard/Logical Partition	Date Ending	Date Starting
LP1	4/10/2010	1/10/2010

Every 90 days, the oldest logical partition will be deleted, and a new logical partition will be created to index new items:

Shard/Logical Partition	Date Ending	Date Starting
LP6	7/6/2011	4/5/2011
LP5	4/5/2011	1/5/2011
LP4	1/5/2011	10/7/2010
LP3	10/7/2010	7/9/2010
LP2	7/9/2010	4/10/2010
LP1 (deleted)	4/10/2010	1/10/2010

Note that a sixth shard is required to index future orders, as all existing shards already hold data.

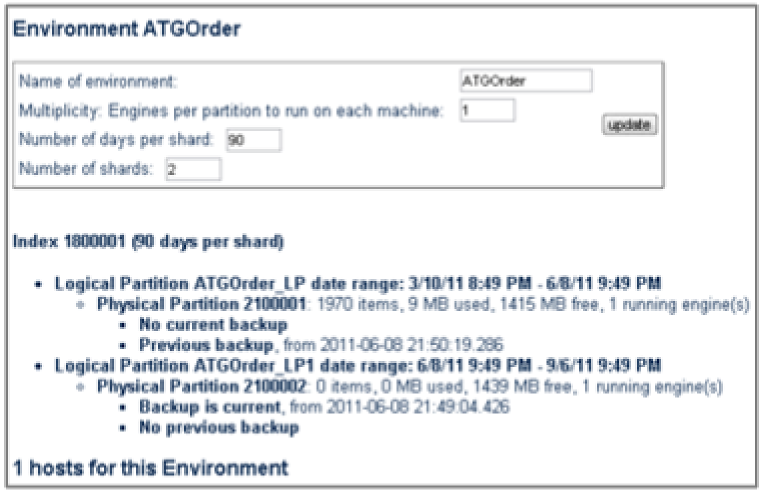
The Sharding Process

By default, sharding is disabled, but can be enabled using the `shardingEnabled` property in `atg/commerce/search/OrderOutputConfig`. Note that shards are created based on the creation date of the order, not the completion date of the order. The `/atg/commerce/search/OrderSharder` implements the sharding of orders and the `/atg/commerce/search/OrderShardRotationService` component's `daysBeforeExpiration` property defines how many days prior to the shard's expiration date the shard should be rotated.

The `/atg/search/routing/respository/SearchConfigurationRepository` contains the following shard items:

- `shardConfig` – Defines the shard configuration. Both the bulk and live search environments have their own `shardConfig`
- `shard` – Defines the date range for the shard and the name of the logical partition for the shard

Once a `bulkLoad` method has been run using `/atg/commerce/search/OrderOutputConfig`, the shard information will appear when you select the `ATGOrder` search environment. For example:


Environment ATGOrder

Name of environment: ATGOrder

Multiplicity: Engines per partition to run on each machine: 1

Number of days per shard: 90

Number of shards: 2

Index 1800001 (90 days per shard)

- **Logical Partition ATGOrder_LP date range: 3/10/11 8:49 PM - 6/8/11 9:49 PM**
    - **Physical Partition 2100001**: 1970 items, 9 MB used, 1415 MB free, 1 running engine(s)
        - **No current backup**
        - **Previous backup**, from 2011-06-08 21:50:19.286
- **Logical Partition ATGOrder_LP1 date range: 6/8/11 9:49 PM - 9/6/11 9:49 PM**
    - **Physical Partition 2100002**: 0 items, 0 MB used, 1439 MB free, 1 running engine(s)
        - **Backup is current**, from 2011-06-08 21:49:04.426
        - **No previous backup**

1 hosts for this Environment

Search Environment Information

Configuring Shards for Production Systems

Because each shard represents a logical partition, you must set the `AEConfig.xml` settings to accommodate for the largest period of activity. The `AEConfig.xml` file configures the search engine's `MemoryReserveSize` setting, which defines how much memory is reserved for an index. This memory may not be allocated until the index grows.

For example, if each shard contains three months of orders, but one shard holds holiday orders, that shard may hold double the amount of orders than the other two shards. You must set the `MemoryReserveSize` to support the maximum number of orders within the shard date range. In this example, you would set the `MemoryReserveSize` site to hold the largest number of orders that might occur during the year, or the holiday season. This ensures that shards created during peak order times have enough memory reserved.

For additional information on setting the `MemoryReserveSize`, refer to the Adjusting Physical Partition Size section in the *ATG Search Installation and Configuration Guide*.

The amount of memory needed is equal to the amount of memory required to index the content, plus the `MemoryThreshold` and heap size per process. It is best that you allocate a minimum of two cores per engine (per physical partition). Use the following formula to determine the number of required CPUs:

```
2 x P_partitions x ((N_days / M_range) + 1)
```

- `P_partitions` – The number of physical partitions per shard. If all items in a shard fit in a single physical partition, then `P_partitions = 1`
- `N_days` – The total number of days to index
- `M_range` – The number of days per shard

For example, for 360 days of orders, you might divide them into four partitions of 90 days each. Including the additional extra partition for future orders, and assuming that all items in a shard fit into a single physical partition, the formula could look like this:

```
2 x 1 x ((360/90) + 1) = 10 cores
```

Note that this example only takes into account the order live indexing processes and does not count other processes such as the Java application server, profile live indexing or other indexes, such as catalog search, solutions, etc.

Performing a Manual Re-Index

It is possible to manually re-index orders or profiles. By default, you can re-index using a date range or an ID.

The `/atg/commerce/search/OrderOutputConfig` contains the `manualIndexRequests` property that enumerates the types of queries used to manually re-index orders. The default queries perform indexing by creation date, last modified date, submitted date, or ID:

```
manualIndexRequests=\
  /atg/commerce/search/OrderManualIndexRequestByCreationDateRange,\
  /atg/commerce/search/OrderManualIndexRequestByLastModifiedDateRange,\
  /atg/commerce/search/OrderManualIndexRequestBySubmittedDateRange,\
  /atg/commerce/search/OrderManualIndexRequestById
```

The `/atg/userprofiling/search/ProfileOutputConfig` contains the `manualIndexRequests` property. By default, queries perform indexing for profiles by last modification date or by ID:

```
manualIndexRequests=\
  /atg/userprofiling/search/ProfilesManualIndexRequestById,\
  /atg/userprofiling/search/
     ProfileManualIndexRequestByLastModifiedDateRange
```

Each of the `ManualIndexRequests` defines an RQL query that sets the criteria for the indexing request.

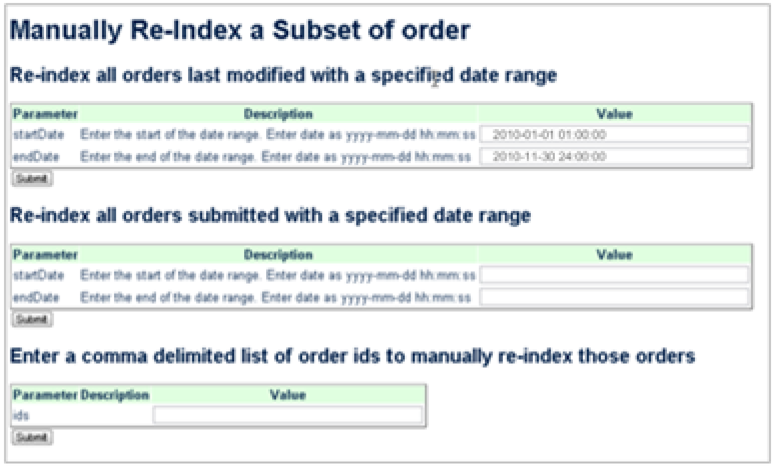
To manually re-index:

1. Using the Dynamo Server Admin, access the `OrderOutputConfig` component to re-index orders, or the `ProfileOutputConfig` component to re-index profiles.

2. Select Manually Re-Index a Subset of Orders or Profiles.

3. Enter the parameters for performing a re-index. The following example displays a manual re-indexing of a subset of orders that have been modified between January 1, 2010 and November 30, 2010.

   Note: The date range must be entered in the yyyy-mm-dd hh:mm:ss format and must include the time. For example, February 28th at 7:30 p.m. would be 2011-02-28 19:30:00.



Manually Re-Index a Subset of order

Re-index all orders last modified with a specified date range

Parameter	Description	Value
startDate	Enter the start of the date range. Enter date as yyyy-mm-dd hh:mm:ss	2010-01-01 01:00:00
endDate	Enter the end of the date range. Enter date as yyyy-mm-dd hh:mm:ss	2010-11-30 24:00:00

Submit

Re-index all orders submitted with a specified date range

Parameter	Description	Value
startDate	Enter the start of the date range. Enter date as yyyy-mm-dd hh:mm:ss	
endDate	Enter the end of the date range. Enter date as yyyy-mm-dd hh:mm:ss	

Submit

Enter a comma delimited list of order ids to manually re-index those orders

Parameter	Description	Value
ids		

Submit

4. Click Submit to begin the re-indexing process.

   The screen will present a message indicating how many items were queued for indexing.

To review the orders being indexed, enable the `loggingDebug` property in `/atg/commerce/search/OrderLiveDocumentSubmitter`. To review the profiles being indexed, enable the `loggingDebug` property in `/atg/userprofiling/search/ProfileLiveDocumentSubmitter`.

Customizing Manual Indexing

By default, manual re-indexing searches for date ranges and IDs. However, you can customize the re-indexing process. To do this, define an RQL query, or write a custom class that implements the `atg.search.routing.ManualIndexReqest` interface referenced by the `atg.repository.search.indexing.IndexingOutputConfig` class. For information on writing RQL queries, refer to the *ATG Repository Guide*. For information on extending and working with the `IndexingOutputConfig` class, refer to the *ATG Search Administration Guide*.

Adding Searchable Properties

When you perform searches, you may want to sort your search results by a specific criterion. The criterion you select must be indexed before it can be used to sort search results. If the existing index does not contain the property that you want to use for sorting, you must add it. The following section provides information on how to add a property to an index.

The `IndexingOutputConfig` component is defined using the following properties:

- `repository` – The repository referenced by the definition file
- `definitionFile` – An XML indexing definition file that configures the repository item types and properties that are included in the indexing document

For example, the `/atg/commerce/search/OrderOutputConfig` component has the following configuration, which can be accessed using the Dynamo Server Admin:

```
definitionFile=/atg/commerce/search/order-output-config.xml
repository=/atg/commerce/order/OrderRepository
```

The `/atg/userprofiling/search/ProfileOutputConfig` component has the following configuration, which can be accessed using the Dynamo Server Admin:

```
definitionFile=/atg/userprofiling/search/profile-output-config.xml
repository=/atg/userprofiling/ProfileAdapterRepository
```

For detailed information on modifying `IndexingOutputConfig` components, refer to the *Configuring Remote Indexing Engines* section in the *ATG Search Installation and Configuration Guide*.

Working with Definition Files

You can add properties to be indexed by appending them to the definition file associated with the component. The definition file starts by identifying a top-level `item` element that specifies the `item-descriptor` to use, and then lists the properties of that `item` type to include within the index. The properties appear as `property` elements within a `meta-properties` element.

When you specify a meta property in the definition file, you can use the `store-as-meta-index` Boolean attributes to specify how to structure the index. If set to `true`, the value of the property is used as a key in a lookup table, which enables faster retrieval of the document.

For example, the following is a portion of the `ProfileOutputConfig` definition file:

```
<item item-descriptor-name="user" is-document="true">
  <title property-name="login"/>
  <meta-properties>
    <property name="$repositoryId" type="string"/>
    <property name="$url" type="string"/>
    <property name="$baseUrl" type="string" suppress="true"/>
    <property name="$itemDescriptor.itemDescriptorName" type="string"
        suppress="true"/>
    <property name="$repository.repositoryName" type="string" suppress="true"/>
    <property name="login" type="string" store-as-meta-index="true"/>
    <property name="email" type="string" store-as-meta-index="true"/>
    <property name="firstName" type="enum" store-as-meta-index="true"/>
    <property name="lastName" type="string" store-as-meta-index="true"/>
  </meta-properties>
</item>
```

Using the above example, the `ProfileOutputConfig` component is configured by default to use `store-as-meta-index` for the `email` property of the `login` item. Since each e-mail value is unique, if a query includes an exact match “starts with” constraint that specifies the `email` value, the correct profile is retrieved without having to search the entire index. Properties that should be displayed in a results list, or that you want to use to sort the results, should set the `store-as-meta-index="true"` attribute.

Extending the Definition File

To index additional order or profile properties, extend the definition file associated with the `IndexingOutputConfig` component using `xml-combine`.

By adding a new definition file that contains the same Nucleus path to your customization directory, you can layer new property information onto the existing `IndexingOutputConfig` component definition file.

For example, you can create a `/liveconfig /atg/userprofiling/search/profile-output-config.xml` file that appends the default `/atg/userprofiling/search/profile-output-config.xml` file by adding the `middleName` property to the index:

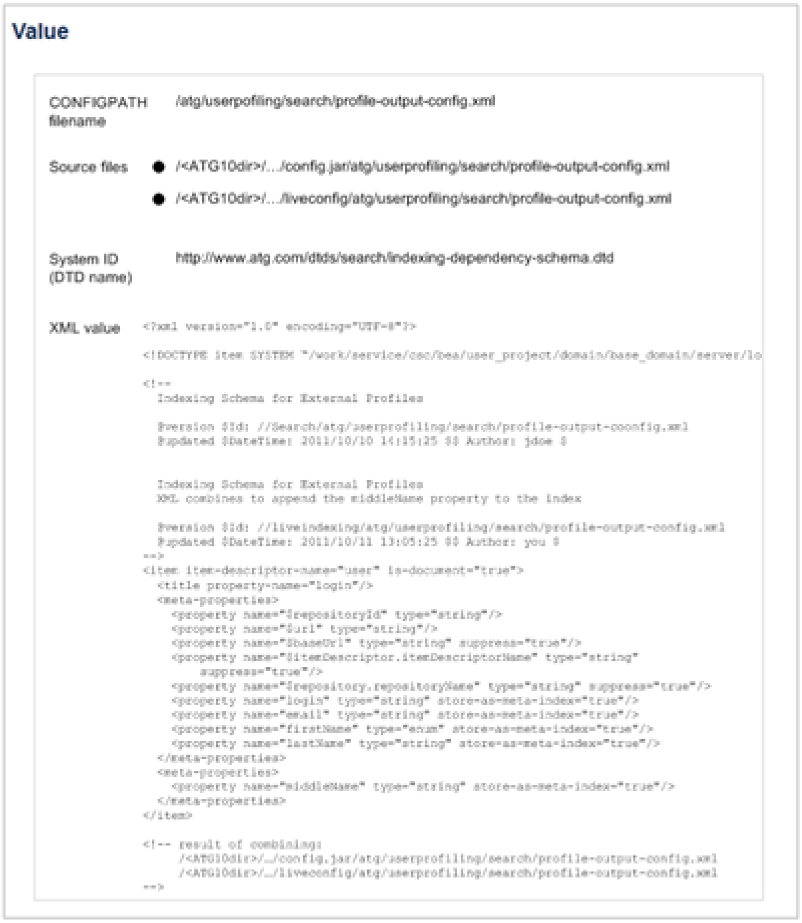
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE item PUBLIC //DTD RepositoryOuput Specifier 1.0//EN"
"http://www.atg.com/dtds/search/indexing-dependency-schema.dtd">

<!—Indexing Schema for External Profiles
   XML combines to append the middleName property to the index
-->

<item item-descriptor-name="user" >
  <meta-properties>
    <property name="middleName" type="string" store-as-meta-index="true"/>
  </meta-properties>
</item>
```

When you build your customization module, the system will run `xml-combine` and combine these files into one file. Note that you do not need to explicitly set the `xml-combine` attribute. By default, the contents of the tag in the second file are appended to the contents of the tag in the first file. Refer to the *ATG Platform Programming Guide* for information on combining XML files.

Once you have rebuilt your customization module, invoke the `bulkLoad` method on the `IndexingOutputConfig` component. You can then review the `ProfileOutputConfig` component's definition file in the Dynamo Server Admin. The definition file will display the XML value after the combination of the two XML files.

Using the previous examples, the definition file of the `ProfileOutputConfig` component now contains the default properties, as well as the custom `middleName` property you created in your customization module.

Note: The following is an example that contains modified path names for demonstration purposes:



Value

CONFIGPATH filename	/atg/userpofiling/search/profile-output-config.xml
Source files	● /<ATG10dir>/…/config.jar/atg/userprofiling/search/profile-output-config.xml ● /<ATG10dir>/…/liveconfig/atg/userprofiling/search/profile-output-config.xml
System ID (DTD name)	http://www.atg.com/dtds/search/indexing-dependency-schema.dtd

XML value

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE item SYSTEM "/work/service/csc/bea/user_project/domain/base_domain/server/lo

<!--
  Indexing Schema for External Profiles

  @version $Id: //Search/atg/userprofiling/search/profile-output-coonfig.xml
  @updated $DateTime: 2011/10/10 14:15:25 $$ Author: jdoe $


  Indexing Schema for External Profiles
  XML combines to append the middleName property to the index

  @version $Id: //liveindexing/atg/userprofiling/search/profile-output-config.xml
  @updated $DateTime: 2011/10/11 13:05:25 $$ Author: you $
-->
<item item-descriptor-name="user" is-document="true">
  <title property-name="login"/>
  <meta-properties>
    <property name="$repositoryId" type="string"/>
    <property name="$url" type="string"/>
    <property name="$baseUrl" type="string" suppress="true"/>
    <property name="$itemDescriptor.itemDescriptorName" type="string"
        suppress="true"/>
    <property name="$repository.repositoryName" type="string" suppress="true"/>
    <property name="login" type="string" store-as-meta-index="true"/>
    <property name="email" type="string" store-as-meta-index="true"/>
    <property name="firstName" type="enum" store-as-meta-index="true"/>
    <property name="lastName" type="string" store-as-meta-index="true"/>
  </meta-properties>
  <meta-properties>
    <property name="middleName" type="string" store-as-meta-index="true"/>
  </meta-properties>
</item>

<!-- result of combining:
     /<ATG10dir>/…/config.jar/atg/userprofiling/search/profile-output-config.xml
     /<ATG10dir>/…/liveconfig/atg/userprofiling/search/profile-output-config.xml
-->
```

Profile Output Configuration Example

Once you have added a property to an index, it can now be used to sort search results.

5 Configuring Catalog Search

Commerce Service Center, by default, uses SQL for searching the product catalog. It is possible to extend the search capabilities of the catalog search feature by integrating with Endeca's MDX search engine.

When you configure your Commerce Service Center using CIM, you can select the option to install Endeca Catalog Search. Commerce Service Center uses the Experience Manager configuration that you have established for storefront applications, which generates the same search results for the agent as produced for the storefront. Commerce Service Center provides a generic, yet customizable, presentation of the search results.

Configuring catalog search using the Endeca MDX search engine enables an agent to perform the same search as a customer. An Endeca MDX search engine-based catalog search:

- Displays the same search results to the agent as displayed to the customer
- Provides guided navigation to the agent with result paging and sorting
- Allows an agent to add search term queries and obtain auto suggestions
- Provide site-based catalog configuration and filtering
- Uses Commerce Service Center-specific page displays, such as dimensions, breadcrumbs and search refinements

Catalog Search with Endeca MDEX Prerequisites

Before you can configure Commerce Service Center catalog search to work with the Endeca MDX engine, you must have installed and configured an Endeca MDX server. Commerce Service Center's implementation depends upon the storefront's integration with Endeca MDEX and the Experience Manager configurations that have already been created. Refer to Endeca documentation for additional information on setting up and configuring an Endeca server.

The `DCS-CSR-UI.Endeca` module contains the configuration property files that Commerce Service Center uses to integrate with the storefront's MDEX engine. To continue performing SQL catalog browses and searches, do not include the `DCS-CSR-UI.Endeca` module in the startup list and use only the `DCS-CSR-UI` module.

Using Live Indexing and Endeca MDEX Catalog Search

To use both live indexing for order and profile search and the Endeca MDEX catalog search, your live indexing instances must not run the ATG Endeca MDEX modules. If you have ATG Search configuration changes in the storefront module that are required on the live indexing server, you must package those configurations in a separate module so that they may be included on the live indexing instance without including additional store front configuration information.

Overview of Catalog Search with Endeca MDEX

The search process starts with Commerce Service Center issuing a request, based upon the search criteria entered by the agent. The request incorporates the following:

- Issues a service framework request for the search results panel stack. The request may optionally contain an Endeca content URI as an input parameter. A default URI is used when one is not provided
- The search result panel invokes the `CSRInvokeAssembler` droplet, which issues the search request to the Endeca Assembler using the given content URI
- The search results, in the storefront's format, is parsed into a map of content items that is keyed by the content item type
- A series of page fragments render the content items by type. For example, there are fragments that render `Breadcrumbs`, `RefinementMenus` and `ResultList` content items
- Any follow-on content URIs rendered in the result, such as `NavigationActions`, are wrapped in a service framework request for the search result panel stack by the `ContentRequestURI` droplet

Initiating an Endeca MDEX Catalog Search Request

Requests for search content are made through the catalog search panel stack. The `cmcProductCatalogSearch` panel JSP, `/panels/catalog/endeca/productCatalogSearch.jsp` calls the `CSRInvokeAssembler` servlet bean and includes all of the page fragments used to display product records, dimensions, refinements, paging controls, sorting controls and the search term query form.

Note: Auto Suggestions are sent through a separate request. For information on how auto suggestions work refer to the Configuring Auto-Suggestions (page 40) section.

Determining Content URI

The `productCatalogSearch` page determines the correct content URI using the `contentURI` parameter. If the `contentURI` is not found, the page looks at the cached content URI from the `SearchState`, which is a session-scoped component that holds data on the most recently requested content URI. If the `contentURI` is still not found, the page uses the `defaultcontentURI` value that has been entered in the `configuration.properties` file. It then updates the cache `contentURI` in the `SearchState`:

```
<c:if test="${empty endecaContentURI}">
  <c:set var="endecaContentURI" value="${searchState.lastContentURI}"/>
  <c:if test="${empty endecaContentURI}">
    <c:set var="endecaContentURI"
        value="${endecaConfig.defaultContentURI}"/>
  </c:if>
</c:if>
<dsp:setvalue bean="/atg/commerce/custsvc/catalog/
    endeca/SearchState.lastContentURI" value="${endecaContentURI}"/>
```

Filtering the Requests

The Endeca site and catalog filters are applied during all requests to ensure that the correct data is returned. The `CSRSiteFilterBuilder` extends `SiteFilterBuilder` to set the current site filter based on the current site scope in the search state object. The `CSRCatalogFilterBuilder` sets the catalog filter based on the current site scope, using either the current catalog, or all of the catalogs in the current cart sharing group. For detailed information on the `SiteFilterBuilder` and `CatalogFilterBuilder`, refer to the *ATG-Endeca Integration Guide*.

UI Page Fragments

A PageFragment is a component that defines the URL path and servlet context for a specific JSP page. A number of these components are used to render the Endeca catalog browse and navigation. These components are located under `/atg/commerce/custsvc/ui/fragments/catalog/endeca`.

The following is an example of the `ResultsListContentItem` page fragment:

```
$class=atg.web.PageFragment
URL=/include/catalog/endeca/displayResultListContentItem.jsp
servletContext=DCS-CSR
```

Note that the URL and servlet context can be changed when making customizations to the UI. For defaulted information on working with page fragments, refer to the *ATG Page Developer's Guide*. For additional information on working with customizing catalog search page fragments, refer to the Customizing Search Results (page 42) section.

Encoding Framework URL

All navigation requests to the Commerce Service Center UI must use the `framework.jsp` servlet bean. Therefore, the service framework URL is encoded on all anchor tags, forms and navigation requests. For example, the following URL navigates to the catalog search page:

```
/framework.jsp?ps=cmcCatalogPS&p=cmcProductCatalogSearch
```

Each request for search results also includes an Endeca content URI, such as “/browse”. The content URI is encoded on the service framework URL as a URL parameter. For example:

```
/framework.jsp?_windowid=1&ps=cmcCatalogPS&p=cmcCatalogSearchP&contentURI=/browse/_/
N-1z141n6
```

This encoding is performed on all anchor tags or success URLs that generate search results. The `contentRequestURL` droplet can be used to generate URLs for requesting search results. Refer to the Content Request URL Droplet Servlet Bean (page 37) section.

Defining Navigation Actions

Endeca `NavigationActions` objects contain the meta data for changing the current navigation state, such as selecting a refinement, or paging through the result screens.

The following is an example of a refinement `NavigationAction` in JSON format.

```
"@class": "com.endeca.infront.cartridge.model.Refinement",
"multiSelect": true,
"navigationState": "/Canyon/_/N-1z141qc?Nrpp=12&format=json",
"contentPath": "/browse",
"count": 13,
"siteRootPath": "/pages",
"label": "Canyon",
"properties": { }
```

The content URI for a `NavigationAction` is constructed by the `ContentRequestUrlDroplet`. Using the above example, the content URI result would be:

```
/browse/Canyon/_/N-1z141qc?Nrpp=12&format=json
```

And the resulting URL would be:

```
/framework.jsp?_windowid=1&ps=cmcCatalogPS&p=cmcCatalogSearchP&contentURI=/browse/_/
N-1z141qc&Nrpp=12&format=json
```

CSRInvokeAssembler

Commerce Service Center uses the `CSRInvokeAssembler` servlet bean to make the request to the Endeca Assembler. The servlet bean parses the result from Endeca into a map of `ContentItems` keyed by type (`Map<String.List<? ContentItem)`, which is included as an output parameter named `contentItemMap`. The raw search results are also returned as an output parameter. The `contentItemMap` is a convenience object used for quickly accessing specific content items in the result:

```
<dsp:droplet name="/atg/commerce/custsvc/catalog/endeca/InvokeAssembler">
  <dsp:param name="includePath" value="${endecaContentURI}"/>
  <dsp:oparam name="output">
    <dsp:getvalueof var="contentItemResult" param="contentItem"/>
    <dsp:getvalueof var="contentItemMap" param="contentItemMap"/>
  </dsp:oparam>
</dsp:droplet>
```

Setting the Agent Profile

Because Commerce Service Center is an agent-facing application, the agent's internal profile is the active session profile for all requests. To generate customer-specific results from the Assembler, Commerce Service Center swaps the active customer profile for the duration of the call to the Assembler. To do this, Commerce Service Center implements the Assembler's `GenericInvokeAssemblerCallbackImp` callback interface. This implementation is configured using the `AssemblerTools` component:

```
AssemblerTools.properties:
callbacks=+ProfileInvokeAssemblerCallback
```

Configuring an Endeca MDEX Catalog Search

Once the content items and types are identified, they are associated with a page fragment through the `/atg/commerce/custsvc/catalog/endeca/configuration.properties` file. This configuration file contains the following properties, which are used to configure various aspects of catalog search:

Property	Description
`alternateResultContentItemTypes`	A list of content item types that, if located in the results, will be displayed in the result area of the UI just below the traditional results list. This list also defines the order in which they are displayed. For example: `alternatResultsContentItemTypes= MyContentItem`
`autoSuggestMinLength`	The minimum number of characters that must be typed into the search term box before auto suggestions occur. The default is `3`.
`autoSuggestURI`	The URI used for requesting auto suggestion content. If null, the auto suggestion feature is disabled.
`breadcrumbContentItemType`	The content item type used for breadcrumbs. The default is `Breadcrumbs`.
`collectionPropertyNames`	Identifies which property name of the content item contains the collection to display.
`defaultContentURI`	The root URI for Endeca search requests. The default content URI is expected to be the “root” content URI, which is set in `/atg/commerce/custsvc/catalog/endeca/configuration.properties` using the `defaultContentURI` property. Refer to the*Configuring Catalog Search* (page 31) section for additional information. The default value is `/browse`.
`defaultObjectRenderingPageFragment`	Identifies the page fragment used to render objects from the collection on a result content item. By default, all objects will be handled as products. If multiple record types are being returned in the collection, you can perform type-specific rendering by specifying page fragments using the `objectRenderingPageFragmentsByType` property.

Property	Description
`defaultResultContentItemPageFragment`	The page fragment used by default to render result content items. The default fragment displays a collection of records or repository items that are attached to the content item. The property that contains the collection is specified by the `collectionPropertyName` property.

`endecaResourcedValuePropertyNames`	This property maps the UI key name to a property name. The map is then used to display resource values in the UI. For example, the `refinementMenu.title` key determines the property name of the content item that contains the text to display on a refinement menu title.
	
	There are three resourced value property names used by default in Commerce Service Center:
	
	`refinementMenu.title=displayName`
	`sortOption.label=label`
	`dimensionSearchGroup.displayName=`
	`displayName`

`endecaResourceValuesResourceBundle`	Identifies the resource bundle to use if the `showRawEndecaResourceValue` is false.
`objectRenderingPageFragmentsByType`	This property maps objects types to a page fragment used to render the object in the display. To render by type, Endeca records must have a record type property defined by `recordTypePropertyName`.
`recordProductIdPropertyName`	Specifies the key in the record attribute map that references the Product ID. This property is used by the page fragment that renders a record as a product.
`recordTypePropertyName`	Specifies the key in the record attribute map that contains the type value, which is required to specify different rendering page fragments by type.
`refinementMenuContentItemType`	The content item type used for refinement menus. The default is `RefinementMenu`.

`resultContentItemPageFragments`	Use this property to map custom page fragments to specific result content item types. If a type is not specified in the map, the `defaultResultContentItemPagFragment` is used. The default setting for this property is `resultContentItemPageFragments=`
	`ResultsList=/atg/commerce/custsvc/ui/`
	`fragments/catalog/endeca/`
	`ResultListContentItem`

Property	Description
`defaultResultContentItemPageFragment`	Identifies the page fragment used to render a content item when its type is not configured by the `resultContentItemPageFragments` property.
`resultContentItemTitleLookupKeys`	Looks up the property name that contains the title for the records contained in the content item displayed in the results.
`resultsListContentItemType`	The content item type used for the result list. The default is `ResultsList`.
`showRawEndecaResourcedValues`	Displays the value of the Endeca property in its raw form when true. If false, the value will be used as a key to look up a resourced value in the configured resource bundle.

Catalog Search Servlet Beans and Form Handlers

The following servlet beans and form handlers are used for Endeca Catalog Search.

Content Request URL Droplet Servlet Bean

The `ContentRequestUrlDroplet`, which extends the `FrameworkUrlDroplet`, generates a URL with the content URI or content collection name encoded as a URL parameter. The droplet generates URLs for category browse navigation links, default and follow-on content URIs, as well as the auto suggest collection content.

Class	**`atg.commerce.csr.catalog.endeca.ContentRequestUrlDroplet`**
Components	`/atg/commerce/custsvc/catalog/endeca/ContentRequestUrlDroplet`

The servlet bean identifies the default URL for the search results page using the `searchResultsPageURL` property. By default, the value is set to `/framework.jsp?ps=cmcCatalogPS& p=cmcProductCatalogSearch`. The `ContentRequestUrlDroplet` contains the following parameters. Note that only a single `dimensionId`, `contentPath` or `navigationAction` parameter can be used to determine the content URI. If no value is provided for `contentPath` or `navigationAction`, the default content URI will be used:

Input Parameters

- `url` – Optional. The base URL on which the `ContentURI` parameter will be encoded. If this parameter is not provided, the value will default to the configured `searchResultsPageURL`
- `dimensionId` – Optional. This identifies the dimension ID. When an value is specified, the default content URI is used with a navigation filter parameter (N=x) to specify the dimension value

- `contentPath` – Optional. This parameter specifies a value to use as the content URI in the resulting URL
- `navigationAction` – Optional. An Endeca `NavigationAction` object from which to construct the content URI
- `recordOffset` – Optional. The result record offset for paging. When a value is provided, the `navigationAction` should be a paging template where record offset can be substituted into the URI
- `recordsPerPage` – Optional. Specifies the number of records per page for the results. When a value is provided, the `navigationAction` should be a paging template where records per page can be substituted into the URI

Open Parameters

- `output` – This parameter is rendered only once.

Examples

The following example identifies the `defaultContentURI` as the content path:

```
<dsp:droplet name="ContentRequestURLDroplet">
  <dsp:param name="contentPath" value="${endecaConfig.defaultContentURI}" />
  <dsp:oparam name="output">
    <dsp:getvalueof var="contentURL" bean="ContentRequestURLDroplet.url" />
  </dsp:oparam>
</dsp:droplet>
```

The following example uses a `navigationAction`:

```
<dsp:droplet name="ContentRequestUrlDroplet">
  <dsp:param name="navigationAction" value="${sortOption}" />
  <dsp:oparam name="output">
    <dsp:getvalueof var="contentURL" bean="ContentRequestURLDroplet.url" />
  </dsp:oparam>
</dsp:droplet>
```

The following is an example of a navigation action with a record offset:

```
<dsp:droplet name="ContentRequestURLDroplet">
<dsp:param name="navigationAction"
    value="${resultsListContentItem['pagingActionTemplate']}"/>
<dsp:param name="recordOffset" value="${pageRecordIndex}"/>
<dsp:param name="recordsPerPage" value="${recsPerPage}"/>
<dsp:oparam name="output">
  <dsp:getvalueof var="contentURL" bean="ContentRequestURLDroplet.url"/>
  <a href="#" onclick="atgSubmitAction({url: '${contentURL}'});return false;">
      <c:out value="${pageNumber}"/></a>
</dsp:oparam>
</dsp:droplet>
```

Content Item Results Droplet

This servlet bean iterates over a collection of objects referenced by a `ContentItem` and determines the page fragment that should be used to render each object in the collection.

Class	**`atg.commerce.csr.catalog.endeca.ContentItemResultslDroplet`**
Components	`/atg/commerce/custsvc/catalog/endeca/ContentItemResultsDroplet`

Input Parameters

- `contentItem` – The `contentItem` containing the collection

Output Parameters

- `renderingPageFragment` – The PageFragment for rendering the object in the collection.
- `objectType` – The type of the object in the collection, which is either a repository item descriptor name, or an Endeca record type

Paging Droplet

This servlet bean is used to render the paging controls for catalog search results. Refer to the *ATG API Reference for Commerce Service Center* for additional information on the servlet bean.

Class	**`atg.commerce.csr.catalog.endeca.PagingDroplet`**
Components	`/atg/commerce/custsvc/catalog/endeca/PagingDroplet`

Input Parameters

- `recordsPerPage` – The number of records displayed on a page
- `currentRecordOffset` – The index of the first record displayed
- `totalRecords` – The total number of records returned in the result

Open Parameters

- `prevPageGroup` – Rendered once if there is a previous record group. A previous record group is the group of records previous to the current record group. For example, if the current page group is 4,5,6, a previous page group exists for pages 1,2,3. If the current page group is 1,2,3, this oparam would not be rendered because you are currently on the first page group
- `page` – Rendered once for each page in the current page group with the exception of the current page, which has its own oparam so that it can be uniquely identified
- `currentPage` – Rendered once for the current page
- `nextPageGroup` – Rendered once when there is a next record group. The next record group is the group next in line from the current record group. For example, if the current page group is 7,8,9, the next page group would be 10,11,12. If there is no page 10, the `nextPageGroup` is not rendered

Output Parameters

- `prevPageGroupRecordIndex` – The record index of the first record on the last page of the previous page group
- `nextPageGroupRecordIndex` – The record index of the first record on the first page of the next page group
- `pageNumber` – The page number
- `pageRecordIndex` – The record index for the first record on the page

Site Scope Form Handler

The `SiteScopeFormHandler` is responsible for setting the `SearchState`'s `siteScope` value, which is used to filter the search results by site.

Form Handler	**`/atg/commerce/csr/catalog/Endeca/SiteScopeFormHandler`**
Components	`/atg/commerce/custsvc/catalog/Endeca/SiteScopeFormHandler`

Configuring Auto-Suggestions

The search term page performs term queries as well as dynamic auto-suggestions. **Note:** The auto-suggestion feature must be enabled in your Endeca MDEX configuration before it can be configured in Commerce Service Center. Refer to your Endeca documentation for information.

If auto-suggestions are configured, when an agent enters a search term, the UI will automatically provide dimension suggestions based on the terms entered.

Implementing Auto-Suggestions

Auto-suggestions are generated using a dimension search based on a search terms entered by the agent. Auto-suggestions are not accessed using the catalog search results page, but the `autoSuggestJson.jsp` page, which is designed specifically for requesting auto-suggestion content collections from the Assembler and then rendering the results.

Auto-suggestions in Commerce Service Center are enabled by setting the `autoSuggestURI` property in the `/atg/commerce/custsvc/catalog/endeca/configuration.properties` file. The URI of the auto-suggestion content collection is used to create the URL that requests auto-suggestions. The number of characters that an agent enters into a search field before being provided with suggestions is set using the `autoSuggestMinLength` property in the `configuration.properties` file.

Displaying Auto-Suggestions

The `autoSuggestJson.jsp` page requests the auto-suggestions from the Assembler and renders the dimension search values. The `ContentRequestURLDroplet` is used to create a URL for each dimension search value.

```
<json:object name="dimensionSearchResults">
  <json:array name="dimensionSearchGroups" var="content"
      items="${contentItemMap['AutoSuggestPanel']}">
    <c:forEach var="autoSuggest" items="${content['autoSuggest']}">
      <c:forEach var="dimensionSearchGroup"
          items="${autoSuggest['dimensionSearchGroups']}">
        <json:object>
          <json:property name="displayName">
            <dsp:include src="${displayEndecaResourcedValueFragment.URL}"
                otherContext="${displayEndecaResourcedValueFragment.
                servletContext}">
              <dsp:param name="key" value="dimensionSearchGroup.displayName"/>
              <dsp:param name="contentItem" value="${dimensionSearchGroup}"/>
            </dsp:include>
          </json:property>
        <json:array name="dimensionSearchValues"
            items="${dimensionSearchGroup['dimensionSearchValues']}"
            var="dimensionSearchValue">
          <json:object>
            <dsp:droplet name="ContentRequestURLDroplet">
              <dsp:param name="url"
                  value="${UIConfig.contextRoot}${searchResultPageURL}"/>
              <dsp:param name="navigationAction" value="${dimensionSearchValue}"/>
              <dsp:oparam name="output">
                <dsp:getvalueof var="contentURL"
                    bean="ContentRequestURLDroplet.url"/>
              </dsp:oparam>
            </dsp:droplet>
```

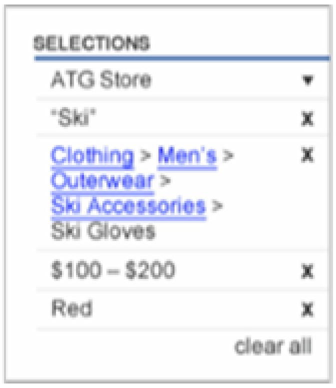
`DimensionSearchValues` are `NavigationActions` that populate the pop-up search term page that is displayed to the agent. The page uses the `displayEndecaResourcedValueFragment` to display the auto-suggestion resource values.

Working with Endeca MDEX Breadcrumbs

Breadcrumbs are elements that display in the Selections section of the menu, identifying the search constraints entered into the query. The breadcrumbs content item is identified by the `BreadcrumbContentItemType` parameter in the configuration file. Endeca breadcrumb content items reference either search or refinement breadcrumbs. The breadcrumb page fragment uses the type of the breadcrumb to determine the correct JSP renderer.

Search breadcrumbs contain information about an active keyword search, such as a search term, and is rendered by the search breadcrumb page fragment at `/atg/commerce/custsvc/ui/fragments/catalog/Endeca/SearchCrumb`.

Refinement breadcrumbs contain information about a refinement and is rendered by the refinement breadcrumb page fragment located at `/atg/commerce/custsvc/ui/fragments/catalog/Endeca/RefinementCrumb`.


SELECTIONS

ATG Store ▾

"Ski" x

Clothing > Men's > Outerwear > Ski Accessories > Ski Gloves x

$100 – $200 x

Red x

clear all

The following example shows how refinement breadcrumbs are displayed:

```
<c:forEach items="${refinementCrumbs}" var="refinementCrumb">
  <dsp:include src="${refinementCrumbDisplayFragment.URL}"
      otherContext="${refinementCrumbDisplayFragment.servletContext}">
    <dsp:param name="refinementCrumb" value="${refinementCrumb}"/>
    <dsp:param name="breadCrumb" value="${breadCrumb}"/>
    <dsp:param name="contentItemMap" value="${contentItemMap}"/>
    <dsp:param name="contentItemResult" value="${contentItemResult}"/>
  </dsp:include>
</c:forEach>
```

Customizing Search Results

You can customize the way that the search results are displayed, including paging and navigation, by working with page fragments. For detailed information on working with page fragments, refer to the *ATG Page Developer's Guide*.

Catalog Search Page Fragments

The main container page for search results is the `productCatalogSearch.jsp` page. All catalog search page fragments exist in the `/atg/commerce/custsvc/ui/fragments/catalog/endeca` name space:

Page Fragment	**Description**
`AutoSuggestJson`	Displays the auto-suggestion page and pop-up.
`Breadcrumbs`	Display the current refinement selections.
`CategoryTree`	Displays the top three levels of the category tree and navigation components.
`DisplayCollectionWithTitle`	Renders a collection of objects in a content item as a table with a title in the Results area.

Page Fragment	Description
`DisplayEndecaResourcedValue`	Displays a resourced value, such as a refinement menu or a sort option label.
`DisplayProduct`	Renders the product information from an Endeca record or product repository item in the Results area.
`DisplayResultContentTitle`	Displays the heading title for the table of results.
`DisplaySubcategories`	Displays sub-category trees in the browser popup menu.
`NoResults`	Displays when no result list or alternate content is found in the results.
`RefinementCrumb`	Display and individual refinement breadcrumb.
`Refinements`	Displays the refinement menus. Each refinement menu contains of list of refinement navigation actions. Refinements also display More and Less links for displaying addition or less data.
`ResultListContentItem`	Displays the `ResultList` content item.
`Results`	Displays the results area.
`ResultsPaging`	Displays the paging controls for the results.
`ResultsSortOptions`	Displays the sort options from the `ResultsList`.
`RootCategories`	Displays a root category list for browse pop-up menu.
`SearchCrumb`	Displays an individual search term breadcrumb.
`SearchTermInput`	Displays the search term input controls.
`SiteScope`	Displays the site scope selection controls.
`SubCategories`	Displays subcategories for a root category in the browse popup menu.

The `/panels/catalog/endeca/productCatalogSearch.jsp` file is the page associated with the results panel and can be changed by modifying the `serviceFramework.xml` configuration file. The following example is a definition of the results panel:

```
<panel-definition>
  <id>
    cmcProductCatalogSearchP
  </id>
  <title-key>
    cmcEndecaCatalogSearchP
  </title-key>
  <content-url>
    /panels/catalog/endeca/productCatalogSearch.jsp
  </content-url>
```

```
</panel-definition>
```

Using Endeca Resourced Values

There are values that are displayed in the results UI that can be localized into the agent's language. The refinement titles, sort option labels and dimension search group labels are values that can be localized for display.

Commerce Service Center provides two ways to deal with the display of these values. You can display the raw value from the Endeca result, or you can treat the value as a key to a value in a resource bundle. The following components and configuration define how these resources are displayed:

- `DisplayEndecaResourcedValue` - This page fragment component defines the JSP page for rendering the resource values. This fragments takes the following parameters:
  - `key` – the UI key for the value being resourced. These are Commerce Service Center-defined values that identify the Endeca resource values. This key is used to identify the configured property name in the content item that contains the resourced value
  - `contentItem` – The content item that contains the resourced value

  For example:

  ```
  <dsp:include src="${displayEndecaResourcedValueFragment.URL}"
  otherContext="${displayEndecaResourcedValueFragment.servletContext}">
  <dsp:param name="key" value="refinementMenu.title"/>
  <dsp:param name="contentItem" value="${refinementMenu}"/>
  </dsp:include>
  ```

- `endecaResourcedValuePropertyName` – This configuration maps the Commerce Service Center UI key to a content item property name. For example:

  ```
  endecaResourcedValuePropertyNames=refinementMenu.title=displayName,\
  sortOption.label=label,\
  dimensionSearchGroup.displayName=displayName
  ```

- `showRawEndecaResourcedValues` – This Boolean configuration value determines if the content item's raw value is display in the UI. If false, the content item's raw value is used as a key to a resource bundle lookup
- `endecaResourcedValuesResourceBundle` – This configuration value provides the name on the resource bundle for doing lookups

Displaying Alternate Content

Commerce Service Center provides a configurable way to render content in the results area other than the traditional results list. For example, depending on the current refinements, the Assembler may produce results that do not include a traditional `ResultList` content item, but may include other content that should be displayed to the agent. Or, you may want to render some content in addition to the traditional result list. This feature is referred to as displaying alternate content.

To display alternate content on the results page, perform the following:

1. Identify the content item types that contain the data to be displayed. By default, the alternate content will be displayed below the standard result list display when it is present.

2. Edit the configuration file to add your new content item:

   ```
   alternateResultContentItemTypes=NewContentItemType1, NewContentItemType2,
   NewContentItemType3
   ```

   Note: If you are creating multiple new content item types, the order they are listed in the configuration will determine the order that they are displayed.

3. Define a `PageFragment` component that identifies the JSP for rendering the alternate content. Configure this page in Commerce Service Center:

   ```
   resultsContentItemPageFragments=NewContentItemType1=/com/app/
   NewContentItemType1PageFragment
   ```

4. Commerce Service Center provides a default page fragment for rendering the alternate content. This default page assumes that there is a collection of product records or repository items on the content item and renders them as a list with a title heading. The property name containing the new list must be configured to use this default page:

   ```
   collectionPropertyNames=+ NewContentItemType1=records
   ```

5. Configure the property containing the title:

   ```
   resultsContentItemTitlePropertyName+=
   NewContentItemType1=newContentItemTitleProperty
   ```

Configuring Oracle for SQL Catalog Searching

If you are configuring your system to perform SQL catalog searches using an Oracle database, you must perform the following:

1. Configure your Oracle `ConText` settings to index the columns that the catalog search queries.

2. You can also set the `simulateTextSearchQueries` of your Product Catalog to `TRUE`.

   Note: When running on a production environment, do not set the `simulateTextSearchQueries` to `TRUE`, as it will affect performance.

3. Add a new index:

   ```
   create index
   dcs_prd_chldsku_sid_idx on dcs_prd_chldsku (sku_id) indextype is
   ctxsys.context;
   ```

   This is needed because the following SQL is generated when searching by SKU:

   ```
   SELECT DISTINCT t1.product_id,t1.product_type
   FROM dcs_product t1, dcs_prd_chldsku t2, dcs_prd_catalogs t3
   WHERE t2.product_id=t1.product_id
   AND t3.product_id=t1.product_id
   AND (CONTAINS(t2.sku_id,'xsku1126',0) > 1
   AND t3.catalog_id = 'masterCatalog')
   ```

6 Programming Commerce Service Center

Commerce Service Center can be modified to fit the needs of your environment. There are a number of ways in which you may customize Commerce Service Center. You can modify the properties of Nucleus components to enable or disable functionality, or you can extend Commerce Service Center classes and create new components from these classes.

The `CSRConfigurator` component is used to set and customize a number of Nucleus component properties that configure Commerce Service Center.

Using the CSRConfigurator Component

The `/atg/commerce/custsvc/util/CSRConfigurator` component configures Commerce Service Center settings, through the following properties:

Property Name	Description
`cartShareableTypeId`	Checks if two sites are in the same cart sharing site group. Also finds sites in the cart sharing site group. This variable is used with environment management. For information on environment management, refer to the *ATG Ticketing User Guide*.
`catalogTools`	Specifies the `catalogTools` component to use. The default is `/atg/commerce/catalog/CatalogTools`, For information on the `catalogTools` component, refer to the *ATG Commerce Programming Guide*.
`commerceSiteType`	Indicates the Site Type, as configured in the Site Administration Console in the BCC Home page. The default for Commerce Service Center is set to `commerce`. For additional information on site types, refer to the *ATG Multisite Administration Guide*.

Property Name	**Description**
`defaultAppeasementLimits`	Sets the default appeasement limit that an agent can offer. The default format for the limit is `USD=500.00`. For additional information on this property, refer to Setting Global Appeasement Limits (page 72).
`defaultCatalogId`	Identifies the default catalog to use when an anonymous user profile is created. For additional information on this property, refer to Configuring Current Catalog and Price Lists (page 89).
`defaultSiteIconURL`	Sets a default icon to use if there is no icon identified for the site. For additional information on this property, refer to Configuring the Default Site Icon (page 15).
`defaultSiteId`	Specifies the global default site that is loaded when the agent logs in. For additional information on this property, refer to Configuring the Default Site (page 14).
`maximumAlmostQualifiedFor` `PromotionsInShortList`	The number of promotion closeness qualifiers to display on the shopping cart. The default number is set to `10`.
`paymentGroupTypeConfigurations`	Identifies the payment group type configurations to display. For additional information on this property, refer to Customizing a Payment Group Type (page 116).
`paymentGroupTypesToBeInitialized`	The type of payment groups that can be initialized by the Payment Group Droplet. These include `creditCard`, `storeCredit` and `giftCertificate`. For additional information on this property, refer to Customizing a Payment Group Type (page 116).
`pricingTools`	Sets the location of the pricing tools used for the running commerce application. The default location is set to `/atg/commerce/pricing/PricingTools`. For additional information on `pricingTools`, refer to the *ATG Commerce Programming Guide*.
`processReturnRequestImmediately`	If set to `true`, then the return request process will begin immediately. The default is set to `false`.
`quantityInputTagMaxLength`	The maximum number of characters that the quantity field will accept. The default number is set to `10`.
`quantityInputTagSize`	The display size of the quantity input field. If the `quantityInputTagMaxLength` size is set to 10, yet the `quantityInputTagSize` is set to 5, the user will be able to enter 10 characters, but only 5 of the 10 characters will be displayed. The default number is set to `10`.
`shippingGroupTypeConfigurations`	Identifies the shipping group type configurations to display. For additional information on this property, refer to Customizing a Shipping Group Type (page 110).

Property Name	**Description**
`shippingGroupTypeToBeInitialized`	The type of shipping groups that can be initialized by the Shipping Group Droplet. This includes `hardgoodShippingGroup` and `electronicShippingGroup`. For additional information on this property, refer to Customizing a Shipping Group Type (page 110).
`supportedPaymentGroupTypes`	Identifies the supported payment group types for modified orders. If an order has a type that is not in the configuration file, it will not load the order. For additional information on this property, refer to Customizing a Payment Group Type (page 116).
`usingGiftLists`	If set to `true`, indicates that the running commerce application uses gift lists. For additional information on this property, refer to *Working with Wish and Gift Lists* (page 51).
`usingInStorePickup`	Specifies if the Commerce In Store Pickup features are enabled. The default is `true`. In Store Pickup uses the `InStorePickupPaymentGroup` and `CashPaymentGroup`, as well as the `InStoreShippingGroup`. Refer to the *Working with Shipping and Payment Groups* (page 105) section for information. For setting up In Store Pickup features, refer to the *ATG Commerce Programming Guide*.
`usingOrderApprovals`	Specifies if Order Approvals are enabled. The default is `true`. For additional information on this property, refer to Enabling the Order Approval Process (page 72).
`usingPriceLists`	If set to `true`, indicates that the running commerce application uses price lists. For additional information on this property, refer to the Using the Current Price List (page 90).
`usingSalesPriceLists`	Indicates if a sales price list has been enabled. For information on this property, refer to the Add On Modifications (page 174) section of Appendix C, *CIM Configuration Components* (page 173)
`usingScheduledOrders`	Specifies if Scheduled Orders are enabled. The default is `true.` For additional information on this property, refer to Enabling and Disabling Scheduled Orders (page 61)

7 Working with Wish and Gift Lists

Gift and wish lists are core Commerce features that allow customers who visit your site to add items to a list that can be accessed by other customers, or saved and referenced for future purchases. Commerce Service Center extends this functionality by enabling an agent to assist a registered customer with the creation and management of their lists. Agents can also assist a customer who wants to search and purchase items from another customer's list. However, agents cannot create gift or wish lists for anonymous shoppers, nor can they delete an item from a gift list, or delete a gift registry.

For detailed information on working with gift and wish lists in Commerce, refer to the *Setting up Gift Lists and Wish Lists* section of the *ATG Commerce Programming Guide*.

Modifying Gift List Forms

Within Commerce Service Center, agents create gift lists using the Gift List menu in the customer's profile. This form is displayed if the `usingGiftlists` property of the `CSRConfigurator` is enabled. Refer to theUsing the CSRConfigurator Component (page 47) section for additional information.

Note that the agent must also have access rights that enable them to create a gift or wish list. Refer to *Setting Up Internal Access Control* (page 77) for information on agent access rights.

By default, agents can create a new gift list by providing the following information:

- Event Name
- Event Description
- Event Date
- Event Type
- Shipping Address
- Public or Private List
- Special Instructions

The gift list form can be modified to include fields or information. There are two properties files that can be configured, the default configuration and the extended configuration.

The default configuration for the Create Gift List form, which is located in `/DCS-CSR-UI/config/atg/commerce/custsvc/ui/fragments/gift/GiftListCreateDefault.properties`, contains the following settings:

```
URL=/include/gift/giftlist/giftlistCreateUIFragment.jsp
servletContext=DCS-CSR
```

The JSP file that creates the Gift List form is located at `/panels/gift/giftlistCreate.jsp` and is configured to import the following page fragments:

```
<dsp: importbean var="defaultPageFragment"
  bean="/atg/commerce/custsvc/ui/fragments/gift/GiftListCreateDefault" />
```

```
<dsp: importbean var="extendedPageFragment"
  bean="/atg/commerce/custsvc/ui/fragments/gift/GiftListCreateExtended" />
```

The extended configuration properties file, `GiftListCreateExtended.properties`, does not contain a reference to a JSP file.

For additional information on configuring and modifying page fragments, refer to the *ATG Service Center UI Programming Guide*.

Rendering Gift Lists

Gift lists are rendered using the `DCS-CSR-UI` module and the `/atg/commerce/custsvc/ui/renderers/ProductSkuRenderer` component `pageOptions` property:

```
pageOptions=\
  actionRenderer=/renderers/order/sku/skuBrowserAction.jsp,\
  giftlistActionRenderer=/renderers/gift/skuGiftlistBrowserAction.jsp,\
  formHandler=/atg/commerce/custsvc/order/CartModifierFormHandler,\
  successPanelStacks=cmcCatalogPS,\
  errorPanelStacks=cmcCatalogPS,\
  successUrlProperty=addItemToOrderSuccessURL,\
  errorUrlProperty=addItemToOrderErrorURL
```

The `skuGiftlistBrowserAction.jsp` file defines the drop-down menu that displays available gift lists, as well as the controls that allow agents to add to lists.

The `/atg/commerce/custsvc/gifts/CSRGiftlistFormHandler`, which is located in the `DCS-CSR` module, extends the Commerce `GiftlistFormHandler`, and enables agents to create and add lists. For information on the `CSRGiftlistFormHandler`, refer to the CSRGiftlistFormHandler (page 55) section.

Displaying Gift List Information

The `DCS-CSR-UI` module's `/atg/commerce/custsvc/ui/fragments/gift/GiftlistDetailsViewDefault` page fragment displays list information.

```
$class=atg.web.PageFragment
URL=/include/gift/giftlist/giftlistDetailsViewUIFragment.jsp
servletContext=DCS-CSR
```

The JSP displays the following information:

- Event Name
- Event Type
- Event Date
- Status (Public or Private)
- Site (If running in multisite mode)
- Shipping Address

When items are contained within gift or wish lists , they are displayed in one of two components that use the servlet bean `atg.svc.agent.ui.tables.TableConfiguration`. The components are located in the `/atg/commerce/custsvc/ui/tables/gift/` directory:

List	View Results	Edit Results
Gift Lists	`/giftlist/GiftlistViewResultsTable`	`/giftlist/GiftlistEditResultsTable`
Wish Lists	`/wishlist/WishlistViewResultsTable`	`/wishlist/WishlistEditResultsTable`

The following is an example of the `GiftlistViewResultsTable`:

```
$class=atg.svc.agent.ui.tables.TableConfiguration
$scope=global
columns=\
        /atg/commerce/custsvc/ui/tables/gift/giftlist/Site,\
        /atg/commerce/custsvc/ui/tables/gift/giftlist/ItemImage,\
        /atg/commerce/custsvc/ui/tables/gift/giftlist/ItemDescription,\
        /atg/commerce/custsvc/ui/tables/gift/giftlist/Desired,\
        /atg/commerce/custsvc/ui/tables/gift/giftlist/Purchased

tablePath=/atg/commerce/custsvc/ui/tables/gift/giftlist/GiftlistViewResultsTable
tablePage=/atg/commerce/custsvc/ui/tables/gift/giftlist/GiftlistTablePage
```

Configuring Gift List Search

The gift list search instance can be customized to include additional fields or information. Both default and extended properties files can be modified.

The default configuration property file is available at /DCS-CSR-UI/config/atg/commerce/custsvc/ui/fragments/gift/GiftListSearchDefault.properties and contains the following:

```
URL=/include/gift/search/giftlistSearchUIFragment.jsp
servletContext=DCS-CSR
```

The following JSP files are used for configuring the Gift List Search form:

JSP File	Description

/panels/gift/giftlistsearch.jsp	Calls the following page fragments:
	
	`<dsp: importbean var="defaultPageFragment" bean="/atg/commerce/custsvc/ui/fragments/gift/GiftListSearchDefault" />`
	`<dsp: importbean var="extendedPageFragment" bean="/atg/commerce/custsvc/ui/fragments/gift/GiftListSearchExtended" />`

/panels/gift/searchResults.jsp	Calls the following page fragments:
	
	`<dsp: importbean var="defaultPageFragment" bean="/atg/commerce/custsvc/ui/fragments/gift/GiftListSearchResultsDefault" />`
	`<dsp: importbean var="extendedPageFragment" bean="/atg/commerce/custsvc/ui/fragments/gift/GiftListSearchResultsExtended" />`

The /panels/gift/giftListSearchGrid.jsp provides the gift list search results grid:

```
/atg/commerce/custsvc/ui/tables/gift/GiftlistGrid.properties
$class=atg.svc.agent.ui.tables.GridConfiguration
columns=\
   /atg/commerce/custsvc/ui/tables/gift/search/EventName,\
   /atg/commerce/custsvc/ui/tables/gift/search/LastName,\
   /atg/commerce/custsvc/ui/tables/gift/search/FirstName,\
   /atg/commerce/custsvc/ui/tables/gift/search/CustomerID,\
   /atg/commerce/custsvc/ui/tables/gift/search/EventType,\
   /atg/commerce/custsvc/ui/tables/gift/search/EventDate,\
rowsPerPage=10
gridHeight=450px
gridInstanceId=atg.commerce.csr.gift.giftlistInstance
gridPath=/atg/commerce/custsvc/ui/tables/gift/search/GiftlistGrid
gridWidgetId=atg_commerce_csr_customer_gift_GiftlistTable
progressNodeId=atg_commerce_csr_gift_GiftlistGridStatus
searchFormId=atg_commerce_csr_giftlistSearchForm
dataModelPage=/atg/commerce/custsvc/ui/tables/gift/search/GiftsistDataPage
gridPage=/atg/commerce/custsvc/ui/tables/gift/search/GiftlistGridPage
```

The extended configuration property, GiftListSearchExtended.properties, does not contain links to a JSP file.

Gift List Form Handlers

The following form handlers can be used when customizing gift list forms:

CSRGiftlistFormHandler

This form handler sets the messaging, environment and profile tools, as well as populates the gift lists with the gift items. This form handler extends the `GifListFormHandler`.

Form Handler	**`atg.commerce.csr.gifts.CSRGiftlistFormHandler`**
Components	`/atg/commerce/custsvc/gifts/CSRGiftlistFormHandler.properties`

GiftlistTableFormHandler

This form handler populates the customer gift list grid on the customer view page.

Form Handler	**`atg.commerce.csr.gifts.GiftlistTableFormHandler`**
Components	`/atg/commerce/custsvc/gifts/GiftlistTableFormHandler.properties`

This form handler uses the following properties:

- `DoOwnerSearch` – By default, this property is set to `true` indicating that the search query will always return results based on the gift list owner property
- `DoSiteFilterSearch` – By default, this property is set to `true` so when multisite is enabled the search query will only return results that have a valid site. By default, any gift list with a site that has been disabled will still be returned as part of the search results
- `IncludeDisabledSites` – Can be set to `false` to omit Disabled Sites from the search results

For more information on Commerce Service Center specific Gift List form handlers, refer to the *ATG API Reference for Commerce Service Center*.

Auditing Gift Lists

Audit events are created for gift and wish lists whenever lists are created or deleted, or whenever items are added or removed from a list. Audit events are also created whenever quantities are changed within a list. The audit events are stored in the `csr_giftlist_event` table. For information on this table, refer to the Appendix A, *Commerce Service Center Database Tables* (page 157).

Recorded events are configured in the `GiftItemEventRecorder.properties` file. By default, the properties recorded are `giftlistId`, `eventName`, `catalogRefId`, `OldQuantity` and `newQuantity`.

For information on configuring auditing, refer to the *Reporting and Logging* (page 141) section.

8 Issuing Returns, Exchanges and Refunds

The following sections provide information on the returns and exchange processes and calculators that are specific to Commerce Service Center.

Commerce Service Center extends the Commerce Returns features for generating returns. Refer to the *ATG Commerce Programming Guide* for detailed information on the Returns process.

Commerce Service Center-Specific Return Components

Return Form Handler

This form handler contains Commerce Service Center-specific UI handlers and extensions of the `BaseReturnFormHandler`.

Form Handler	**`atg.commerce.csr.returns.ReturnFormHandler`**
Components	`/atg/commerce/custsvc/returns/ReturnFormHandler.properties`

This form handler uses the following properties and components:

Components	Description
`handleReceiveReturnItems`	This handler receives items that are being returned.
`postConfirmReturn`	Sends email notifications, associates the active ticket with the replacement order and creates a new active order in the Commerce Service Center environment.

For more information on this form handler, refer to the *ATG API Reference for Commerce Service Center*.

Is Item Returnable Droplet Servlet Bean

This servlet bean extends the Commerce `IsReturnable` servlet bean and identifies is an item is returnable in Commerce Service Center.

Class	**`atg.commerce.csr.order.IsItemReturnable`**
Components	`/atg/commerce/cstsvc/order/IsItemReturnable.properties`

Order Is Returnable Droplet Servlet Bean

This servlet bean determines in an order is in a state where a return or exchange can occur in Commerce Service Center.

Class	**`atg.commerce.csr.order.OrderIsReturnable`**
Components	`/atg/commerce/cstsvc/order/OrderIsReturnable.properties`

The `OrderIsReturnable` servlet bean contains the following:

Input Parameters

- `order`– The ID of the order

Oparams

- `true` – Rendered if there is a return in progress
- `false` – Rendered if there is no return in progress
- `error` – Rendered if there is an error

Prepare Replacement Order Pipeline

This pipeline chain, which is defined in Commerce, is executed when an exchange is confirmed. It prepares the replacement order for submission and includes the following processors:

Processor	Description
`ExecuteValidateForCheckoutChain`	Executes a Commerce pipeline that validates that the order is ready for checkout.
`RemoveEmptyShippingGroups`	Removes shipping groups without relationships.
`RemoveEmptyPaymentGroups`	Removes payment groups without relationships.

Processor	Description
`CreateImplicitRelationships`	Creates relationships for orders that have a single shipping or payment group with no relationships.
`SetPaymentGroupAmount`	Sets the amount of the payment groups based of the group's relationships.
`AuthorizePayment`	Authorizes payment groups in the order.
`SetSalesChannel`	Sets the sales channel of the replacement order based on the CSC extra parameter. Identifies where the order was submit. The `BaseFormHandler` sets the sales channel extra parameter Map to its configured value. The `ReturnFormHandler` for CSC will set the sales channel to `contactCenter`.
`UpdateReplacementOrderStateOnConfirm`	Updates the replacement order state to "pending customer return".
`AddOrderToRepository`	Adds the replacement order to the repository.

Working with Exchange Orders

When an agent initiates a return and exchange, three working orders are used to calculate the correct refund amount and exchange item prices. For detailed information on these working orders, refer to the *ATG Commerce Programming Guide*.

Exchange Calculators

Commerce Service Center adds four calculators which are used for pricing exchange orders. There are two item pricing post calculators in `/atg/commerce/custsvc/pricing/calculators`:

- `ExchangePromotionEvaluationUpdateCalculator` – This calculator executes the process that updates the Promotion Evaluation Order. This process copies the exchange items into the Promotion Evaluation Order and then re-prices it. This results in exchange item pricing in accordance to other items still owned by the customer and any item promotions that may apply
- `ExchangeItemAdjustmentCalculator` – This calculator copies pricing information for each exchange item from the Promotion Evaluation Order to the exchange order

There are two order pricing post calculators in `/atg/commerce/custsvc/pricing/calculators`:

- `ExchangeOrderAdjustmentCalculator` – This calculator applies order promotions to the exchange order that are applied to the Promotion Evaluation Order. The promotion adjustment value is based on the order discount share value applied to the exchange items
- `ExchangeOrderDiscountCalculator` – This calculator applies manual adjustments that are applied to the Promotion Evaluation Order to the exchange order. The manual adjustment value is based on the manual adjustment share value applied to the exchange items

The `ItemPricingEngine` configuration contains the `ExchangeItemAdjustmentCalculator` item pricing calculator. This calculator adjusts the `ItemPriceInfo` of an item in the Exchange Order based on pricing of the corresponding item in the PEO and copies that information from the PEO to the `ItemPriceInfo` created for the exchange item. This includes the amount, all `PricingAdjustments` and `DetailedItemPricingInfos`. This prices the item in the Exchange Order with the exact pricing it received in the PEO, including any item level promotions that may have been applied.

If manual adjustments are applied to the Promotion Evaluation Order, they are applied to the exchange order by the `OrderPricingEngine` post calculator `OrderAdjustmentCalculator`. This calculator implements `priceOrder` to trigger the calculation or the manual adjustment value map when pricing an Exchange Order, then determines if manual adjustments should be applied to the exchange order. If necessary, the calculator overrides the `adjustOrderSubTotal` to adjust the order subtotal based on the pre-calculated value of each manual adjustment.

Tiered Pricing and Exchanges

When you work with exchanges, tiered pricing produces the correct results, but exhibits the following behavior as a result of how the cost of exchange items is determined.

For example, in an order with a quantity of 7 items within the same tier levels, the original item would be priced at 2 at 50, 3 at 40, 2 at 30. In an exchange of 1 for 1, the expected refund would be $30 and the exchange order item would be $30, with a net no charge. However, because of the methodology used in pricing the exchange items in the promotion evaluation order, the exchange order item is priced at 50, and the refund is calculated at $50. This results in the expected net no charge for the 1 item in the exchange order. Additionally, if exchange order quantity increases to 2, the total would be $100 and the refund would be $70, with an expected net change of $30 for the additional item.

For additional information on pricing calculators, refer to the *Pricing Overview* section of the *ATG Commerce Programming Guide*.

Applying Promotions to Exchange Orders

No new promotions will be applied to an exchange order. New promotions include all promotions available to the shopper for their next new order checkout. Promotions that were applied to the original purchase can still be applied to the exchange order. For example, a promotion that is “buy X and get Y free” will still be applied if Y is exchanged for another Y.

However, once a promotion no longer applies to the order it cannot be applied on subsequent exchanges. For example, a promotion that is “buy X and get Y free” will not be applied if Y is exchanged for Z, and then Z is exchanged for Y. The system calculates and saves the changes in the promotion value within the `ReturnRequest`. This value is also saved in the repository with the return request item.

9 Working with Scheduled Orders

Scheduled Orders is a feature available in Commerce that sets up automated recurring orders. For additional information on scheduled orders, refer to the *ATG Commerce Programming Guide*. Agents can assist customers with setting up scheduled orders.

Configuring Scheduled Orders

A scheduled order is comprised of a *template order* and a *schedule*. A template order contains all of the order information but is not submitted. A template can be associated with one or many schedules. Based on the schedule(s) associated with the template, the template object is cloned to create an order object, which is then submitted.

Agents can create scheduled order templates using the new order checkout process, or by copying an existing order. Scheduled orders can be reviewed from the customer's profile. Agents can also display all schedules for a specific template, and submit an ad hoc order from a scheduled order.

Agents set up scheduled orders based on a daily, weekly, or monthly schedule. The schedule may be *periodic*, which is based on an interval of time between each scheduled run, or *calendar-based*, which is based on specific day to run. For periodic schedules, Commerce Service Center allows an agent to choose intervals based on a number of days or weeks. For calendar-based schedules, Commerce Service Center allows an agent to specify days of the week or dates of the month. A schedule also defines a start end date, which can be used to limit the period of time that the schedule is active.

Enabling and Disabling Scheduled Orders

Scheduled orders are enabled by default. To disable the scheduled orders feature, modify the `CSRConfigurator` component, setting `usingScheduledOrders` to `false`.

For example, to disable scheduled orders:

```
$class=atg.commerce.csr.util.CSRConfigurator
scope=global

usingScheduledOrders=false
```

The `usingScheduledOrders` option controls the availability of the scheduling options at the end of checkout, the scheduled orders panel on the Profile View screen and the scheduled order view screen for template orders.

Configuring Price Lists

Note: If your site uses price lists, you must configure the use of price lists as outlined in Defining the Default Price List (page 91) later in this document.

The following section describes configuration that affects the pricing of scheduled orders when using price lists in combination with scheduled orders. This configuration affects the pricing of scheduled orders when:

- Pricing a new scheduled order for submission

  This occurs in Commerce via the `ScheduledOrderService` and `ScheduledOrderTools` components when scheduled orders are automatically created and submitted based on their pre-defined schedules.

- Pricing a scheduled order template for view in Commerce Service Center

  Scheduled order templates are priced whenever an agent selects one for viewing on the scheduled order view. This ensures that the agent always views the current day pricing when looking at a scheduled order template. The result of this pricing operation is not saved to the repository, and is only temporary for the view.

- Pricing a new scheduled order instance when the agent clicks Submit Now in Commerce Service Center

  The Submit Now process allows an agent to manually create and submit an order from a scheduled order template.

Working with Scheduled Order Templates

Commerce Service Center always uses the `CustomerPricingModels` `PricingModelHolder` for pricing order templates. Scheduled order templates are priced this way to provide the agent with pricing information that reflects the current day pricing. This gives the agent an accurate representation of the order total if an order were to be submitted from the template on that day.

Modifying scheduled order templates also uses clone editing and therefore, pricing changes are not saved to the template unless the post create checkout process is completed for the template.

Scheduled Order Templates are also unique because they are priced before being viewed by an agent. For example, if an agent views the contents of a template in read-only mode before actually selecting the template to work on, the view would reflect current day pricing for the template. However, any pricing changes made for viewing the template are not persisted to the repository; they are only performed to provide the agent with current day pricing in the view.

ScheduledOrderTools

This Commerce component automatically generates and submits scheduled orders based on their pre-defined schedules and contains configuration specific to the pricing of the scheduled orders they have submitted.

The property `useOrderPriceListsFirst` controls how `ScheduledOrderTools` determines the correct price lists to use. Set by default to `false`, the price list assignment is determined by looking at the price lists assigned to the customer profile that owns the scheduled order. Because price list assignments are an optional feature in Commerce, if price list assignments are not implemented, the price list will come from the `PriceListManager` default configuration of `defaultPriceList` and `defaultSalesPricelist`. For additional information on Commerce price lists, refer to the *Using Price Lists* section of the *ATG Commerce Programming Guide*.

If `ScheduledOrderTools.isUseOrderPriceListsFirst` is set to `true`, it will first attempt to get the price lists by extracting them from the template. Pricing information must be stored with the scheduled order template when it is created to be able to extract price lists from order templates. It may not be possible to extract both price lists from the template. If the items in the order are all on sale, the list price cannot be determined. Conversely, if no items are on sale, the sale price list cannot be determined. If the price list cannot be determined from the template, the price list assignment is obtained from the customer profile that owns the scheduled order. If price list assignments are not implemented, the price list will come from the `PriceListManager` default configuration of `defaultPriceList` and `defaultSalesPricelist`.

If running in a multisite environment, the `getPriceListFromSite` method determines the price list for the site associated with the order. The `useSitePriceLists` property retrieves price lists from the site using the scheduled order and defaults to `true`.

It is important to note that there are situations that may occur when using either of these methods:

- When a site changes customer price list assignments based on storefront selection, it is possible for an agent to set up a scheduled order based on the current assignments for a customer. If the customer switches storefronts, any new scheduled orders will be priced using the new assignments
- If the price list on a customer's profile is updated, and Commerce Service Center has been configured to obtain price lists from the order items first, the change will not be applied to any scheduled orders

Customizing Scheduled Orders

Once you have enabled your site to use scheduled orders the following customizations can be made. For additional information on customization of scheduled orders, refer to the *ATG Commerce Programming Guide*.

Scheduled Order Form Handler

The `CSRScheduledOrderFormHandler` creates two variables, the `CalendarSchedule` variable used for specific days, and the `PeriodicSchedule` that is used for interval schedules. Additionally, the `CSRScheduledOrderFormHandler` is responsible for creating and updating the scheduled order repository item based on the input provided from the agent.

In addition to these two schedule types, Commerce Service Center provides an additional `N number of intervals` option used with `PeriodicSchedule` that an agent can use to limit the number of scheduled intervals. This schedule option automatically calculates the schedule's end date by calculating the schedule's `nextRunTime` *n* number of times starting from the specified start time.

Note: The number of intervals is used only to calculate the end date and will not be available when a schedule is viewed or updated.

`CSRScheduledOrderFormHandler` contains the following configurable values for controlling the default selections on the schedule creation form.

Property	Description
`defaultScheduleType`	Identifies the default schedule type. Possible values are `Calendar` or `Interval`.
`defaultDaysOption`	Controls which `CalendarSchedule` day selection is the default. Possible values are `allDays`, `selectedDays`, or `selectedDates`.

Property	Description
`defaultMonthsOption`	Controls which `CalendarSchedule` month selection is the default. Possible values are `allMonths` or `selectedMonths`.
`defaultEndDateOption`	Controls which Schedule End Date option is the default. Possible values are none, `endBy` or `endAfterOccurrences`.
`defaultOccurrencesOption`	Controls which `CalendarSchedule` occurrences option is the default. Possible values are `allOcurrences` or `selectedOccurrences`.
`defaultInterval`	Controls the default interval for an `PeriodicSchedule`.
`defaultIntervalOption`	Controls the default interval option for `PeriodicSchedules`. Possible values are days or weeks.

Displaying Scheduling Information

Commerce Service Center provides an extension to the Commerce `ScheduledOrderInfo` servlet bean that provides additional output parameters that describe the contents of the schedule. Refer to the *ATG Commerce Programming Guide* for information on `ScheduledOrderInfo`.

`CSCScheduledOrderInfo` has the following additional output parameters:

- `readableDays`
- `readableDates`
- `readableHours`
- `readableMinutes`
- `readableMonths`
- `readableOccurrences`

Scheduled Order Components

The following components are all members of the `atg.commerce.csr.order.scheduled` class located in the `/atg/commerce/custsvc/order/scheduled/` directory and are used for scheduled orders:

Components	Description
`CSRScheduledOrderTools`	Contains various helper APIs.
`CSRScheduledOrderFormHandler`	Form handler used to create and update schedules.
`DuplicateAndSubmit`	Form handler used by the Submit Now feature.
`IsScheduledOrder`	Servlet bean used to determine if an order is a scheduled order template.

Components	Description
`ScheduledOrderTableFormHandler`	Form handler used to generate the content displayed in the customer's scheduled orders panel.
`SchedulesTableFormHandler`	Form handler used to generate the content displayed in the scheduled order view's schedules panel.
`SubmittedOrdersTableFormHandler`	Form handler used to generate the content displayed in the scheduled order view's submitted orders panel.
`ActivateSchedule`	Form handler used in the `LoadAndExecuteAction` class to activate a schedule from the schedule order view.
`DeactivateSchedule`	Form handler used in the `LoadAndExecuteAction` class to deactivate a schedule from the schedule order view.

Scheduled Orders Pipeline Additions

The following are additions to the `commercepipeline.xml` that defines various purchase process pipelines used by Commerce Service Center.

Pipeline Chain	Description
`initScheduledOrderEdit`	This chain is executed by the `CloneEditManager` to prepare for updating a previously created scheduled order template.
`reconcileScheduledOrder`	This chain is executed by the `CloneEditManager` to reconcile changes made to a previously created scheduled order template.
`processTemplateOrder`	This chain is executed by the `CSRCommitOrderFormHandler` when the Schedule option is used at the end of new order checkout. It validates that a new order is ready to be saved as a scheduled order template.

10 Issuing Promotions

Commerce Service Center allows agents to access a number of promotion methods. Promotions such a are displayed to the agent in the promotions browser. For detailed information on configuring promotions, refer to the *ATG Commerce Programming Guide*.

Providing Promotions Browser Access

The Promotions Browser allows an agent to view all of the promotions that are available, or may become available, to a user. Agents can view promotions that have been granted and the discount amount applied by each promotion. Additionally, agents can search through promotions and grant a promotion manually.

For an agent to access the Promotions Browser, they must have the `cmcPromotionsP` and `commerce-custsvs-browse-promotions-privilege` access rights. These access rights are provided in the `csrOrders` role. For additional information on configuring access rights, refer to .

Customizing Gift with Purchase Promotions

In addition to the standard promotions that are available to customers, Commerce Service Center supports the Commerce Gift with Purchase promotion. This promotion can automatically add a free item to the shopping cart when the order qualifies for a promotion. The gifts can be defined as an SKU, product or a single selection from a category or content group.

When an agent adds an item to an order that qualifies it for a free gift, the free gift is automatically added to the cart with the total price of the item set to $0.00. **Note:** The gift is added automatically only if the gift is defined as an SKU or product. When a gift is part of a multiple selection, it cannot be automatically added to the cart.

If an agent adds an item to the order that qualifies it for a gift that must be selected from multiple choices, a gift selector becomes available. Once the selected gift has been added to the cart, a Change Gift option is available if the agent needs to make any changes.

Gift With Purchase Page Fragments

The following three `PageFragment` components are used to render the Gift with Purchase links on the cart page and the gift selection pop-up. The page fragments are stored in the `/DCS-CSR-UI/config/`

`atg/commerce/custsvc/ui/fragments/gwp` directory:

- `ChangeGiftListPageFragment` – This fragment renders the Change Gift link next to an item when it contains a gift that has multiple choices
- `SelectGiftLinkPageFragment` – This fragment renders a gift selection link for any gift that has not yet been added to the order. The fragment uses the `GiftWithPurchaseSelectionsDroplet` to obtain the `GiftWithPurchaseSelection` objects for the current order. Any `GiftWithPurchaseSelection` that is found with a `quantityMissingFromOrder` greater than 0 will have a link rendered for it. If the `giftType` is `category` or `contentGroup`, a selection link is rendered for each unit that is missing from the order. If the `giftType` is `sku` or `product`, a single selection is rendered that adds the entire missing quantity to the cart
- `SelectGiftPopupPageFragment` – The `SelectGiftPopupPageFragment` is used to render the gift selection pop-up. It requires several input parameters for determining the gift choices and initializing the `GiftWithPurchaseFormHandler` values:
  - `itemId` – Indicates which commerce item references the changed gift selection. The value is optionally provided to the `GiftWithPurchaseFormHandler`
  - `promotionId` – The ID of the promotion. Required by the `GiftWithPurchaseFormHandler`
  - `quantity` – Required by the `GiftWithPurhcaseFormHandler`, this provides the quantity of the gift selection
  - `giftHashCode` – Required by the `GiftWithPurchaseFormHandler`, this identifies the promotion hash code
  - `giftType` – Identifies the gift type, and is required for the `GiftWithPurchaseSelectionChoicesDroplet`
  - `giftDetail` – Provides gift detail information and is required for the `GiftWithPurchaseSelectionChoicesDroplet`

For additional information on the `GiftWithPurchaseSelectionChoicesDroplet`, refer to the *ATG Commerce Programming Guide*.

Should the agent proceed with checkout without first selecting eligible gifts for the order, the `postMoveToPutchaseInfo` method in the `CSRCartModifierFormHandler` will issue a message to inform the agent that the order qualifies for the gifts that are missing from the order.

The Commerce `GWPManager` queries all Gift with Purchase selections for the order. If any of them have a `quantityMissingFromOrder` greater than 0, an informational message is posted for the agent, such as "This order qualifies for a free gift that has not yet been selected."

For additional information on Gift with Purchase configuration, refer to the Extending Objects for Cloning (page 134) section.

Returns and Exchanges of Gifts with Purchases

If an item that qualified a promotion is returned, by default, the gift will not be automatically removed from the order. It will be left in the order and re-priced at full value, offsetting the value of the refund. For additional information on how the returns and exchange pricing engine works, refer to the *Pricing in Commerce Service Center* (page 99) section.

To provide gift selection links in the cart view of the change order, Commerce Service Center generates `GiftWithPurchaseSelection` objects based on the gifts being returned and then stores them in the

ReturnRequest. Commerce Service Center extends the GiftWithPruchaseSelectionsDroplet to return the ReturnRequest selection objects when rendering the selection links in the cart view of the exchange order.

Each time the cart view is rendered, the generated GiftWithPurhcaseSelection objects are filtered by the ReturnManager generateFilteredReturnRequestSelections API to ensure that selection links are not rendered for gifts that may have already been selected in the exchange order. For detailed information on gift with purchase exchange orders, refer to the Working with Exchange Orders (page 59) section.

Reconciling Gift with Purchase Orders

The Gift with Purchase feature extends the CommerceItem object to store Gift with Purchase meta data within repository markers.

Commerce Service Center's clone editing feature supports the cloning and reconciliation of the Gift with Purchase Commerce item markers when an order containing gifts is modified by an agent.

The SubPropertyHandler interface defines the requirements for implementing a CloneEditHandler for objects referenced by a sub-property of another parent object. The parent object is the CommerceItem and the markers are the objects referenced by the sub-property gwpMarkers.

The subclass CollectionSubPropertyEditHandler extends the CollectionEditHandler class and implements a SubPropertyHandler for handling collection sub-properties.

The CommerceItemMarkerEditHandler class extends the CollectionSubPropertyHandler class to handle a collection property that contains Commerce item markers. A component of this type, which is named /atg/commerce/custsvc/order/edit/CommerceItemGWPMarkerHandler, is defined by Commerce Service Center to handle clone and reconciliation of Gift with Purchase Commerce item markers.

The following is an example of the CommerceItemGWPMarkerHandler:

```
$class=atg.commerce.order.edit.CommerceItemMarkerEditHandler

keyPropertyName=repositoryId
sortPropertyName=creationDate
collectionPropertyName=gwpMarkers
cloneEditManager=/atg/commerce/custsvc/order/edit/CloneEditManager

propertiesToCopyOnUpdate=\
           gwpCommerceItemMarker=key,,value,,data,,targetedQuantity,,
                                 automaticQuantity,,selectedQuantity
```

The CollectionEditHandler implementation supports the integration of multiple SubPropertyHandler components. These components are configured on the CollectionEditHandler through the subPropertyHandlers property.

The /atg/commerce/custsvc/order/edit/CommerceItemHandler contains a CollectionEditHandler that handles Commerce items and references the SubPropertyHandler for GWP markers:

```
$class=atg.commerce.csr.order.edit.CSRCommerceItemEditHandler
subPropertyHandlers=/atg/commerce/custsvc/order/edit/CommerceItemMarkerHandler,\
                    /atg/commerce/custsvc/order/edit/CommerceItemGWPMarkerHandler
```

Note that the CommerceItemHandler references two subPropertyHandlers; one for each marker property defined on a CommerceItem.

Note: The Commerce Service Center Copy Order and Submit Now features exclude Commerce item markers when making a copy of the order. This means that no Gift with Purchase related information, such as which items are manually selected gifts, will be carried over to the new order.

11 Using Order Approvals

Agents can provide appeasements for customers by providing price overrides and manual adjustments in orders.

Configuring Order Approval

When the order is submitted, the total of the order's price overrides and manual adjustments can be verified against a preset appeasement limit. If the appeasement surpasses the limit, an approval is required to continue the order submission.

The approval system for Commerce Service Center is based on the ATG platform approval process, and provides the process for submitting new orders that have appeasement discounts. The system consists of processes that determine:

- If an approval is required
- Saves the data required to complete the action when approved
- Interrupts the action
- Approves or rejects the action after review

For detailed information on the ATG platform approval process, refer to the *ATG Commerce Programming Guide*.

The Commerce Service Center approval system generates instances of `atg.commerce.csr.approvals.Approval` class to create an Order Approval object that is verified by the `/atg/commerce/custsvc/approvals/order/OrderApprovalHandler`, which is the `ApprovalHandler` that processes order approvals. The pipeline `CheckIfOrderApprovalRequired` processor calls into the `ApprovalHandler` to determine if the approval is required.

The `atg.commerce.csr.approvals.ApprovalsManager` class calls other classes that create new approval objects and repository items, determine if an approval is required, save or load approval objects to or from their corresponding repository items, query the approvals repository and approve or reject an approval item. The `ApprovalManager` also maintains the state of the approval, which can be PENDING, APPROVED, or REJECTED. The `ApprovalsManager` uses the `ApprovalType` property of the Approval object to determine which `ApprovalHandler` to use. The `ApprovalHandler` determines if approval is required. If approval is required, the object will be saved in the approvals repository. Refer to the Commerce Service Center Order Approval Tables (page 158) for information on the repository files.

Agent access to the approvals process is granted through the `cmcApprovals` access right, which is incorporated into the `CSR-Manager` role. For additional information on access rights, refer to *Setting Up Internal Access Control* (page 77).

The `OrderApprovalHandler` checks to see if the agent placing the order does not require approval by checking that they have the `cmcApprovals` access right. Then the handler checks that an appeasement limit exists by first checking the agent's appeasement for the currency in question and then checking the default appeasement limit for the currency in question. Finally the handler totals the price overrides and manual adjustments and compares them with the appeasement limit.

After the order has been sent through the approval process, an e-mail is generated using the e-mail address associated with the approval record and e-mail templates `OrderApprovalAcceptedTemplate` and `OrderApprovalRejectedTemplate`. The `sendConfirmationEmailmethod` of `CSRAgentTools` creates the e-mail. If there is no e-mail address associated with the approval record, e-mail will not be sent.

Enabling the Order Approval Process

The use of appeasement limits is configured with the `/atg/commerce/custsvc/util/CSRConfigurator` component `usingOrderApprovals` property. The default setting of this property, which is `true`, enables the order approval process. You can disable the process by setting the property to `false`.

Setting Global Appeasement Limits

An agent's profile defines their appeasement limit. If an agent does not have a specified appeasement limit available in their profile, Commerce Service Center will use the global default appeasement limit for that currency. The default appeasement limit is configured in the `/atg/commerce/custsvc/util/CSRConfigurator` component with the `defaultAppeasementLimits` property. The appeasement limit is based upon the currency of the order, such as:

```
defaultAppeasementLimits=USD=500.00
```

To allow an agent to have an unlimited appeasement limit, set the `defaultAppeasementLImits` to a value of `-1`, or update the agent's access rights to contain the `cmcApprovals` access right.

Modifying Individual Appeasement Limits

User profiles contain an extension that allows you to set appeasement limits in multiple values. Limits can be set per currency. If no appeasement limit is set, the limit will default to the appeasement limits set in the `CSRConfigurator` component.

By default, the appeasement limit for individuals is set to `-1` in US Dollars. This removes all appeasement verification.

To Modify an Agent's Appeasement Limit

1. Open the BCC Home page > Access Control > Users panel. Select the agent.

2. Scroll to the General pane.

3. Click the Add button to add the appeasement limits. Enter the appeasement limit key and value.

4. Save the agent's profile.

Providing Approval Authorization

The `cmcApprovals` access right controls the agent’s ability to see the Order Approvals panel. The `cmcApprovals` access right is listed in the `CSR-Manager` role. Agents that are granted the `cmcApprovals` access right or the `CSR-Manager` role will be able to review orders that need approval, as well as the ability to approve and reject orders. Appeasement limits that are applied against agents with these rights will be ignored.

Servlet Beans and Form Handlers for Approving Orders

The following servlet beans and form handlers are used for approving orders.

Find Pending Approvals Droplet Servlet Bean

Returns all approval items for the given `approvalType` with a state of PENDING. If no `approvalType` is provided, all `approvalTypes` are returned.

Class	**`atg.commerce.csr.approvals.FindPendingApprovalsDroplet`**
Components	`/atg/commerce/custsvc/approvals/FindPendingApprovalsDroplet`

The `FindPendingApprovalsDroplet` contains the following:

Input Parameters:

- `approvalType` – Looks for items that are pending
- `elementName` – The name of the return element

Oparams:

- `output` – if an order has any appeasements
- `empty` – if there are no appeasements

Output Parameters

- `element` – the total appeasements for the order

Get Total Approvals For State Droplet Servlet Bean

This servlet bean queries the approvals repository and returns the total number of results for a given type of approval.

Class	**`atg.commerce.csr.approvals.GetTotalApprovalsForStateDroplet`**
Components	`/atg/commerce/custsvc/approvals/order/GetTotalPendingOrderApprovalsDroplet`

The `GetTotalPendingOrderApprovalsDroplet` contains the following:

Input Parameters

- `elementName` – The name of the return element

Oparams

- `output` – if an order has any appeasements
- `empty` – if there are no appeasements

Output Parameters

- `element` – the total appeasements for the order

Is Order Pending Approval Droplet Servlet Bean

This servlet bean determines if an order is currently pending approval and returns the approval ID that is associated with the order.

The servlet bean is used on confirmation pages to determine if an order emerged from the submission process in a PENDING approval state. It is also used on the Order View page to determine if “Approve” or “Reject” elements should appear.

Class	**`atg.commerce.csr.approvals.order.IsOrderPendingApprovalDroplet`**
Components	`/atg/commerce/custsvc/approvals/order/IsOrderPendingApprovalDroplet`

The `IsOrderPendingApprovalDroplet` contains the following:

Input Parameters

- `orderId`
- `elementName` – The name of the return element

Oparams

- `true` – if an approval is found for the order
- `false` – If no approval is found for the order
- `error` – If an error occurred while looking up the approval

Output Parameters

- `element` – If found, the approval item for the order.

The following is an example of the droplet:

```
<dsp:droplet name="/atg/commerce/custsvc/approvals/order/IsOrderPendingApproval">
  <dsp:param name="orderId" value="orderId">
  <dsp:oparam name="output">
    <dsp:valueof param="element"/>
  </dsp:oparam>
```

```
</dsp:droplet>
```

Get Total Order Appeasements Droplet Servlet Bean

This servlet bean determines if an order has any appeasements applied to it, and returns the total number of appeasements if any are found.

Class	**`atg.commerce.csr.order.GetTotalOrderAppeasementsDroplet`**
Components	`/atg/commerce/custsvc/order/ GetTotalAppeasementsForOrderDroplet.properties`

The `GetTotalOrderAppeasementDroplet` contains the following:

Input Parameters

- `order` – The ID of the order
- `elementName` – The name of the return element

Oparams

- `output` – if an order has any appeasements
- `empty` – if there are no appeasements

Output Parameters

- `element` – the total appeasements for the order

For example:

```
<dsp:droplet name="/atg/commerce/custsvc/approvals/order/
      GetTotalOrderAppeasements">
  <dsp:param name="order" value="order">
  <dsp:oparam name="output">
    <dsp:valueof param="element"/>
  </dsp:oparam>
</dsp:droplet>
```

Order Approval Form Handler

The `OrderApprovalFormHandler` generates the order approval within the UI and extends the `EnvironmentChangeFormHandler`. This form handler provides handlers for both approving and rejecting an approval request. It also contains e-mail templates for approvals and rejections.

Form Handler	**`atg.commerce.csr.approvals.order.OrderApprovalFormHandler`**
Components	`/atg/commerce/custsvc/approvals/order/OrderApprovalFormHandler`

Approval Repository Query Form Handler

The `ApprovalRepositoryQueryFormHandler` finds approvals of a specified state and type. It handles the search request from the UI and returns a list of search results, as well as handles paging of the search results. This form handler extends the `RepositoryQueryTableFormHandler`.

Form Handler	**`atg.commerce.csr.approvals.order.ApprovalRepositoryQueryFormHandler`**
Components	`/atg/commerce/custsvc/approvals/order/ApprovalRepositoryQueryFormHandler`

Update Order Approval Customer Email Form Handler

The `UpdateOrderApprovalCustomerEmailFormHandler` updates the customer e-mail address on an order approval object.

Form Handler	**`atg.commerce.csr.approvals.order.UpdateOrderApprovalCustomerEmailFormHandler`**
Components	`/atg/commerce/custsvc/approvals/order/UpdateOrderApprovalCustomerEmailFormHandler`

For more information on Commerce Service Center specific Gift List form handlers, refer to the *ATG API Reference for Commerce Service Center*.

12 Setting Up Internal Access Control

When Commerce Service Center is installed, it is preconfigured with various access rights, global roles, and access controllers. These elements are used to restrict access to certain pages in Commerce Service Center.

Access Control Overview

Access control in Commerce Service Center is enables you to provide different levels of access to the application depending on the job that agents are performing. When you set up access control, you are controlling who can access which part of the Commerce Service Center application. There are several elements connected with access control:

- Users – Each individual, or agent, is an internal user. Each internal user is given access to the Commerce Service Center application by an administrator. Users are members of an organization that has associated roles, allowing them access to the Commerce Service Center application

  Your customers are defined as external users. **Note:** Configuring customer access to different sites or shops is done by creating segments, as described in the *ATG Personalization Programming Guide* and/or by creating scenarios and targeters, as described in the *ATG Personalization Guide for Business Users*

- Organizations – You can group users by making them members of an organization. For example, you could set up organizations that are based on different geographic areas, or on different business units within your company. For additional information on creating and working with Organizations, refer to the *ATG Business Control Center Administration and Development Guide*

- Roles – Roles determine the set of rights that are assigned to a user. Once you have created roles, you associate rights and users to the role. There are Global Roles, which work across all users and organizations, and there are organization-based roles, that usually correspond to tasks or a specific function

- Access Rights – Access rights define tasks that the user can perform, for example, creating a new customer profile. Access rights are granted to a role. Commerce Service Center comes preconfigured with access rights that have been designed based on specific CSR agent activities. A subset of these rights is assigned to each Commerce Service Center role, and you assign the appropriate roles to agents to give them the access rights they need

For further information on access rights for agents, refer to theAppendix B, *Commerce Service Center Access Rights* (page 169)

Default Internal User Access Control Configuration

The default internal user access control configuration provided with Commerce Service Center includes a number of access controllers. The access controllers are Nucleus components that are added to your installation when you install Commerce Service Center. The access rights and roles are repository data that you import into your database (from supplied XML files) as a configuration step after you install Commerce Service Center. For more information about installing and configuring Commerce Service Center, see *Installing and Configuring the Commerce Service Center Server* (page 9).

Commerce Service Center Roles

Commerce Service Center comes preconfigured with four global roles for setting access rights granted to agents. The Commerce Service Center roles use these roles as template to simplify their configuration:

- `csrTicketing` – Includes the access rights necessary to access the Commerce Service Center application and to use the ticketing components
- `csrOrders` – Includes the access rights needed to create and modify orders. In addition, this role includes `csrTicketing` as a template role, so all ticketing access rights are included. This role also includes the access rights necessary to grant promotions using the Promotions Browser. This role does not provide authorization to create new customer profiles, or the other rights specific to `csrProfiles`
- `csrProfiles` – Includes the access rights needed to create and modify customer profiles, set up and review gift lists and respond to customers via e-mail. In addition, this role includes `csrTicketing` as a template role, so all ticketing access rights are included. This role does not include the scheduling access rights that are included in `csrOrders`
- `csrManager` – Includes all of the rights within both `csrOrders` (and thus `csrTicketing`) and `csrProfiles`, as well as allowing price overrides

For a list of all access rights for each role, and a description of the access right, refer toAppendix B, *Commerce Service Center Access Rights* (page 169).

When you create an agent's profile in the Internal User Profile Repository, you assign the agent a role that corresponds to the tasks the agent is authorized to perform. For example, a typical agent may be able to create and modify orders, but only a manager can override prices and issue credits.

Access Controllers

The following table summarizes the Commerce Service Center access controllers that can be configured using the security properties files:

ID	Name
`issueCredit`	`commerce-custsvc-issue-credit-privilege`
`adjustPrice`	`commerce-custsvc-adjust-price-privilege`
`Adjust Password`	`commerce-custsvc-change-customer-password-privilege`
`viewProfile`	`commerce-custsvc-view-profiles-privilege`

ID	Name
`Return`	`commerce-custsvc-create-return-privilege`
`Record Return`	`commerce-custsvc-record-merchandise—return-privilege`
`Change Password`	`commerce-custsvc-change-customer-password-privilege`
`Create Order`	`commerce-custsvc-create-orders-privilege`
`Create Profile`	`commerce-custsvc-create-profiles-privilege`
`Edit Order`	`commerce-custsvc-edit-orders-privilege`
`Edit Profile`	`commerce-custsvc-edit-profiles-privilege`
`Issue Credit`	`commerce-custsvc-issue-credit-privilege`
`browsePromotions`	`Commerce-custsvc-browse-promotions-privilege`

For more information regarding access rights and controllers, refer to the *ATG Personalization Programming Guide*.

Creating New Roles

If you have requirements that none of the existing roles meet, you can create new roles.

To create new roles:

1. Open the Agent Server BCC Home page > Access Control page.
2. Select Roles to display Role Folders.
3. Select the location to store the role, or create a new role folder by accessing the + action menu.
4. Use the + action menu to create the new role.
5. Enter the name and description of the new role.
6. In the Access Rights pane add existing access rights, or create new access rights by specifying Direct Access Rights or incorporating the access rights from existing roles by using the Template Role field.
7. Once you are finished, click Save to save your settings.

For additional information on creating roles, refer to the *ATG Business Control Center Administration and Development Guide*.

Creating Agent Profiles

By default, Commerce Service Center is not preconfigured with any agent profiles. As part of setting up Commerce Service Center, you need to:

1. Create the necessary organization, organizational roles, global roles and access rights. Refer to the *ATG Business Control Center Administration and Development Guide* for information on creating these objects.
2. Create a profile in the Internal User Profile Repository for each agent.
3. Assign each profile the necessary roles.

Note: When working in a multisite environment, you cannot set security to limit an agent per site.

Note: This account should be added only to the database that is provided with the ATG platform for evaluation purposes. You should not include it in a production database, as this is a serious security risk.

For additional information on creating users, refer to the *ATG Business Control Center User's Guide*.

1. Using the Agent Server BCC Home Page, select Access Control.
2. Click the + action icon to create a new Internal User.
3. Enter the agent information.
4. Click the Orgs & Roles tab, set the agent's organization and roles. You can also set site restrictions and user preferences. Once you have finished, click Create.

   For example, to create an agent who can work with existing orders but not create new customers, you would select the `csrOrders` template role. This role contains all of the order access permissions needed to see orders and work with orders, yet it does not contain the create customer profile permissions. **Note:** Both the `csrOrders` and `csrProfiles` roles include the `csrTicketing` role, which allows access to the application, the log in screen, etc. For a list of all of the access rights, refer to theAppendix B, *Commerce Service Center Access Rights* (page 169)
5. You can select the user to review their rights. If you need to add additional rights, you must create a corresponding role, and then add the role to the user profile.

Creating a New Agent Role

You can create new roles based on the requirements of your agents. You can manually create a new role and add the Direct Access Rights individually to the role, or you can copy an existing role and modify the copy. The following steps use the example of copying an existing role and then creating an access right for a Summer-only agent who does not have the ability to modify orders or perform scheduled orders, exchanges or returns.

1. Open the Dynamo Server Admin and select the `/atg/userprofiling/InternalProfileRepositoy`. Select the Roles component.
2. To copy one of the roles, use the <print-item> command. For example, to copy the `csrManager` role:

   ```
   <print-item item-descriptor="role" id="csrManager"/>
   ```

   This returns the following:

   ```
   <add-item item-descriptor="role" id="csrManager">
   <set-property name="description"><![CDATA[Role for the CSR
   manager]]></set-property>
   <set-property name="accessRights"><![CDATA[cmcConfirmReturnP,
   cmcOrderReturnsP,issueCredit,cmcProductViewP,cmcRelatedTicketsP,
   cmcApprovals,cmcConfirmNewScheduleP,cmcCustomerCreateP,adjustPrice,
   cmcOrderHistoryP,cmcPromotionsP,cmcMoreCatalogsP,cmcShoppingCartP,
   cmcScheduleCreateP,cmcRefundTypeP,cmcGiftlistsViewP,cmcReturnItemsP,
   cmcOrderSearchP,cmcOrderResultsP,cmcSubmittedOrdersP,
   ```

```
cmcCustomerResultsP,cmcConfirmOrderP,cmcAddProductByIdP,
cmcConfirmExchangeP,cmcBillingP,cmcCompleteOrderP,cmcReturnsHistoryP,
commerceTab,cmcProductCatalogSearchP,cmcShippingMethodP,cmcSchedulesP,
cmcExchangeSummaryP,cmcReturnSummaryP,cmcPurchasedItemsHistoryP,
cmcGiftlistViewPurchaseModeP,cmcRelatedOrdersP,
cmcExistingScheduledOrderP,cmcCrossSellP,cmcCustomerInfoP,
cmcTicketHistoryP,cmcCompleteExchangeP,cmcScheduleUpdateP,
cmcMorePriceListsP,cmcConfirmUpdateScheduleP,cmcReturnDetailsP,
cmcGiftlistSearchP,cmcExistingOrderP,cmcShippingAddressP,
RespondComposeMessagePanel,cmcCompleteReturnP,respondTab,
cmcScheduledOrdersP,cmcPurchaseHistoryP,cmcMultisiteSelectionPickerP,
cmcProductCatalogBrowseP,cmcCustomerSearchP,cmcCustomerP]]>
</set-property>
<!-- rdonly <set-property name="version"><![CDATA[2]]></set-property> -->
<set-property name="templateRoles"><![CDATA[csrOrders,csrProfiles]]>
</set-property>
<set-property name="name"><![CDATA[CSR-Manager]]></set-property>
</add-item>
```

3. Copy and paste this into the Run XML Operations Tag field.
4. Rename the ID, Description and Name. For example, change all references to `CSR Manager` to `CSR Summer`. The renamed references are shown as bold in the following step.
5. Remove any access rights you do not want. For example, modify the existing `CSRManager` role by removing rights for approvals, exchanges, returns and any other rights that the agent should not have. For example:

```
<add-item item-descriptor="role" id="csrSummer">
<set-property name="description"><![CDATA[Role for the CSR
Summer Staff]]></set-property>
<set-property name="accessRights"><![CDATA[cmcConfirmReturnP,
cmcProductViewP,cmcRelatedTicketsP,cmcCustomerCreateP,
cmcOrderHistoryP,cmcPromotionsP,cmcMoreCatalogsP,cmcShoppingCartP,
cmcGiftlistsViewP,cmcOrderSearchP,cmcOrderResultsP,
cmcSubmittedOrdersP,cmcCustomerResultsP,cmcConfirmOrderP,
cmcAddProductByIdP,cmcBillingP,cmcCompleteOrderP,cmcReturnsHistoryP,
commerceTab,cmcProductCatalogSearchP,cmcShippingMethodP,
cmcPurchasedItemsHistoryP,cmcGiftlistViewPurchaseModeP,
cmcRelatedOrdersP,cmcCrossSellP,cmcCustomerInfoP,cmcTicketHistoryP,
cmcMorePriceListsP,cmcGiftlistSearchP,cmcExistingOrderP,
cmcShippingAddressP,cmcPurchaseHistoryP,cmcMultisiteSelectionPickerP,
cmcProductCatalogBrowseP,cmcCustomerSearchP,cmcCustomerP]]>
</set-property>
<!-- rdonly <set-property name="version"><![CDATA[2]]></set-property> -->
<set-property name="templateRoles"><![CDATA[csrOrders,csrProfiles]]>
</set-property>
<set-property name="name"><![CDATA[CSR-Summer]]></set-property>
</add-item>
```

6. Click Enter to add the new role. Once the role is added, you can modify or edit it using the BCC Home page.
7. Log into the BCC Home page on the Agent Server.
8. Using the Access Control screen, select the new role.
9. Make any modifications to the role that you require and add it to the agents' profile.

Default Roles

The `csrOrder` role contains the access rights from `csrTicketing`, as well as four additional roles. These roles allow the agent to browse promotions and work with Scheduled Orders. To create an agent that does not have the ability to edit orders or to perform returns, exchanges, or appeasements, the role might contain the following rights:

Direct Access Right	Allows User to Access the…
`browsePromotions`	Customer's available promotions
`CustomerInformationPanel`	Information panel on Customer page
`CustomerOrderHistoryPanel`	Order History panel on Customer page
`CustomerResultsPanel`	Customer Search Results panel on Customer page
`CustomerSearchPanel`	Customer Search Panel on Customer page
`CustomerTicketHistoryPanel`	Ticket History panel on Customer page
`GlobalPanel`	General Preferences panel in Preferences
`HelpfulOpenByIDPanel`	Open Solution panel in Helpful Panels
`HelpfulRecentTicketsPanel`	Recent Tickets Panel in Helpful Panels
`HelpfulTicketHistoryPanel`	Case History panel in Helpful Panels
`HelpfulTicketSummary`	Case Summary panel in Helpful Panels
`TasksAllTicketPanel`	All Tickets panel in Tickets page
`TasksMyTicketPanel`	My Tickets panel in Tickets page
`TicketActivityPanel`	Ticket History panel in Tickets page
`TicketsCustomerInformationPanel`	Customer Information panel in Tickets page
`TicketsResultsPanel`	Ticket Search Results panel in Tickets page
`TicketsSearchPanel`	Ticket Search panel in Tickets page
`TicketsSummaryPanel`	Ticket Summary panel in Tickets page
`cmcAddProductByIdP`	Add Product by ID panel
`cmcBillingP`	Billing panel
`cmcCompleteExchangeP`	Complete Exchange panel
`cmcCompleteOrderP`	Complete Order panel
`cmcConfirmNewScheduleP`	Confirm New Schedule panel

Direct Access Right	Allows User to Access the…
`cmcConfirmOrderP`	Confirm Order panel
`cmcConfirmUpdateScheduleP`	Confirm Updated Schedule panel
`cmcCrossSellP`	Cross Sell panel
`cmcCustomerCraeteP`	Customer Create panel
`cmcCustomerInfoP`	Customer Information panel
`cmcCustomerP`	Customer panel
`cmcCustomerResultsP`	Customer Results panel
`cmcCustomerSearchP`	Customer Search panel
`cmcExistingOrderP`	Exiting Order panel
`cmcExistingScheduledOrderP`	Existing Schedule Order panel
`cmcMoreCatalogsP`	More Catalogs panel
`cmcMorePriceListsP`	More Price Lists panel
`cmcMultisiteSelectionPickerP`	Multisite Site picker
`cmcOrderHistoryP`	Order History panel
`cmcOrderReulstsP`	Order Results panel
`cmcOrderSearchP`	Order Search panel
`cmcProductCatalogBrowserP`	Catalog Browser panel
`cmcProductCatalogSearchP`	Search Catalog panel
`cmcProductViewP`	Product View panel
`cmcPromotionsP`	Promotions panel
`cmcPurchaseHistoryP`	Purchase History panel
`cmcPurchaseItemsHisotyrP`	Purchased Items History panel
`cmcRelatedOrdersP`	Related Orders panel
`cmcRelatedTicketsP`	Related Tickets panel
`cmcScheduleCreateP`	Create Schedule Order panel
`cmcScheduledOrdersP`	Scheduled Orders panel
`cmcScheduleUpdateP`	Update Scheduled Orders panel

Direct Access Right	**Allows User to Access the…**
`cmcScheduelsP`	Scheduled panel
`cmcShippingAddressP`	Shipping Address Panel
`cmcShippingMethodP`	Shipping Method panel
`cmcShoppingCartP`	Shopping Cart panel
`cmcSubmittedOrdersP`	Submitted Orders panel
`cmcTicketHistoryP`	Ticket History panel
`commerceDesignTab`	Commerce Design page
`commerceTab`	Commerce page
`customersTab`	Customers page
`tasksTab`	Tasks page
`ticketsTab`	Tickets page
`workspaceLogin`	Workspace Login screen

Customizing the Default Landing Page

You can create a default landing page for the agents by configuring `/DCS-CSR/install/data/csrOptions` file. Change the `defaultValue` property of the `GlobalOption` item-descriptor with the `id="AgentUserDefaultHomeTab`" to the desired value.

For example, change the `<set-property name="defaultValue" value="commerceTab"/>` to `<set-property name="defaultValue" value="orderTab"/>` to change the default landing page to the order page.

13 Configuring E-mail

Customizing E-Mail

Note: These components are modified using the ACC.

Configuring E-mail Notifications

When an agent creates a new customer profile, a password is automatically generated for the customer's account, and a notification is typically sent to the customer's e-mail address. The `/atg/svc/agent/UI/Formhandlers/CustomerProfileFormHandler` component uses the following properties to manage new account e-mail notifications:

Property Name	Description
`newAccountTemplateEmailInfo`	The `atg/svc/email/DefaultTemplateEmailInfo` component creates the e-mail message.
`persistNewAccountEmails`	If `true`, new account e-mails are persisted in the customer's profile before they are sent. Default is `false`.
`sendNewAccountEmailInSeparateThread`	If it is a new account, e-mail is sent in a separate thread. The default is `true`.
`sendNewAccountEmails`	If `true`, an e-mail containing the new account login and password is generated and sent to the customer. Default is `true`.

For information about configuring a `TemplateEmailInfo` component, see the *ATG Personalization Programming Guide*.

Automatically Sending E-mail for Orders

To configure an automatic e-mail to be sent when an order is created or updated, use the `/atg/commerce/custsvc/order/CommitOrderFormHandler.properties`:

```
# Confirmation Email Settings
```

```
autoSendEmail=false
autoSendNewOrderEmail=false
autoSendUpdateOrderEmail=false
```

By default, these properties are set to `false`. To enable an automatic e-mail whenever a new order is created, the `autoSendEmail` and `autoSendNewOrderEmail` properties should be set to `true`. To enable an automatic e-mail whenever an order is updated, the `autoSendEmail` and `autoSendUpdateOrderEmail` properties should be set to `true`.

Configuring New Passwords

When an agent generates a new password for a customer in the Commerce Service Center, the customer's profile must have a valid e-mail address so the new password can be e-mailed to the customer. The `/atg/svc/agent/UI/Formhandlers/CustomerProfileFormHandler` component uses the following properties to manage passwords:

Property Name	Description
`persistResetPasswordEmails`	If `true`, new passwords are persisted in the customer's profile before they are sent. Default is `false`.
`resetPasswordtemplateEmailInfo`	The `resetPasswordTemplateEmailInfo` component that creates the e-mail message. Default is `ForgotEmailTemplateInfo` (in `/atg/svc/email/`).
`sendResetPasswordEmailInSeparateThread`	If `true`, the reset password e-mail is sent in a separate thread. The default is `true`.
`sendResetPasswordEmails`	If `true`, an e-mail containing the new password is generated and sent to the customer. Default is `true`.

For information about configuring a `TemplateEmailInfo` component, see the *ATG Personalization Programming Guide*.

Configuring Order Confirmation E-Mails

It is possible to configure e-mail confirmations that occur once an order has been placed. The `/atg/commerce/custsvc/util/CSRAgentTools` component `configurationEmailMap` allows you to configure order confirmation e-mails.

Property Name	Description
`APPROVAL_ACCEPTED`	`/atg/commerce/custsvc/profile/ApprovalAcceptedEmailInfo`
`APPROVAL_REJECTED`	`/atg/commerce/custsvc/profile/ApprovalRejectedEmailInfo`

Property Name	Description
`NEW_ORDER`	`/atg/commerce/custsvc/profile/NewOrderEmailInfo`
`ORDER_EXCHANGE`	`/atg/commerce/custsvc/profile/OrderExchangeEmailInfo`
`ORDER_RETURN`	`/atg/commerce/custsvc/profile/OrderReturnEmailInfo`
`ORDER_UPDATE`	`/atg/commerce/custsvc/profile/OrderUpdateEmailInfo`
`SCHEDULED_ORDER _UPDATE`	`/atg/commerce/custsvc/profile/ ScheduledOrderUpdateEmailInfo`
`SCHEDULED_ORDER_ADD`	`/atg/commerce/custsvc/profile/ScheduledOrderAddEmailInfo`

A new e-mail information component can be associated with any of the keys to override the default e-mail component.

If you are working in a multisite environment, you can configure the site in the order objects to include site information in different parts of the configuration e-mail. For example, you could enter all of the `From` field values into a branch based on the order's submit site. Additionally, you could add a property to the `From` site object and reference that property based on the order's submit site.

Configuring E-Mail Templates

The `TemplateEmailSender` service is responsible for sending template-based e-mail. Using `TemplateEmailInfo`, it renders the page specified using the `templateURL`. For information on working with e-mail notifications and templates, refer to the *ATG Personalization Programming Guide*.

Commerce Service Center provides a default e-mail template implementation for each of the following actions:

- New Account Registration
- New Account Registration Following Checkout
- Password Reset
- New Order Confirmation
- Order Modification Confirmation
- Return Confirmation
- Exchange Confirmation

The e-mail information component can be reconfigured to use a different `templateURL` property, which points to a different JSP page specifying a different template.

To change an e-mail template, modify the `templateURL` property of any of the following `TemplateEmailInfo` components:

- `/atg/svc/email/NewAccountTemplateEmailInfo`
- `/atg/commerce/custsvc/profile/NewAccountEmailInfo`

- `/atg/svc/email/ResetPasswordTemplateEmailInfo`
- `/atg/commerce/custsvc/profile/NewOrderEmailInfo`
- `/atg/commerce/custsvc/profile/OrderExchangeEmailInfo`
- `/atg/commerce/custsvc/profile/OrderReturnEmailInfo`
- `/atg/commerce/custsvc/profile/OrderUpdateEmailInfo`

14 Using Catalogs and Price Lists

The following section provides information on configuring catalogs and price lists in Commerce Service Center.

Configuring Current Catalog and Price Lists

Commerce Service Center contains two global environment objects used by the application to manage catalogs and orders: current catalog and current price list. The current settings of these objects can be viewed and changed in the Commerce page submenu.

The current catalog and price list used are specified in the agent’s profile. The catalog and price list are initialized using information from the current customer. When the agent logs in, the current catalog and price list are initialized based on the `CSRConfigurator.defaultCatalogId` and `PriceListManager.defaultPriceList` settings. However, when the agent selects a new customer, the current catalog and price list settings automatically adjust to whatever has been assigned to the customer.

Using the Current Catalog

The current catalog determines which catalog is used by the agent when browsing and searching the catalog. If using a multisite environment, the catalogs are site-aware.

The value of the current catalog can be set by the following actions:

- When the agent first logs in, a new, transient profile is automatically loaded as the active customer. The current catalog is set from the new profile’s assigned catalog. If the profile does not have an assigned catalog, the current catalog is set based on the `CSRConfigurator defaultCatalogId` property
- When the agent selects a customer from the profile repository to be the active customer, the current catalog is set based on the selected profile’s assigned catalog
- If multisite has been enabled, when an agent selects a site, the default catalog associated with that site will become the current catalog
- The agent can manually select a different catalog using the UI. Once an agent has explicitly selected a catalog, that catalog remains active even if a new customer with a different assigned catalog is selected from the repository. Starting a new call will reset the current catalog and remove the agent’s explicitly selected catalog

Using the Current Price List

The current price list is only used when the price lists are employed. It determines which price list is used for catalog and order pricing operations. It is also used to determine the pricing locale.

The `CSRConfigurtor` has a property called `usingPriceLists,` which must be set to `true` when the application is using price lists.

The values of the current price list can be set by the following actions:

- When the agent first logs in, a new, transient profile is automatically loaded as the active customer. The current price list is set from the new profile's assigned price list. If the profile doesn't have an assigned price list, the current price list is set based on the `PriceListManager defaultPriceList`
- When the agent selects a customer from the profile repository to be the active customer, the current price list is set based on the selected profile's assigned price list
- When the agent selects an order from the repository to be the active order, the current price list is set based on the first commerce item in the order with a price list assigned. A commerce item's price info contains a reference to the price list used to the price it
- If multisite is enabled, when the agent selects a site, the price list will change to the price list for that site
- The agent can manually select a different price list using the UI. Once an agent has explicitly selected a price list, that price list remains active even if a new customer with a different assigned price list is selected from the repository. Starting a new call will also reset the current price list and remove the agent's explicitly selected price list

CSREnvironmentTools

This component contains the API for gaining access to the current catalog and current price list.

Class	**`atg.commerce.csr.environment.CSREnvironmentTools`**
Component	`/atg/commerce/custsvc/environment/CSREnvironmentTools`

CSRAgentTools

This component contains the API for generating a parameter map for pricing operations that contains the active price list.

Class	**`atg.commerce.csr.util.CSRAgentTools`**
Component	`/atg/commerce/custsvc/util/CSRAgentTools`

ChangeCatalogAndPricelist Form Handler

This form handler component is used to manually change the catalog and price list through the UI.

Form Handler	**`atg.svc.agent.environment.EnvironmentChangeFormHandler`**
Component	`/atg/commerce/custsvc/environment/ChangeCatalogAndPricelist`

Defining the Default Catalog

The catalog displayed to the user is derived from the catalog assigned in their profile. If there is no catalog assigned, the `defaultCatalogId` property of the `/atg/commerce/custsvc/util/CSRConfigurator` component specifies the catalog to use for anonymous shoppers.

Defining the Default Price List

If your site uses price lists, you must configure the following:

- The `/atg/commerce/custsvc/util/CSRConfigurator` component property `usingPriceLists` must be set to `true`. This enables the use of price lists
- Each customer must be assigned a price list. The `defaultPriceListId` property of the `/atg/commerce/pricing/pricelists/PriceListManager` component specifies the price list to use for anonymous shoppers

Setting the Pricing Locale

The active customer pricing locale drives the currency and currency codes of prices used in Commerce Service Center. The active pricing locale may be customized based on the supported currencies of the store. This component is modified using `CRSAgentTools`.

Class	**`atg.commerce.csr.util.CSRAgentTools`**
Component	`/atg/commerce/custsvc/util/CSRAgentTools`

The `getActiveCustomerPricingLocale()` method provides access to the pricing locale and will return the price list locale (if one is specified and price lists are being used). If the price list locale is unavailable, the locale stored in the customer profile is used. Failing that, if `/atg/commerce/custsvc/util/CSRAgentTools.useRequestLocale=true`, the current request locale is used. If all of these attributes are unavailable, the value of the `/atg/commerce/pricing/PricingTools.defaultLocale` is used.

Specifying Quick Access Catalogs and Price Lists

Quick access catalogs and price lists are available as sub-navigational menus from the catalog and price list menus, allowing agents quick access to the most commonly used catalogs and price lists.

Quick access catalogs and price lists are configured by creating `SiteOption` items in the `/atg/svc/option/OptionRepository` and specifying the catalog or price list IDs of the catalogs or price lists that should appear in the quick-access menus. The `defaultValue` property specifies a list of catalog or price list IDs for the catalogs or price lists to display.

The following is an example of a quick-access catalog site:

```
<add-item item-descriptor="SiteOption" id="QuickAccessCatalogs">
  <set-property name="name" value="QuickAccessCatalogs"/>
  <set-property name="accessRight" value="serviceAdminDefaultRight"/>
  <set-property name="multiValued" value="true"/>
  <set-property name="dataType" value="String"/>
  <set-property name="defaultValue" value="catalog10002,masterCatalog"/>
</add-item>
```

The following is an example of a quick-access price list site:

```
<add-item item-descriptor="SiteOption" id="QuickAccessPriceLists">
  <set-property name="name" value="QuickAccessPriceLists"/>
  <set-property name="accessRight" value="serviceAdminDefaultRight"/>
  <set-property name="multiValued" value="true"/>
  <set-property name="dataType" value="String"/>
  <set-property name="defaultValue" value="listPrices,plist20003,salePrices"/>
</add-item>
```

For additional information on configuring catalogs and price lists, refer to the *ATG Commerce Programming Guide*.

15 Understanding Environment Monitoring

An environment is the collective state of an agent's current working context. Environment monitoring defines the integration of applications within an environment and coordinates changes to global environment objects.

Overview of Environment Monitoring

Commerce Service Center exposes the following global objects to the environment management system:

- Current Order – The order the agent is currently working on
- Current Catalog – The catalog context the agent is current working in
- Current Pricelist – The currently selected price list. This price list is used for pricing operations on the catalog, cart and checkout pages
- Current Site – The currently selected site. Agent actions, such as change order, or start new call will maintain the appropriate state per site
- Site versus Catalog Mapping – In the product catalog page, by default the site's default catalog is used for browsing and searching. If the agent chooses a different catalog for the site, the currently selected catalog is used for browse and product searches. The mapping is cleared when the agent switches from the current sharing group to another sharing group

Environment Monitoring Components

The following components are used to configure environment monitoring.

CSREnvironmentTools

Class	**`atg.commerce.csr.environment.CSREnvironmentTools`**
Component	`/atg/commerce/custsvc/environment/CSREnvironmentTools`

This component provides the core API for applying changes to the Commerce Service Center environment objects. It also provides the API for accessing the Commerce Service Center-managed environment objects.

The `CSRENvironmentTools` component contains the `doSitesShare` method, which checks to see if the current site and the desired new site share the same site group. If both sites share the same site group, the method returns true.

CSREnvironmentMonitor

Class	**`atg.commerce.csr.environment.CSREnvironmentMonitor`**
Component	`/atg/commerce/custsvc/environment/CSREnvironmentMonitor`

This component detects changes, generates warnings and applies changes for Commerce Service Center-managed objects.

The `CSREnviornmentMonitor` component contains the following methods:

- `getUsersCatalog()` calls the core Commerce API to find the right catalog
- `getUsersPriceList()` calls the core Commerce API to find the right price list
- `getUsersSalePriceList()` calls the core Commerce API to find the right sales pricelist
- `generateDependentDetailsForActiveSiteChange()` will change the order, catalog, pricelist and sale price list if the site is changed
- `generateSiteChangeForOrderChange()` generates the site change for the order change. If the order contains the submitted state, the submitted site is loaded as the current site

CSREnvironmentConstants

Class	`atg.commerce.csr.environment.CSREnvironmentConstants`

This static class exposes the environment change keys and their input parameters defined by Commerce Service Center. The `orderId` parameter is an example of an input parameter name required to execute the `changeOrder` change.

EnvironmentChangeFormHandler, ChangeOrder

These components are used to perform Commerce Service Center environment changes from UI gestures.

Form Handler	**`atg.svc.agent.environment.EnvironmentChangeFormHandler`** **`atg.commerce.csr.environment.ChangeOrder`**

Components	`/atg/commerce/custsvc/environment/ChangeCatalogAndPriceList`
	`/atg/commerce/custsvc/environment/ChangeOrder`
	`/atg/commerce/custsvc/environment/CreateNewOrder`
	`/atg/svc/agent/environment/ChangeSiteFormHandler`

The existing `EnvironmentChangeFormHandler` is used to change a site. The following is the sample form for changing sites:

```
<svc-ui: frameworkUrl var="errorURL" panelStacks=""/>
  <svc-ui: frameworkUrl var="successfulSiteChangeURL"
    panelStacks="globalPanels,cmcShoppingCartPS" tab="commerceTab"/>
  <%/*form used to change a site */%>
  <dsp: form style="display: none" id="atg_commerce_csr_loadExistingSiteForm"
    formid="atg_commerce_csr_loadExistingSiteForm">
  <dsp: input type="hidden" name="errorURL" value="${errorURL}"
    bean="/atg/svc/agent/environment/ChangeSiteFormHandler.errorURL" />
  <dsp: input type="hidden" name="successURL" value="${successfulSiteChangeURL}"
    bean="/atg/svc/agent/environment/ChangeSiteFormHandler.successURL" />
  <dsp: input type="hidden" name="siteId" bean="/atg/svc/agent/environment/
    ChangeSiteFormHandler.inputParameters.siteId" value=""/>
  <dsp: input type="hidden" priority="-10"
    bean="/atg/svc/agent/environment/ChangeSiteFormHandler.changeEnvironment"
    value=""/>
  </dsp: form>
```

The following is the `ChangeSiteFormHandler.properties` configuration for changing sites:

```
/atg/svc/agent/environment/ChangeSiteFormHandler.properties
$class=atg.svc.agent.environment.EnvironmentChangeFormHandler
$scope=request
  environmentTools=/atg/svc/agent/environment/EnvironmentTools
  transactionManager=/atg/dynamo/transaction/TransactionManager
  messageTools=/atg/web/messaging/MessageTools
  confirmURL=include/environment/confirm.jsp
  confirmPromptURL=include/environment/changePrompt.jsp
  environmentChangeKey=changeSite
  environmentChangeState=/atg/svc/agent/environment/EnvironmentChangeState
  doWarnings=true
  doTicketDispositionPrompt=true
  applicationName^=/atg/svc/agent/environment/
    EnvironmentTools.agentApplicationName
  ticketingTools=/atg/svc/agent/ticketing/TicketingTools
```

For additional information on this form handler, refer to the *ATG API Reference for Commerce Service Center*.

Ticket Disposition Monitoring

Commerce Service Center includes the ability to automatically handle the disposition of the current ticket whenever it changes. Whenever a different ticket will be loaded into the environment because of a change request, one of several disposition options must be selected for the ticket being replaced (e.g. the current ticket). For detailed information on ticket disposition, refer to the *ATG Ticketing User Guide*.

Commerce Service Center uses the `/atg/commerce/custsvc/ticketing/CSRTicketDispositionMonitor`, which it adds to the ticketing manager's `ticketDispositionMonitors` property. This ticketing monitor's `shouldDiscard()` and `shouldDiscardImmediately()` methods both return `false` if a ticket contains any of the activity types listed in the monitor's `nonDiscardableActivityTypes` property.

This property is an array of activity type names. By default, `nonDiscardableActivityTypes` is set to a list of all Commerce activity types, which means that `CSRTicketDispositionMonitor` will not allow a ticket to be discarded if there are any Commerce activities associated with the ticket. You can change this behavior by setting the value of this property to a different list of activity types. For detailed information on discarding tickets, refer to the *Discarding Tickets* section of the *ATG Ticketing User Guide*.

EnvironmentTools

When you are creating a multiple site environment, changing one environment object may affect another environment object. For example, if you change an order, it may change the site, price list, sale price list, user, ticket or other environment-managed objects.

In a multisite setting, environment management recognizes site support and manages site-related effects. Because basic site environment management support is added to Service Center, functions like pricing model holder initialization, site dependent changes or order dependent changes are supported using the `/atg/svc/agent/environment/EnvironmentTools` API.

The `atg.svc.agent.environment.EnvironmentTools` API is configured as follows:

```
/atg/svc/agent/environment/EnvironmentTools.java
  public void addChangeSiteDetail(String pNewSiteId,
EnvironmentChangeState
    pEnvironmentChangeState) throws EnvironmentChangeDetailConflict,
    EnvironmentException {}
  public Site getCurrentSite() throws EnvironmentException {}
  public void setCurrentSite(Site pSite) throws EnvironmentException {}
```

The following methods are used to get or set site information:

- `addChangeSiteDetail` – This method adds site change details
- `getCurrentSite` – This method obtains the current site from the `CurrentSiteHolder`
- `setCurrentSite` – This method sets the current site in the `CurrentSiteHolder`

The `ServiceEnvironmentMonitor` API also sets site information:

```
/atg/svc/agent/environment/ServiceEnvironmentMonitor.java
  protected void
    generateInitialChangesForChangeSite(EnvironmentChangeState
    pEnvironmentChangeState)  throws EnvironmentException {}
```

```
public void revertSiteChangeDetail(EnvironmentChangeDetail
  pEnvironmentChangeDetail,EnvironmentChangeState
  pEnvironmentChangeState) {}
public void applySiteChangeDetail(EnvironmentChangeDetail
  pEnvironmentChangeDetail, EnvironmentChangeState
  pEnvironmentChangeState)throws EnvironmentException {}
```

The `ServiceEnvironmentMonitor` uses the following methods:

- `generateInitialChangesForChangeSite` – generates initial changes for the change site
- `applySiteChangeDetail` – This method applies the site change information in the `CurrentSiteHolder`
- `revertSiteChangeDetail` – This method reverts site details to the old site details that are set in the `CurrentSiteHolder`

Environment Management and Site Context

Sites are defined with a Site ID. Environment management monitors the site environment using the `CurrentSiteHolder` object. Whenever the site is changed, the `CurrentSiteHolder` is updated with the current site.

For each request `siteContext` is set using the `CurrentSiteContextRuleFilter` component in `atg.multisite.SiteContextPipelineServlet`. If `CurrentSiteContextRuleFilter` returns a `siteId`, then the `siteId` is set as the site context for the entire request. The `CurrentSiteContextRuleFilter` gets the current site from the `CurrentSiteHolder`.

If the `CurrentSiteHolder.currentSite` is empty, then the site context is not set. If the site context is not set, then environment management picks the catalog and price list based on the current user. If the site context is set, then environment management picks the catalog and price list based on the current site and user.

16 Pricing in Commerce Service Center

The following section outlines how pricing is performed in Commerce Service Center and describes the components and API that can be used or modified to change pricing behavior. For information on core Commerce pricing, refer to the *ATG Commerce Programming Guide*.

Loading Orders and Pricing

When a modifiable order is loaded into the current global context, or selected by an agent, it is priced by Commerce Service Center using the `CSREnvironmentMonitoring` component. The entire order is priced using `PricingConstants.OP_REPRICE_ORDER_TOTAL` as the pricing operation.

Only modifiable orders are priced when selected for the global context, however, it is possible to configure which orders can be modified by Commerce Service Center based on the state of the order. `CSRAgentTools` contains API and configurable properties for determining which orders are considered modifiable by state.

Determining if Orders are Modifiable

The `CSRAgentTools.isOrderModifiable(Order pOrder)` method determines if an order is modifiable by comparing the order's current state to the configured values and can be extended if necessary.

An order is not modifiable when:

- Any items in the order have been shipped
- Any payment groups in the order are in a non-modifiable state
- The order is in a non-modifiable order state

The following example displays the configurable properties in `/atg/commerce/custsvc/util/CSRAgentTools`:

```
#both nonModifiableOrderStates and nonModifiablePaymentGroupStates are
#used to determine a non-modifiable order.

#states that indicate an order cannot be modified
nonModifiableOrderStates=REMOVED,\
             QUOTED,\
             NO_PENDING_ACTION,\
             PENDING_REMOVE,\
             PENDING_CUSTOMER_RETURN,\
                AGENT_REJECTED

nonModifiablePaymentGroupStates^=\
/atg/commerce/order/PaymentGroupManager.nonModifiablePaymentGroupStates=REMOVED,\
                                                                      SETTLED
```

Determining if Orders are Submitted

The `CSRAgentTools.isOrderSubmitted` method determines if an order is in a submitted state. Submitted order states are configured using the `submittedOrderStates` property of `CSRAgentTools`:

```
submitedOrderStates=SUBMITTED, PROCESSING, PENDING MERCHANT ACTION
```

Commerce Service Center uses this API to:

- Determine which pipelines to use for clone edit initialization and reconciliation. All orders that the API considers submitted use pipelines configured for the submitted state
- Determine if the original order prices should be used to price the order
- Present dynamic page content when working on a submitted order. For example, disabling the Promotion Browser when working on a submitted order

Price Lists and Pricing

When configured to use price lists, Commerce Service Center provides two global context price lists: one for list prices and another for sale prices. Commerce Service Center takes advantage of a Commerce feature that allows the price lists to be passed into the pricing engine through the extra parameter map to override the default behavior that determines which price list to use. The following Commerce `PricingTools` API is used for pricing orders. For additional information, refer to the *ATG Commerce Programming Guide*. Note the `Map` parameter that is included on the API:

```
public OrderPriceInfo priceOrderTotal(Order pOrder,
                           PricingModelHolder pPricingModels,
                           Locale pLocale,
                           RepositoryItem pProfile,
                           Map pExtraParameters)
```

The `CSRAgentTools` API generates a map that contains the IDs of the current global context price list selections. This API is used by Commerce Service Center to generate the extra parameter map for pricing operations. Commerce Service Center has also extended `createRepriceParameterMap` of all the `PurchaseProcessFormHandlers` to call this API:

```
public Map addPriceListParameter(Map pExtraParameters)
```

The `CSRAgentTools` API is used for pricing an order that will call `addPriceListParameter` as part of the process:

```
public void repriceOrder(String pRepricingOperation, Order pOrder,
           PricingModelHolder pUserPricingModels,
           Locale pLocale, RepositoryItem pCustomerProfile,
           RepositoryItem pAgentProfile,Map pExtraParameters,
           PipelineErrorHandler pErrorHandler)
```

When loading an order into the global context, Commerce Service Center attempts to extract the correct price lists from the order itself. If the price lists can be determined from the order, they will load into the global context at the same time as the order.

`CSREnvironmentTools` contains the API for accessing the global context price list selections, setting new global context price list selections and extracting a price list from an order:

```
public RepositoryItem getCurrentPriceList()
public RepositoryItem getCurrentSalePriceList()
public void setCurrentPriceList(RepositoryItem pPriceList)
public void setCurrentSalePriceList(RepositoryItem pPriceList)
public RepositoryItem getListPriceListFromOrder(Order pOrder)
public RepositoryItem getSaleListPriceListFromOrder(Order pOrder)
```

For additional information on environment monitoring, refer to the *Understanding Environment Monitoring* (page 93) section.

Automatic Removal of Items

If Commerce Service Center is configured to use price lists, it will automatically remove items from the order that cannot be priced by the current global context price list. This avoids pricing errors when the order is first loaded. When loading an order into global context, Commerce Service Center attempts to determine the correct price lists (list price list and sale price list) to use by extracting them from the order being loaded. This helps eliminate the problem of removing items from an order because an incorrect price list is in global context when the order is loaded. Commerce Service Center automatically changes the global context price list selections to match those found in the order.

Promotions

Commerce Service Center defines two `PricingModelHolders` for pricing orders. Commerce Service Center decides which one to use based on the state of the order and each holder is initialized with promotions differently.

- `atg/commerce/custsvc/pricing/CustomerPricingModels` – this holder is used when pricing incomplete orders. It is initialized with the active customer’s current promotions
- `atg/commerce/custsvc/pricing/SubmittedOrderPricingModels` – this holder is used when pricing submitted orders. It is initialized with the order’s applied promotions

Incomplete Orders or Schedule Order Templates

Commerce Service Center uses the active customer's current promotions when pricing incomplete (e.g. orders not submitted) orders.

Submitted Orders

For orders that have already been submitted to fulfillment, Commerce Service Center uses the promotions that were originally applied to the order when it was submitted.

Note that this is not the same as the promotions that were available at the time the order was submitted. It only includes the promotions that were applied to the order when it was submitted. For example, if a buy-two-get-one-free promotion was available when the order was submitted but only one was purchased and therefore not applied to the order, that promotion will not be in the `SubmittedOrderPricingModels` when the order is modified after submission to add the second item.

Determining the Correct PricingModelHolder

`CSREnvironmentTools` contains an API that returns the correct pricing model holder for the order currently loaded into the global context. `CSREnvironmentMonitor` and all of the Commerce Service Center extensions to the `PurchaseProcessFormHandler` call this API to determine the correct pricing model holder for pricing operations.

```
public PricingModelHolder getCurrentOrderPricingModelHolder
```

Configuring Manual Pricing Adjustments

It is possible to adjust an order total by a fixed amount. Adjustments can either be a fixed increase (debit) or decrease (credit) to the order total. The details of a manual adjustment such as the amount, adjustment type and reason code are permanently stored within the database. For information on the corresponding Commerce API, refer to the *ATG Commerce Programming Guide*.

All manual adjustments are created as transient repository items. Subsequent processing in the `updateOrder` pipeline determines if the adjustments is permanently added to the repository based on the following rules.

- All adjustments created for submitted orders are unconditionally persisted to the repository. As configured by default, this is the only time Commerce and Commerce Service Center unconditionally saves adjustments to the repository
- All adjustments created for persistent, incomplete orders are conditionally saved based on configuration. Commerce Service Center uses a Boolean configuration setting, whose default value is `false`, that determines if adjustments should be saved immediately for persistent, incomplete orders

The Commerce `updateOrder` pipeline contains a processor that saves the manual adjustment items to the repository when appropriate. `saveManualAdjustment` executes the `atg/commerce/order/`

`processor/SaveManualAdjustments` processor with the following two processor configurations:

```
#
# The processor will save the manual adjustments to the repository for
# orders in these states, depending on the value of saveIncomplete
# saveForIncompleteOrders
#
incompleteStates^=/atg/commerce/order/OrderLookupService.incompleteStates
#
# The processor will save the manual adjustments to the repository for
# orders in the configured incomplete states if this property is true.
# Otherwise, the manual adjustments are not saved for incomplete orders.
#
saveIncompleteOrderAdjustments=false
```

Important: Be aware of a condition that may occur when saving incomplete manual adjustments. Manual adjustments are, by default, applied unconditionally. Once added to the order, manual adjustments affect the order's price despite the contents of the order. This is important if an incomplete order is saved with manual adjustments as subsequent changes to the order at checkout time will not change any adjustments that have been applied. For example, if an agent applies a $20 credit adjustment to an order with $100 merchandise and saves it in an incomplete state, the customer could return and remove $80 worth of merchandise from the order and checkout with a $0 total. As such, the processor is configured by default to not save manual adjustments for incomplete orders.

The `OrderAdjustmentCalculator` adjusts the order's subtotal based on the manual adjustments associated with the order. The `/atg/commerce/pricing/calculators/OrderAdjustmentCalculator` component contains the following configuration:

```
$class=atg.commerce.pricing.OrderAdjustmentCalculator
pricingTools=/atg/commerce/pricing/PricingTools
```

The adjustment calculator is added as a `postCalculator` in the `/atg/commerce/pricing/OrderPricingEngine`. As such, the calculator runs after the pre-calculators and all calculators associated with any promotions for the order. The configuration for the `OrderPricingEngine` is:

```
postCalculators+=\
      calculators/OrderAdjustmentCalculator
```

For additional information on pricing and calculators, refer to the Working with Exchange Orders (page 59) section and the *ATG Commerce Programming Guide*.

17 Working with Shipping and Payment Groups

Commerce Service Center supports, by default, three shipping group types: Hard Goods, Electronic and In Store Pickup. Commerce Service Center also supports, by default, the following payment group types: Gift Certificate, Credit Card, Store Credit, Pay In-Store and Cash.

You can customize shipping groups and payment groups to do things such as add custom fields or add support for a new shipping group type. For additional information on working with shipping and payment groups, refer to the *ATG Commerce Programming Guide*.

For detailed information on working with the Service Center UI and using page fragments, refer to the *ATG Service Center UI Programming Guide*.

Shipping Group Page Fragments

Shipping and payment group configuration is defined using the `atg.commerce.csr.order.CommerceTypeConfiguration` class. The class contains the following:

```
protected PageFragment mAddPageFragment;
protected PageFragment mEditPageFragment;
protected PageFragment mDisplayPageFragment;
protected PageFragment mImagePageFragment;
protected String mResourceBundle;
protected String mType;
protected String mAddPageFragmentTitleKey;
protected String mEditPageFragmentTitleKey;
protected String mImageHoverTextKey;
```

The `CommerceTypeConfiguration` class uses the following properties:

Property	Description
`addPageFragment`	Adds a specific shipping or payment group type.
`editPageFragment`	Edits a specific shipping or payment group type.

Property	Description
`displayPageFragment`	Displays a specific shipping or payment group type.
`imagePageFragment`	Displays an image used with a specific shipping or payment group type.
`resourceBundle`	The configuration used to obtain the resourced values.
`type`	The primary key used to identify the shipping or payment group type configuration. The key is used as a map key for the system-generated shipping or payment group configuration map, which locates the configuration type.
`addPageFragmentTitleKey`	Defines the text used in the `AddPageFragment` title.
`editPageFragmentTitleKey`	Defines the text used in the `EditPageFragment` title.
`imageHoverTextKey`	Defines the text displayed when a user hovers over the image defined in the `ImagePageFragment`.

Commerce Service Center provides the following default shipping group types: hard goods, electronic goods and in-store pickup. Shipping group types are configured using the `atg.commerce.csr.order.CommerceTypeConfiguration` class. The following information describes how to reconfigure the default shipping group types or to configure new types.

The shipping group configuration files and the shipping group page fragment configuration files are available at the following locations:

File	Location
Configuration File	`/DCS-CSR-UI/config/atg/commerce/custsvc/ui/`
Page Fragments	`/DCS-CSR-UI/j2ee-apps/DCS-CSR-UI/include/order/`
Display JSPs	`/DCS-CSR-UI/j2ee-apps/DCS-CSR-UI/include/order/`
Add and Edit JSPs	`/DCS-CSR-UI/j2ee-apps/DCS-CSR-UI/panels/order/shipping/`

Working with Shipping Group Page Fragments

To change how your shipping groups are displayed, you must modify the `displayPageFragment` property page fragment file. Each shipping group type defines three display values that appear on various pages. A display value consists of a title and content and is configured in the display page fragment JSP file, where the three display values are identified in the file as `value1`, `value2` and `status`.

You can customize the default display values for the existing shipping group types. However, if you create new shipping group types, each new shipping group type must have a defined set of display values.

The following display values are configured for the hard good shipping type:

Value	Title	Content	Example

`Value1`	Shipping Address	`[first] [last]`
		`[address 1]`
		`[address 2]`
		`[city], [state] [zip code]`
		`[country]`
		`[phone number]`
			Bob Smith
			119 Grand Street
			Apt. 1509
			Brooklyn, NY 10023
			USA
			212-555-4321

`Value2`	Shipping Method	`[shipping method]`	Ground
`Status`	Status	`[shipping group status]`	Ready to Ship

The following display details are configured for the electronic shipping group type:

Value	Title	Content	Example
`Value1`	Electronic Address	`[email address]`	bsmith@company.com
`Value2`	Blank	`[blank]`	Blank
`Status`	Status	`[shipping group status]`	Emailed

The following display values are configured for the in-store pickup shipping group type:

Value	Title	Content	Example

`Value1`	Store Address	`[store name]`
		`[address 1]`
		`[address 2]`
		`[city], [state] [zip code]`
		`[country]`
		`[phone number]`
			The Big Store
			11 Flatbush Street
			Suite 1509
			Brooklyn, NY 10023
			USA
			212-555-9874

`Value2`	Blank	`[blank]`	Blank
`Status`	Status	`[shipping group status]`	The goods are ready for pickup.

Display details are used on the following pages for all shipping group types, unless otherwise indicated:

Display Location	Value1	Value2	Status
Shipping Addresses Page Shipping Method Page (hard good only) Billing	X		
Order Review Page Return Items Page Confirmation Emails	X	X	
Order View Page Scheduled Orders Page	X	X	X

The `/DCS-CSR-UI/atg/commerce/custsvc/ui/HardGoodShippingGroupConfiguration.properties`, `ElectronicGoodShippingGroupConfiguration.properties` and `InStorePickupShippingGroupConfiguration.properties` files hold all of the properties of the page fragments. The following is an example of the `HardGoodShippingGroupConfiguration.properties` file:

```
$class=atg.commerce.csr.order.CommerceTypeConfiguration

addPageFragment=/atg/commerce/custsvc/ui/fragments/order/
   AddHardgoodShippingGroup
editPageFragment=/atg/commerce/custsvc/ui/fragments/order/
   EditHardgoodShippingGroup
displayPageFragment=/atg/commerce/custsvc/ui/fragments/order/
   DisplayHardgoodShippingGroup
imagePageFragment=/atg/commerce/custsvc/ui/fragments/order/
   HardgoodShippingGroupImage
type=hardgoodShippingGroup
addPageFragmentTitleKey=addHardgoodShippingGroupTitle
editPageFragmentTitleKey=editHardgoodShippingGroupTitle
imageHoverTextKey=hardgoodShippingGroupImageHoverText
resourceBundle=atg.commerce.csr.order.WebAppResources
```

The `DisplayHardgoodShippingGroupConfiguration.properties` file identifies the location of your page fragment JSP file and the `servletContext`. For example:

```
$class=atg.web.PageFragment

URL=/include/order/displayHardgoodShippingGroup.jsp
servletContext=DCS-CSR
```

The `displayHardgoodShippingGroup.jsp` file contains the values for the display details, including the title and the content. You must modify the `value1`, `value2` and `status` properties with your custom information. The combination of these parameters defines one display value, such as `value1`:

- `displayValue` – This optional parameter is used to display the value of the desired field
- `displayHeading` – This optional parameter is used to display the heading of the desired field

Example: Shipping Group Display Page Fragment Components

The following is a portion of the default configuration of the `displayhardgoodShippingGroup.jsp` file. In this example, the `displayHeading` parameter, which defines the display `value1` title, is set to use the Shipping Address header. The `displayValue` parameter, which defines the display `value1` content, is set to use the `addressView.jsp` fragment:

```
<c: if test="${propertyName == 'value1'}">
  <c: if test="${displayHeading == true}">
    <fmt: message key="shipping.address.header"/>
  </c: if>
  <c: if test="${displayValue == true}">
    <dsp: include src="/include/addresses/addressView.jsp"
       otherContext="${csrConfigurator.contextRoot}">
          <dsp: param name="address" value="${address}"/>
    </dsp: include>
  </c: if>
```

To display your customized shipping group title and content, modify the `displayHeading` and `displayValues` accordingly. For example, in the `displayElectronicShippingGroup.jsp` file, the `value1` title has been modified to define the `displayHeading` as Electronic Address and the `displayValue` as `emailAddress`:

```
<c: if test="${propertyName == 'value1'}">
  <c: if test="${displayHeading == true}">
    <fmt: message key='shippingSummary.electronicAddress.header'/>
  </c: if>
  <c: if test="${displayValue == true}">
    <li>
      <c: out value="${shippingGroup.emailAddress}"/>
    </li>
  </c: if>
</c: if>
```

As noted above, the title and content for `value1` is shown on the Shipping Addresses, Shipping Method (hard good only), Order Review, Return Items, Order View, Scheduled Orders pages and on confirmation e-mails. You may also modify the `value2` property as needed. Note that electronic shipping groups do not use the `value2` property by default, so it is left blank. You may add the `displayHeading` and `displayvalues` to the property as required.

Other components that can be customized within the display page fragment include the `status` property. This property is used to identify the status of the group object and displayed only on the Order View and the Scheduled Order pages. The default title for both the hard good and electronic shipping group is Status, and the content defaults to `shippingGroup.stateAsString`. Modify the heading of the display value to point to your customized information as needed:

```
<c: if test="${propertyName == 'status'}">
  <c: if test="${displayHeading == true}">
    <fmt: message key='shippingSummary.shippingStatus.header'/>
  </c: if>
  <c: if test="${displayValue == true}">
  <dsp: droplet name="ShippingGroupStateDescriptions">
    <dsp: param name="state" value="${shippingGroup.stateAsString}"/>
    <dsp: param name="elementName" value="stateDescription"/>
    <dsp: oparam name="output">
```

```
      <dsp: droplet name="IsHighlightedState">
        <dsp: param name="obj" value="${shippingGroup}"/>
        <dsp: oparam name="true">
            <span class="atg_commerce_csr_dataHighlight"><dsp: valueof
              param="stateDescription"></dsp: valueof></span>
        </dsp: oparam>
        <dsp: oparam name="false">
          <dsp: valueof param="stateDescription"></dsp: valueof>
        </dsp: oparam>
      </dsp: droplet>
    </dsp: oparam>
  </dsp: droplet>
  </c: if>
</c: if>
```

The `selectOptionText` property is used to identify the information that is presented in the address drop down on the Ship to Multiple Addresses and Split Quantity pages. By setting the `displayValue`, you configure what address information is presented in the drop down. For example, the default configuration for the hard good shipping group contains the following:

```
<c: if test="${propertyName == 'selectOptionText'}">
  <c: if test="${displayValue == true}">
    ${fn: escapeXml(address.address1)}${!empty address.address2 ? ' ' : ''
      }${!empty address.address2 ? fn: escapeXml(address.address2) : '' }
  </c: if>
</c: if>
```

The electronic shipping group `selectOptionText` property has been modified to include the following:

```
<c: if test="${propertyName == 'selectOptionText'}">
  <c: if test="${displayValue == true}">
    ${fn: escapeXml (shippingGroup.emailAddress)}
  </c: if>
</c: if>
```

Modify the display value with your customized page fragment.

Customizing a Shipping Group Type

For additional information on creating customized Shipping Groups, refer to the *Working with Purchase Process Objects* chapter of the *ATG Commerce Programming Guide*.

1. Refer to the *Order Tools* section of the *ATG Commerce Programming Guide* to create a new shipping group type. This includes defining the type-to-class name mapping for ShippingGroup objects.

2. Refer to the *Create a Shipping Group* section of the *ATG Commerce Programming Guide* to create the shipping group.

   Note: If you do not want to initialize the custom shipping group types using the `ShippinGroupDroplet`, continue on to step 5.

3. Write a new `ShippingGroupInitializer` implementation. The `initializeShippingGroups()` method should gather the user's `ShippingGroups` by type and add them to the `ShippingGroupMapContainer` referenced by the `ShippingGroupFormHandler`.

4. Open your custom application and modify `ShippingGroupDroplet.properties` `shippingGroupInitializers` parameter by adding your new shipping group type. For example:

```
/DCS-CSR-UI/atg/commerce/custsvc/order/ShippingGroupDroplet.properties
## ServiceMap of shippingGroupTypes to ShippingGroupInitializer
Nucleus components
shippingGroupInitializers+=\
newShippingGroup=/atg/commerce/custsvc/order/NewShippingGroupInitializer
```

5. Update the `/atg/commerce/custsvc/util/CSRConfigurator.properties` file `shippingGroupTypesToBeInitialized` property and `shippingGroupTypeConfigurations` property to include your new shipping group type. This configuration initializes shipping group types in the `ShippingGroupDroplet`.

   Initialize your new shipping group type by adding your new shipping group type to the `shippingGroupTypesToBeInitialized` property. For example:

```
/atg/commerce/custsvc/util/CSRConfigurator.properties
## Shipping group fragment settings
shippingGroupTypesToBeInitialized= newShippingGroup
shippingGroupTypeConfigurations+=\
/atg/commerce/custsvc/ui/NewShipingGroupConfiguration
```

6. Add the `CommerceTypeConfiguration` component to the `shippingGroupTypeConfigurations` property for any new supported shipping group type.

Payment Group Page Fragments

Payment group type configuration is defined using the `atg.commerce.csr.order.PaymentGroupTypeConfiguration` class. The class contains the following:

```
atg.commerce.csr.order.PaymentGroupTypeConfiguration extends
  atg.commerce.csr.order.CommerceTypeConfiguration
  protected PageFragment mEditRefundMethodPageFragment;
  protected PageFragment mDisplayRefundMethodPageFragment;
  mEditRefundMethodPageFragmentTitleKey;
```

The `PaymentGroupTypeConfiguration` class configures the payment group and refund methods. The base class properties configure the payment group. The refund methods are used for the returns and exchange pages. For detailed information on the `PaymentGroupTypeConfiguration` class, refer to the *ATG API Reference for Commerce Service Center*.

Payment group types are provided in these locations:

File	Location
Configuration File	`/DCS-CSR-UI/config/atg/commerce/custsvc/ui/`
Page Fragments	`/DCS-CSR-UI/j2ee-apps/DCS-CSR-UI/include/order/`

File	Location
Display JSPs	`/DCS-CSR-UI/j2ee-apps/DCS-CSR-UI/include/order/`
Add and Edit JSPs	`/DCS-CSR-UI/j2ee-apps/DCS-CSR-UI/panels/order/billing/`

There are five default payment group types: Credit Card, Store Credit, Gift Certificate, PayInStore and Cash. These are defined in the `SupportedPaymentGroupTypes` property of the `CSRConfigurator`. If an order contains a payment group type that is not listed, the order will not be loaded.

Note: Commerce Service Center does not support or display Commerce B2B payment group types. Because Commerce Service Center runs orders through multiple validations, customizations made to support B2B payment group types, such as B2B Approvals, may generate errors.

Working with Payment Group Page Fragments

To make customizations, you can modify the page fragments and override the component or page fragment properties. Payment group information is displayed on a number of pages, including the billing, order view, e-mail and refund method pages. Payment group information is displayed using the `PaymengGroupTypeConfiguration.displayPageFragment` and `PaymentGroupTypeConfiguration.displayRefundMethodPageFragment` property values.

The following table provides display information on Credit Card type payment groups:

Value	Title	Content	Example
`Value1`	Type	`[credit card type] – [last four digits of card]`	Visa – 2112
`Value2`	Expiration Date	`[card expiration year] / [card expiration year]`	11/12

`Value3`	Billing Address	`[first] [last]`
		`[address 1]`
		`[address 2]`
		`[city], [state] [zip code]`
		`[country]`
		`[phone number]`
			Bob Smith
			119 Grand Street
			Apt. 1509
			Brooklyn, NY 10023
			USA
			212-555-4321

`Status`	Status	`[payment group status]`	Authorization succeeded

The following table provides display information on Store Credit type payment groups:

Value	Title	Content	Example
`Value1`	Type	`Store Credit – Store Credit Number`	Store Credit – 1c1123

Value	Title	Content	Example
`Value2`	Amt. Remaining	`[store credit remaining]`	15.32
`Value3`	Blank	`[blank]`	blank
`Status`	Status	`[payment group status]`	Debited

The following table provides display information on Gift Certificate type payment groups:

Value	Title	Content	Example
`Value1`	Type	`Gift Certificate – Gift Certificate Number`	Gift Certificate – 1g4332
`Value2`	Amt. Remaining	`[amount remaining]`	15.32
`Value3`	Blank	`[blank]`	blank
`Status`	Status	`[payment group status]`	Debited

The following table provides display information on Pay In-store type payment groups:

Value	Title	Content	Example
`Value1`	Type	`Pay In Store`	Pay In Store
`Value2`	Blank	`[blank]`	blank
`Value3`	Blank	`[blank]`	blank
`Status`	Blank	`[blank]`	blank

The following table provides display information on Cash type payment groups:

Value	Title	Content	Example
`Value1`	Type	`Cash`	Cash
`Value2`	Blank	`[blank]`	blank
`Value3`	Blank	`[blank]`	blank
`Status`	Blank	`[blank]`	blank

The display values are used in the following locations:

Display Pages	Value1	Value2	Value3	Status
Billing Order Review Refund Type Confirmation Emails	X	X	X	
Order View Scheduled Orders Refund Review	X	X	X	X

The `/atg/commerce/custsvc/ui/CreditCardConfiguration.properties`, `StoreCreditConfiguration.properties` and `GiftCertificateConfiguration.properties` files in the `/DCS-CSR-UI` directory hold all of the properties of the page fragments. The configuration file contains the page fragment location.

The following examples use the Credit Card payment group; however, all three payment group types have corresponding files.

The following is an example of the `CreditCardConfiguration.properties` file:

```
$class=atg.commerce.csr.order.PaymentGroupTypeConfiguration
addPageFragment=/atg/commerce/custsvc/ui/fragments/order/AddCreditCard
editPageFragment=/atg/commerce/custsvc/ui/fragments/order/EditCreditCard
displayPageFragment=/atg/commerce/custsvc/ui/fragments/order/
  DisplayCreditCard

type=creditCard
addPageFragmentTitleKey=addCreditCardTitle
editPageFragmentTitleKey=editCreditCardTitle
resourceBundle=atg.commerce.csr.order.WebAppResources

editRefundMethodPageFragment=/atg/commerce/custsvc/ui/fragments/order/
  EditCreditCardRefundMethod
displayRefundMethodPageFragment=/atg/commerce/custsvc/ui/fragments/order/
  DisplayCreditCardRefundMethod
editRefundMethodPageFragmentTitleKey=editCreditCardRefundMethodTitle
```

The `CreditCardConfiguration.properties` file identifies the location of the page fragment, which in turn, defines your page fragment JSP file and the `servletContext`. For example:

```
$class=atg.web.PageFragment

URL=/include/order/displayCreditCard.jsp
servletContext=DCS-CSR
```

The `displayCreditCard.jsp` file contains the values for the display details, including the title and the content. You must modify the default `value1`, `value2`, `value3` and `status` properties with your custom information. The combination of these parameters defines one display value, such as `value1`:

- displayValue – This optional parameter is used to display the value of the desired field
- displayHeading – This optional parameter is used to display the heading of the desired field

Example: Payment Group Display Page Fragment Components

The following is a portion of the default configuration of the displayCreditCard.jsp file. In this example, the displayHeading parameter, which defines the display value1 title, is set to use the Billing Summary header. The displayValue parameter, which defines the display value1 content, is set to display the credit card information:

```
<c: if test="${propertyName == 'value1'}">
  <c: if test="${displayHeading == true}">
    <fmt: message key='billingSummary.commerceItem.header.type'/>
  </c: if>
  <c: if test="${displayValue == true}">
    <csr: displayCreditCardType creditCard="${paymentGroup}"/>
  </c: if>
</c: if>
```

To display your customized payment group title and content, modify the displayHeading and displayValues accordingly. For example, in the GiftCertificate.jsp file, the value1 title has been modified to use the same displayHeading but the displayValue has been modified to use the newOrderBilling gift certificate value:

```
<c: if test="${propertyName == 'value1'}">
  <c: if test="${displayHeading == true}">
    <fmt: message key='billingSummary.commerceItem.header.type'/>
  </c: if>
  <c: if test="${displayValue == true}">
    <fmt: message
     key="newOrderBilling.displayPaymentMethods.giftCertificate"/>
    <c: if test="${!empty paymentGroup && !empty
      paymentGroup.giftCertificateNumber }">
      <fmt: message key="common.hyphen"/>
      &nbsp;
      <c: out value="${paymentGroup.giftCertificateNumber}"/>
    </c: if>
  </c: if>
</c: if>
```

As noted above, the title and content for value1 is shown on the Billing, Order Review, Refund Type, Order View, Scheduled Orders, and Refund Review pages as well as on confirmation e-mails. You may also modify the value2 and value3 property as needed. Note that Store Credit and Gift Certificate payment groups do not use the value3 property by default, so it is left blank. You may add the displayHeading and displayvalues to the property as required.

Other properties that can be customized within the display page fragment include the status property. This property is used to identify the status of the group object and is displayed only on the Order View, Scheduled Orders and Refund Review pages. The default title for all payment groups is Status. Modify the display value to point to your customized information as needed. The following is an example of the displayCreditCard.jsp status:

```
<c: if test="${propertyName == 'status'}">
  <c: if test="${displayHeading == true}">
```

```
    <fmt: message key='billingSummary.commerceItem.header.state/>
  </c: if>
  <c: if test="${displayValue == true}">
  <dsp: droplet name="PaymentGroupStateDescriptions">
    <dsp: param name="state" value="${paymentGroup.stateAsString}"/>
    <dsp: param name="elementName" value="stateDescription"/>
    <dsp: oparam name="output">
      <dsp: droplet name="IsHighlightedState">
        <dsp: param name="obj" value="${paymentGroup}"/>
        <dsp: oparam name="true">
            <span class="atg_commerce_csr_dataHighlight"><dsp: valueof
              param="stateDescription"></dsp: valueof></span>
        </dsp: oparam>
        <dsp: oparam name="false">
          <dsp: valueof param="stateDescription"></dsp: valueof>
        </dsp: oparam>
      </dsp: droplet>
    </dsp: oparam>
  </dsp: droplet>
  </c: if>
</c: if>
```

Note that there are DSP tag files that, once copied into your customization library, can be used to extend default JSP files. For example, the `displayCreditCardType.tag` file displays the credit card name and renders the last four digits of the credit card number. For information on working with DSP tag files, refer to the *ATG Page Developer's Guide*.

Customizing a Payment Group Type

For additional information on creating customized Payment Groups, refer to the *Working with Purchase Process Objects* chapter of the *ATG Commerce Programming Guide*.

1. Refer to the *Order Tools* section of the *ATG Commerce Programming Guide* to create a new payment group type. This includes defining the type-to-class name mapping for `PaymentGroup` objects.

2. Refer to the *Extending the Payment Process to Support a New Payment Method* section of the *ATG Commerce Programming Guide* to create the payment group.

   Note: If you do not want to initialize the customer payment group type using the `PaymentGroupDroplet`, continue to Step 5.

3. Write a new `PaymentGroupInitializer` implementation. The `initializePaymentGroups()` method should gather the user's `PaymentGroups` by type and add them to the `PaymentGroupMapContainer` referenced by the `PaymentGroupFormHandler`.

4. Within your custom application, create a new `PaymentGroupInitializer` implementation and add it to the `ServiceMap` in the `PaymentGroupDroplet.properties` file `paymentGroupInitializers` property. For example:

   ```
   /atg/commerce/custsvc/order/PaymentGroupDroplet.properties
   ## ServiceMap of paymentGroupTypes to PaymentGroupInitializer
   Nucleus components
   paymentGroupInitializers+=\
   newPayment=/atg/commerce/custsvc/order/NewPaymentInitializer
   ```

5. Update the `/atg/commerce/custsvc/util/CSRConfigurator.properties` file

`paymentGroupTypesToBeInitialized` and `paymentGroupTypeConfigurations` properties to include your new payment group type.

This configuration initializes payment group types in the `PaymentGroupDroplet`. By default, this property initializes the `creditCard` and `storeCredit` types. To initialize a new payment group type, add your new payment group type to the `paymentGroupTypesToBeInitialized` and `paymentGroupTypeConfigurations` properties. For example:

```
/atg/commerce/custsvc/util/CSRConfigurator.properties
## Payment group fragment settings
paymentGroupTypesToBeInitialized=creditCard,storeCredit,newPayment
paymentGroupTypeConfigurations+=\
/atg/commerce/custsvc/ui/NewPaymentGroupConfiguration
```

Note: Ensure that the payment group type value matches the key defined in `OrderTools.paymentTypeClassMap` property.

Limiting Amounts for Payment Groups

When working on the Billing page an agent can enter any amount for a payment group. If the payment group is associated with a claimable item, the agent will not be allowed to enter more than the remaining or maximum - allowed amount.

You can limit the amount of your customized payment group types by extending the `CSRPaymentGroupRemainingAmount` servlet group droplet to set the payment group maximum-allowed and remaining amount limits.

CSR Payment Group Remaining Amount Servlet Bean Droplet

The `CSRPaymentGroupRemainingAmount` servlet bean droplet returns both the remaining and maximum-allowed amounts for the payment group. This droplet sets a maximum -allowed amount limitation in the Billing page and displays the remaining amount of the payment group on the Billing, Order View and Order Review pages.

Class	**`atg.commerce.csr.order.CSRPaymentGroupRemainingAmount`**
Components	`/atg/commerce/custsvc/order/ GetTotalAppeasementsForOrderDroplet.properties`

The `GetTotalOrderAppeasementDroplet` contains the following:

Input Parameters

- `order` –(Optional) This parameter is added for custom extensions that may need to work with the order.
- `paymentGroup` – (Required) This parameter obtains the remaining and max allowed amount

Oparams

- `output` – if an order has any payment groups

Output Parameters

- `remainingAmount` – The remaining amount is calculated for the payment group and returned in this parameter. The remaining amount is only calculated for store credit and gift certificates. This parameter returns the `PaymentGroup` remaining amount that is backed by the claimable item
- `maxAllowedAmount` – This parameter calculates and returns the maximum-allowed amount for the payment group. This parameter is used only in the billing page

You can calculate the remaining or maximum-allowed amounts for any custom payment groups by extending the `getRemainingAmount()` and `getMaxAllowedAmount()` methods within the droplet.

Copying Payment Group Types

When an agent starts the exchange process an exchange order is created. Relevant payment groups must be copied from the original order to the exchange order. The following payment groups are copied from the original order:

- Credit Card – copied by default from the original to the exchange order
- Store Credit – copied if there a remaining amount
- Gift Certificate – copied if there is a remaining amount

To copy additional payment group types, add the payment group type to the `PaymentGroupCopyManager.properties` file.

```
/atg/commerce/custsvc/returns/PaymentGroupCopyManager.properties
## ServiceMap of paymentGroupTypes to Copier Nucleus components
copiers+=\newPaymentGroupType=Component to copy new payment group
```

Note: The `newPaymentGroupType` should be one of the supported payment group types, outlined in the `atg/commerce/order/OrderTools.paymentTypeClassMap` file.

Configuring Shipping Addresses

You must configure shipping addresses for the following types of addresses:

- Returns
- Electronic

Configuring Return Shipping Addresses

To configure the shipping address for returns, perform the following steps:

1. Create a JSP file that contains your shipping address. For example:

```
<ul class="atg_commerce_csr_simpleList">
<li><strong>Ship return items to: </strong></li>
```

```
<li>My Company</li>
<li>Attn: Returns</li>
<li>100 Main Street</li>
<li>My City, My State</li>
<li>My Zip</li>
</ul>
```

2. Open the `atg/commerce/custsvc/ui/renderers/ReturnShippingAddressRenderer.properties` file and provide the location of the new JSP file and the `contextRoot` variable. For example:

```
# This is the default renderer for the returns line item page, default
# renderers will all have their id property set to "default". This
# property is primarily useful in targeting rules.
id=default
# The JSP that renders the returns line item
url=/panels/order/returns/NewreturnShippingAddress.jsp
contextRoot=/NewDCS-CSR
```

3. Save the `ReturnShippingAddressRenderer.properties` file.

For additional information on working with renderers, refer to the *ATG Service Center UI Programming Guide*.

Shipping and Payment Group Servlet Beans and Form Handlers

This section discusses some of the classes in Commerce Service Center that you may want to extend.

Available Priced Shipping Methods Droplet Servlet Bean

This droplet provides the available shipping method with price. Displays the available shipping methods with pricing for a particular shipping group. This extends the `atg.commerce.pricing.AvailableShippingMethodsDroplet`.

Class	**`atg.commerce.csr.pricing.AvailablePricedShippingMethodsDroplet`**
Components	`/atg/commerce/custsvc/pricing/AvailablePricedShippingMethodsDroplet`

The `AvailablePricedShippingMethodsDroplet` contains the following:

Input Parameters

- `shippingGroup` – The shipping group that requires pricing
- `order` – The order that is being priced

- `pricingModels` – A collection of shipping pricing models
- `profile` – The user RepositoryItem requesting the shipping method
- `locale` – Optional. The locale of the user requesting the shipping method

Oparams

- `error` – Contains any errors that occur
- `output` – Includes the `availablePricedShippingMethods`

Output Parameters

- `availablePricedShippingMethods` – The remaining amount is calculated for the payment group and returned in this parameter. The remaining amount is only calculated for store credit and gift certificates. This parameter returns the `PaymentGroup` remaining amount that is backed by the claimable item
- `errorMessage` – The message of any error that may have occurred

CSRShippingGroupFormHandler

The `CSRShippingGroupFormHandler` class is a form handler that manages single and multiple shipping stages of the checkout process. The `CSRShippingGroupFormHandler` extends the Commerce class `atg.commerce.order.purchase.ShippingGroupFormHandler` and allows you to add shipping-related audit-logging events.

Form Handler	**`atg.commerce.csr.order.CSRShippingGroupFormHandler`**
Component	`/atg/commerce/custsvc/order/ShippingGroupFormHandler`

For additional information on this form handler, refer to the *ATG API Reference for Commerce Service Center*.

CSRPaymentGroupFormHandler

The `CSRPaymentGroupFormHandler` class is a form handler that manages the billing stage of the checkout process. The `CSRPaymentGroupFormHandler` extends the Commerce class `atg.commerce.order.purchase.PaymentGroupFormHandler` and provides the ability to claim a store credit, gift certificate or coupon. By default, the `allowCouponClaim` flag is set to `false`.

Form Handler	**`atg.commerce.csr.order.CSRPaymentGroupFormHandler`**
Component	`/atg/commerce/custsvc/order/PaymentGroupFormHandler`

For additional information on this form handler, refer to the *ATG API Reference for Commerce Service Center*.

Configuring In-Store Pickup

In-store pickup is a Commerce feature that allows your customers to choose to receive merchandise they order at a local store. Commerce Service Center uses and extends these Commerce components to enable agents to assist with these types of orders.

When in-store pickup is enabled, agents can identify an item to be picked up in store. This launches the Commerce `GeolocatorServices`, which can either list all applicable stores, or provide a form that the agent can use to search for stores located at specified distances.

In addition to the stores location, Commerce Service Center can retrieve the store's inventory status for the product selected. Refer to the *ATG Commerce Programming Guide* for information on configuring Commerce locations, and setting inventory to work with in-store pickup.

When an agent views a product, the Pickup In-Store button is displayed for items that are eligible for in-store pickup. Items are eligible when they are recognized as hard goods, and are not flagged as available online only. Items are flagged as online only in Commerce using the `isOnlineOnly` method in `CatalogTools`. For detailed information on setting this flag, refer to the *ATG Commerce Programming Guide*.

When the agent continues through the checkout process, any in-store pickup items are displayed as such. The items are identified as part of the `InStorePickupShippingGroup` and one of two of the in-store pickup payment groups: `InStorePaymentPaymentGroup` , or `Cash`.

The `InStorePaymentPaymentGroup` acts as a place holder for payment until the recipient picks up the order. The `InStorePaymentPaymentGroup`, which is based on the Commerce `allowInStorePaymentWhenOtherShippingGroupTypesExist`, displays the Pay In Store payment option on the billing page. When the customer has picked up and paid for the merchandise, the payment group type changes to credit card or cash.

The `Cash` payment group type supports in-store pickup orders that were paid for in cash when the recipient picked up the item. This payment group cannot be used when the agent is generating a Web order, and is only available on submitted orders.

Enabling In-Store Pickup

To enable in-store pickup, use the `CSRConfigurator.usingInStorePickup` property. By default, this is set to `true`. To disable, set the property to `false`.

Setting Distances for Searches

When the `/atg/commerce/locations/GeoLocatorService.provider` is set to null, the service returns all available stores. If the `GeoLocatorService` is set to use a provider that you have configured, a search form is displayed to the agent. The search form supports either a city/state search, or a postal code search.

To configure the list of distances used in the search criteria drop down menu, modify the `/atg/commerce/order/purchase/InStorePIckupDistanceList.distances` property in the `DCS-CSR-UI` module. For example:

```
$class=atg.commerce.order.purchase.InStorePickupDistanceList
$scope=global
distances=\
          5,\
          10,\
            25,\
            50,\
            100
```

For information on the `GeoLocatorService` and setting up a provider, refer to the *ATG Commerce Programming Guide* and the *ATG Commerce Guide to Setting Up a Store*.

Setting Recipient Authorization for In-Store Pickup

Setting up authorization ensures that the correct recipient gets the item when it is received in a store. Agents can identify authorized recipients by entering their first and last names on the shipping methods page. Set the `/atg/commerce/custsvc/order/ShippingGroupFormHandler authorizedRecipientForInStorePickupRequired` property to `true` to display the authorization form.

Displaying the Cash Payment Group

Use the Dynamo Server Admin to configure the `InStorePaymentGroup` to display the `Cash` payment group. Run the following XML operation tag on the `/atg/commerce/order/OrderRepository` component to identify the payment group as cash. In this example, pg1900020 is the payment group ID.

Note that the `CashPaymentGroup` is not included in the `PaymentGroupDroplet`.

```
atg/commerce/order/OrderRepository
<update-item item-descriptor="paymentGroup" id="pg1900020">
  <set-property name="paymentGroupClassType" value="cash"/>
  <set-property name="paymentMethod" value="cash"/>
</update-item>
```

Working with Addresses

An address item can be referenced by more than one profile property. Commerce Service Center allows you to configure how these address items are copied or shared between profile properties.

Enabling and Disabling Copies

Because multiple profile properties, such as `defaultBilling`, `defaultShipping` and `secondaryAddresses`, can reference the same address item, Commerce Service Center supports `cascade-delete` data relationships.

For example, you want to move the default billing address to the `secondaryAddress`, and the `defaultBilling` property specifies `cascade="delete"` in the repository definition file. When you change the default billing address item in the customer profile, the current default address item is copied to the `secondaryAddress` map before being deleted from the `defaultBilling` property. To disable the automatic copying of addresses, set `cascade-delete` property to `false`.

For detailed information on repository data relationship and `cascade-delete`, refer to the *ATG Repository Guide*.

Disabling Address Sharing

Commerce Service Center allows you to identify an address as requiring exclusive access, or preventing multiple properties from sharing the same address. Using the `AddressCollectionReferenceManager`, you can set the `exclusiveReferenceProperties` to indicate which properties require an exclusive reference to an address. This disables the sharing of addresses between properties.

For example, to indicate that the `secondaryAddresses` cannot be the same as the `creditCardAddresses`, you would set the following:

```
AddressCollectionReferenceManger.properties:
# properties of the Profile
exclusiveReferenceProperties=creditCards.address,secondaryAddresses
```

To indicate that all properties require an exclusive reference, configure the following:

```
AddressCollectionReferenceManger.properties:
# properties of the Profile
exclusiveReferenceProperties=*
```

To indicate that single-value addresses require exclusive references, use the `DefaultAddressReferenceManager` to set the `exclusiveReference` property to `true.`

18 Working with Submitted Orders

Commerce Service Center allows the modification of orders that have been submitted but not yet fulfilled. When an agent modifies an order, the order's state determines how the edit process is handled.

Modifying Submitted Orders

When a submitted order is modified, the order is cloned into a transient order that captures all of the modifications made by the agent. The transient order is then reconciled with the original order in a single transaction. Updates are not committed to the original order unless the reconciliation process is successful. If the agent abandons the transient order, any modifications made will be lost.

The cloning technique is performed on selected orders based on their state. The `CSROrderHolder shouldCloneOrder(Order pOrder)` API determines if a particular order qualifies for this process. This API returns a result based on the following criteria:

- `CSRAgentTools.isOrderModifiable` – Compares the order's state against the listed non-modifiable states
- `CSRAgentTools.cloneEditOrderStates` – Compares the order's state against the listed states that may be cloned. The configured states include: `SUBMITTED`, `PROCESSING`, `PENDING_MERCHANT_ACTION`, `TEMPLATE` and `PENDING_AGENT_APPROVAL`

Additional states can be added by modifying these two properties in `CSRAgentTools`.

CSROrderHolder

This class defines the shopping cart used by an agent to modify customer orders. It provides an API for determining when the application is in clone edit mode, for storing the clone edit state object and for masking the current working order ID with the original order ID. The `loadOrder(Order)` API initiates the clone edit process for a given order and stores the clone edit state object.

Classes	**`atg.commerce.csr.order.CSROrderHolder`**
Components	`/atg/commerce/custsvc/order/ShoppingCart`

CSRAgentTools

This class provides the more generic API used by the application. This includes the configuration for submitted order states and the API for determining if an order should used the cloning feature.

Classes	**`atg.commerce.csr.util.CSRAgentTools`**
Components	`/atg/commerce/custsvc/util/CSRAgentTools`

CSRCommitOrderFormHandler

This form handler class provides the handlers for triggering the reconciliation process.

Form Handler	**`atg.commerce.csr.order.CSRCommitOrderFormHandler`**
Components	`/atg/commerce/custsvc/order/CommitOrderFormHandler`

For additional information on this form handler, refer to the *ATG API Reference for Commerce Service Center*.

Handling and Fulfillment

When a site that uses the Fulfillment module submits an order, the order and some of its contained objects will change state once the notification is received by the fulfillment sub-system.

Note: You must run the Fulfillment modules on both the Agent and Production clusters.

Object	State after submit	State after received by fulfillment
`Order`	`Submitted to Fulfillment`	`Processing`
`ShippingGroup`	`Initial`	`Pending shipment`
`ShippingGroupRelationship`	`Initial`	`Pending delivery`

A pipeline is executed to reconcile the changes with the original order. At the end of this chain there is a link that triggers the fulfillment notification messages. This process creates and sends the fulfillment `Modification` objects for changes made to the original order.

When the submitted order is modified by the Agent in the UI, new shipping groups and relationships may be created as a result. In this case, the new shipping groups and shipping group relationships will be saved in their `Initial` state. An application must extend the reconciliation process to perform any state modifications to these objects (for example, to mark them in their pending shipment and pending delivery states). This could be accomplished by either extending the reconciliation pipeline chain or responding the modification fulfillment events generated by the process.

For additional information on handling and fulfillment, refer to the *ATG Commerce Programming Guide*.

Fulfillment Notification for Order Modifications

Commerce Service Center sends standard fulfillment notification messages for all changes made to the order. They are sent out in the form of the following Commerce objects:

```
atg.commerce.fulfillment.PaymentGroupUpdate
atg.commerce.fulfillment.ShippingGroupUpdate
atg.commerce.fulfillment.GenericAdd
atg.commerce.fulfillment.GenericRemove
```

Commerce Service Center uses the Commerce `OrderFulfillmentTools` API to create these objects. For example, when a new payment group is added, Commerce Service Center calls the following `OrderFulfillmentTools` API to generate the modification object.

```
createGenericAddValueToValueModification(Modification.TARGET_PAYMENT_
GROUP, pPaymentGroup, Modification.TARGET_ORDER, pOrder);
```

For new Commerce items, shipping groups and payment groups Commerce Service Center sends `GenericAdd` messages. For updated Commerce items, shipping groups and payment groups, Commerce Service Center sends `GenericUpdate`, `ShippingGroupUpdate` and `PaymentGroupUpdate` objects respectively. Commerce items are the only objects for which Commerce Service Center sends `GenericRemove` events. For updates made to other Order properties, Commerce Service Center generates `GenericUpdate` events.

All the events are generated by the clone edit handlers during the reconciliation process that occurs when an agent commits his updates from the order review page.

Customizing Order Fulfillment

Commerce Service Center emits fulfillment `ModifyOrderNotification` events for the changes made to the order. Each event has an array of `Modification` objects attached that provide detail about the changes. The reconciliation pipeline triggers the creation of these objects:

```
<pipelinelink name="sendFulfillmentNotifications"
  transaction="TX_MANDATORY">
<processor jndi="/atg/commerce/custsvc/order/edit/processor/
  SendFulfillmentNotifications"/>
<transition returnvalue="1" link="sendAgentEvents"/>
</pipelinelink>
```

`OrderFulfillmentTools` contains the API for generating `ModifyOrderNotification` and `Modification` objects. The Commerce Service Center `CloneEditHandlers` use this API to generate the `Modification` objects based on the changes they manage on the order.

For example, when a payment group is added to the order, the payment group edit handler might generate a `Modification` object like this:

```
Modification mod =
  fullfillmentTools.createGenericAddValueToValueModification
(Modification.TARGET_PAYMENT_GROUP,pPaymentGroup,Modification.
  TARGET_ORDER,pOrder);
```

The modification would be attached to a `ModifyOrderNotification` event, along with all the other Modification objects, and then sent.

```
ModifyOrderNotification msg = new ModifyOrderNotification();
msg.setOrderId(order.getId());
msg.setModifications(pArrayOfModifications);
sendCommerceMessage(msg);
```

Cloning Orders

The order is cloned in an initialization pipeline that uses Repository cloning from `RepositoryUtils`. The order repository item is cloned to make a copy, and the copy is then loaded using `OrderManager loadOrder` API.

When the order has been cloned for modification, the application is in clone-edit mode. When an agent is working on a clone order there will be no indication in the UI. The UI always shows the original order ID. The `isCloneEditMode()` API in the `CSROrderHolder` determines when this mode is active. The `getOriginalOrder()` API returns the original order when required.

Cloning Pipeline Chains

The entire cloning process is facilitated by a pair of pipeline chains, which are defined in the `/atg/commerce/commercepipeline.xml` file:

- The `initSubmittedOrderEdit` chain prepares the order for editing by creating the clone copy. The pipeline that will be executed is determined by the order's state and the `CloneEditManager.reconcileOrderChains` property
- The `reconcileSubmittedOrder` chain reconciles the changes with the original order when everything is committed by the agent. The pipeline that will be executed is determined by the order's state and the `CloneEditManager.reconcileOrderChain` property

These two chains are executed based on the state of the order. For example, different chains could be defined for handling orders in different states. Commerce provides one set of chains for handling all orders in a `submitted` state as defined by the submitted order states configured in `CSRAgentTools`. These two chains can be modified using standard configuration override techniques.

When clone editing orders, the orders are copied into a transient state for the edit process, and any changes applied by initial pricing operations on the order are not immediately saved to the repository. This maintains the integrity of the original order.

The `CloneEditManager` component contains the configuration for mapping order states to the chain names using the `initializeEditChains` and `reconcileOrderChains` properties. The following describes each link in the two chains.

Chain: `initSubmittedOrderEdit` – executed at the start of the edit process:

Link	Processor Component	Description
`cloneOrderForEdit`	`/atg/commerce/custsvc/order/edit/processor/CloneOrderForEdit`	Performs the cloning of the order at the repository level and creates the Commerce objects from the newly created order item by calling `loadOrder`.
`validateCloneForEdit`	`/atg/commerce/custsvc/order/edit/processor/ValidateCloneForEdit`	Validates the clone order after it has been created.
`initializeCloneEditState`	`/atg/commerce/custsvc/order/edit/processor/InitializeCloneEditState`	Initializes a state object based on the two orders, which is used to later reconcile the changes back to the original order.
`setOriginalOrderPromotionCounts`	`/atg/commerce/custsvc/order/edit/processor/SetOriginalOrderPromotionCounts`	Stores the original promotion counts in the clone edit state so that they can be compared to the promotion counts after applying updates. Changes in the promotion count are used to identify the promotions to consume.
`saveCouponTracking`	`/atg/commerce/custsvc/order/edit/processor/SaveCouponTracking`	Saves coupon tracking meta-data in the clone edit state so that it can be restored after applying updates in reconciliation.

Chain: `reconcileSubmittedOrder` – execute to reconcile the changes with the original order:

Link	Processor Component	Description
`initializeReconcilitation`	`/atg/commerce/custsvc/order/edit/processor/InitializeReconciliation`	Prepares the state object for reconciliation.
`applyChanges`	`/atg/commerce/custsvc/order/edit/processor/ApplyChanges`	Copies the changes to the original order based on the contents of the clone order and information stored in the clone edit state object created in the initialization pipeline.
`executeValidateOriginalOrder`	`/atg/commerce/order/processor/ExecuteValidateForCheckoutChain`	Executes the core commerce process that validates the original order for checkout.

Link	Processor Component	Description
`removeEmptyShippingGroupsFromOriginal`	`/atg/commerce/order/processor/RemoveEmptyShippingGroups`	Executes the core commerce process that removes empty shipping groups from the original order.
`removeEmptyPaymentGroupsFromOriginal`	`/atg/commerce/order/processor/RemoveEmptyPaymentGroups`	Executes the core commerce process that removes empty payment groups from the original order.
`createImplicitRelationshipsForOriginal`	`/atg/commerce/order/processor/CreateImplicitRelationships`	Executes the core commerce process that creates implied relationships between commerce objects in the original order.
`setPaymentGroupAmountsOnOriginal`	`/atg/commerce/order/processor/SetPaymentGroupAmount`	Executes the core commerce process that sets the amount of the payment groups based on relationship in the original order.
`authorizePaymentGroups`	`/atg/commerce/custsvc/order/edit/processor/AuthorizePaymentGroups`	Executes a process that adjusts for the authorized amounts of the payment groups in the original. For example, new payment groups are authorized and changed payment groups have their authorized amounts adjusted.
`restoreCouponReferences`	`/atg/commerce/custsv/order/edit/processor/RestoreCouponReferences`	Restores the original coupon references on the pricing adjustments using the meta-data saved in the clone edit during initialization.
`updateAdjustmentsWithCouponOnOriginal`	`/atg/commerce/order/processor/UpdateAdjustmentsWithCoupon`	Updates the coupon references on the pricing adjustments for any new coupons that were claimed during the update.
`updateOriginalOrder`	`/atg/commerce/order/processor/UpdateOrder`	Calls `updateOrder` on the original order.
`reconcileGiftlistRepository`	`/atg/commerce/custsvc/order/edit/processor/ReconcileGiftListRepository`	Reconciles gift list quantities for changes made to the original order.
`consumePromotions`	`/atg/commerce/custsvc/order/edit/processor/ConsumePromotions`	Consumes coupon promotions that may have been claimed and used during the edit process.
`sendCouponPromotionUsedMessages`	`/atg/commerce/custsvc/order/edit/processor/SendCouponPromotionUsedMessages`	Sends the core commerce `PromotionUsed` JMS event for each consumed promotion identified by the previous processor.

Link	Processor Component	Description
`sendFulfillment Notifications`	`/atg/commerce/custsvc/order/edit/processor/SendFulfillment Notifications`	Sends fulfillment JMS events for each change applied to the original order.
`sendAgentEvents`	`/atg/commerce/custsvc/order/edit/processor/SendAgentEvents`	Sends agent audit events for each change made to the original order.

Submitted Orders and Clone Editing

When an agent selects an order that is considered modifiable, Commerce Service Center makes a copy of that order when it is considered to be in a clone edit state.

Because the order is copied into a transient state for the edit process, the changes applied by initial pricing operations on the order are not immediately saved to the repository. They are only saved if the agent completes the post-submit checkout of the order.

Cloning Core Classes

The following describes a few of the core classes and components that implement the clone edit feature and are the likely places for applications to apply overrides and extensions.

Clone Edit Manager

The `CloneEditManager` class provides the API for executing the pipeline chains and performing call backs to the `CloneEditHandlers` throughout the entire process. First, the original order is cloned at the repository level and executed, and then the changes in the clone order are reconciled with the original order. Refer to the *ATG Platform API Reference* for detailed information. These handlers are accessed using the main `DCS` module.

Classes	**`atg.commerce.order.edit.CloneEditManager`**
Component	`/atg/commerce/custsvc/returns`
Configuration	`cloneEditHandlers` provide a list of `CloneEditHandler` components. `intializeEditChains` is a map of pipeline chains used in the initialization process. Keyed by order state. `reconcileOrderChains` is a map of pipeline chains used in the reconciliation process. Keyed by order state.

Clone Edit Handler

This abstract class is used for creating application classes that participate in the entire process through a callback interface. Components of this type are configured in the core Commerce `CloneEditManager` class and are executed at various points in the initialization and reconciliation processes. This handler is accessed through the `DCS` module.

Class	**`atg.commerce.order.edit.CloneEditHandler`**

Subclasses	`atg.commerce.order.edit.CollectionEditHandler`
	`atg.commerce.order.edit.CommerceItemEditHandler`
	`atg.commerce.order.edit.CommerceItemMarkerEditHandler`
	`atg.commerce.order.edit.PaymentGroupEditHandler`
	`atg.commerce.order.edit.ShippingGroupEditHandler`
	`atg.commerce.order.edit.ManualAdjustmentEditHandler`
	`atg.commerce.order.edit.RelationshipEditHandler`
	`atg.commerce.order.edit.HandlingInstructionEditHandler`
	`atg.commerce.order.edit.MarkerEditHandler`
	`atg.commerce.order.edit.OrderPropertyEditHandler`

Components	`/atg/commerce/custsvc/order/edit/`
	`CommerceItemHandler`
	`ShippingGroupHandler`
	`PaymentGroupHandler`

CSR Clone Edit Handler

This class is used for extends the core Commerce `CloneEditHandler` class. Components of this type are configured in the `CSRCloneEditManager` and are executed at various points in the initialization and reconciliation processes. These handlers are accessed through the `DCS-CSR` module.

Class	**`atg.commerce.order.edit.CSRCloneEditHandler`**

Subclasses	`atg.commerce.order.edit.CollectionEditHandler`
	`atg.commerce.order.edit.CSRCommerceItemEditHandler`
	`atg.commerce.order.edit.CSRPaymentGroupEditHandler`
	`atg.commerce.order.edit.CSRShippingGroupEditHandler`
	`atg.commerce.order.edit.CSRManualAdjustmentEditHandler`
	`atg.commerce.order.edit.AgentCommentEditHandler`

Components	`/atg/commerce/custsvc/order/edit/`
	`AgentCommentHandler`
	`CloneEditManager`
	`CommerceItemGWPMakerHandler`
	`CommerceItemHandler`
	`CommerceItemMarkerHandler`
	`GWPOrderMarkerHandler`
	`HandlingInstructionHandler`
	`ManualAdjustmentHandler`
	`MarkerHandler`
	`OrderPropertyHandler`
	`PaymentGroupHandler`
	`RelationshipHandler`
	`ShippingGroupHandler`

The `CloneEditHandlers` components are called during various points in the clone edit process:

1. Post-repository cloning.

   The handlers are called just after the original order is cloned at the repository level. You can extend the `cloneOrder` method to customize cloning of data.

2. Verifying the clone.

   The `validateCloneOrder` handlers are called after the clone order has been created and loaded.

3. Initializing the `CloneEditState`.

   Once the clone order has been generated and verified, the `initializeCloneEditState` handlers are called to initialize any meta-data or objects that are needed for the `CloneEditState` object. For example, this is where the `CommerceItemHandler` maps the commerce items in the original order to those used in the clone order. These maps are used in the reconciliation process to identify those items that have been added, deleted or updated.

4. Reconciling the clone.

   After all updates are completed on the clone order, the reconciliation process executes the handlers to perform reconciliation of the data between the clone and the original order. For example, the `CommerceItemHandler` determines which items were added, removed or updated and updates the original order accordingly.

5. Creating post-reconciliation process events.

   After the reconciliation process is complete, the handlers are called to generate any fulfillment notification events for any changes they may have applied to the original order.

Clone Edit State

This class defines the state object used by the clone edit process. It provides access to the original and clone orders, as well as the API for adding and retrieving state information. It is created and returned by the initialization process and is required as input to the reconciliation process.

Classes	`atg.commerce.order.edit.CloneEditState`

Commerce Service Center stores this object in the window scoped `CSROrderHolder`, which can be accessed using the `getCloneEditState` API.

CSR Order Holder

This class defines the shopping cart used by an agent to modify customer orders. It provides an API for determining when the application is in clone edit mode, for storing the clone edit state object and for masking the current working order ID with the original order ID. The `loadOrder(Order)` API initiates the clone edit process for a given order and stores the clone edit state object.

Classes	**`atg.commerce.csr.order.CSROrderHolder`**
Components	`/atg/commerce/custsvc/order/ShoppingCart`

For more information on this component, refer to the *ATG API Reference for Commerce Service Center*.

CSR Agent Tools

This class provides the more generic API used by the application. This includes the configuration for submitted order states and the API for determining if an order should used the cloning feature.

Classes	**`atg.commerce.csr.util.CSRAgentTools`**
Components	`/atg/commerce/custsvc/util/CSRAgentTools`

For more information on this component, refer to the *ATG API Reference for Commerce Service Center*.

CSR Commit Order Form Handler

This form handler class provides the handlers for triggering the reconciliation process.

Classes	**`atg.commerce.csr.order.CSRCommitOrderFormHandler`**
Components	`/atg/commerce/custsvc/order/CommitOrderFormHandler`

For more information on this form handler, refer to the *ATG API Reference for Commerce Service Center*.

Extending Objects for Cloning

When an application has added new properties to the core Commerce objects, it may be necessary to modify the cloning process. Usually, all repository items referenced by the cloned item are also cloned into new transient repository items. This is known as the deep clone. However, you may want a shallow cloning of new properties or you may want to eliminate certain properties entirely from the cloning process.

Adding Objects for Cloning

You may extend core Commerce objects, for example: `Order`, `ShippingGroup`, `PaymentGroup`, `Relationship`. For example, you may have extended the default `OrderImpl` object type by creating a `MyOrderImpl` to add new properties. The configuration of the clone editing components may require adjustments or you may have to create class extensions.

If you have extended the core Commerce objects, the new object type and its new properties must be identified to the clone editing feature so that the new properties are copied to their original order counterpart during the reconciliation process. You must map which properties should be copied based on the type of object. The following is an example of the configuration for the `atg.commerce.order.OrderImpl` object type.

```
/atg/commerce/custsvc/order/edit/OrderPropertyHandler.properties$
$class=atg.commerce.order.edit.OrderPropertyEditHandler

cloneEditManager=/atg/commerce/custsvc/order/edit/CloneEditManager
keyPropertyName=id
```

```
# map of payment group properties that are copied from the clone to the original
# during reconcilation. The class name is the key
propertiesToCopyOnUpdate=\
  atg.commerce.order.OrderImpl=description,,state,,stateDetail,,taxPriceInfo
  ,,priceInfo,,specialInstructions
```

To add a new object type named `MyOrderImpl`, you would create an `/atg/commerce/custsvc/order/edit/OrderPropertyHandler.properties` file in your customization directory and add the new object type and its properties, which will be appended to the default `OrderPropertyHandler` file:

```
propertiesToCopyOnUpdate=\
myapp.commerce.order.MyOrderImpl=language,,locale
```

Note: Only new properties of the class need to be defined as all inherited properties are already covered by the super-type configurations. Additionally, it is not necessary to specify any new properties that will not be updated during the modification process.

The same type of configuration change is needed for extensions to the `CommerceItem`, `ShippingGroup`, `PaymentGroup` and `Relationship` objects. See `CommerceItemHandler`, `ShippingGroupHandler`, `PaymentGroupHandler` and `RelationshipHandler` respectively.

There are two ways that the repository cloning process can be managed: By excluding properties entirely from the cloning process, or by specifying which properties should use deep versus shallow cloning.

Excluding Properties from Cloning

The cloning feature can be configured to exclude certain properties from the clone process in the `atg/commerce/custsvc/order/edit/processor/CloneOrderForEdit` component. The `excludedOrderProperties` property maps order repository item types to a list of properties that should be excluded from the cloning process. The following is the default configuration that excludes the `pricelist` and `pricingModel` properties from the cloning process:

```
excludedOrderProperties=
  itemPriceInfo=priceList,
  pricingAdjustment=pricingModel
```

In this example, all properties that reference an item type of `itemPriceInfo` or `pricingAdjustment` will be cloned, but the `pricelist` and `pricingModel` properties will not be cloned and will be left null.

To exclude multiple properties:

```
excludedOrderProperties=
  itemPriceInfo=priceList|discounted|currencyCode
```

Specifying Deep Versus Shallow Clone

To specify the level of clone, extend the `CloneOrderForEdit` processor's `createCloningPropExceptionsMap` API to create a hierarchical map that specifies which properties have special handling called property exceptions.

The keys used in this map are property names while values are null or another map if the property is an item.

The following example performs a shallow clone on any property named `priceinfo`:

```
Map priceInfoExc = new HashMap();
priceInfoExc.put("priceInfo",null);
return priceInfoExc;
```

The following example performs a shallow clone of only the property named `pricelist` on any item found in a property named `priceInfo`:

```
Map priceInfoExc = new HashMap();
Map priceInfoProps = new HashMap();
priceInfoProps.put("pricelist",null);
priceInfoExc.put("priceInfo", priceInfoProps);
return priceInfoExc;
```

An application can also introduce a custom `CloneEditHandler` to perform special handling on any application-specific properties. For example, an application may want to exclude a custom property from the repository and handle the cloning process another way. The `CloneEditHandler` callback interface is executed after the repository cloning process is finished and provides an opportunity for applications to execute post-cloning logic.

19 Configuring Scenarios

Using Scenarios

You can use scenarios to configure actions that are taken when internal users perform various tasks. Scenarios are also used to configure events that are available to external users, such as cross-sells or promotions. For general information on creating and working with scenarios, refer to the *ATG Personalization Guide for Business Users*. For information on scenarios used with Commerce, refer to the *Using Commerce Elements in Scenarios* chapter of the *ATG Commerce Guide to Setting Up a Store*.

Configuring Scenario Events

When you use the ACC scenario editor on an ATG instance running Commerce Service Center, the scenarios you create are *internal* scenarios that respond to actions performed by internal users such as agents, not external users such as customers.

Commerce Service Center includes a number of scenario events that can be triggered by agent activities. You can incorporate these events in internal scenarios that you create.

The following table lists the scenario events included with Commerce Service Center:

Event display name	**Triggered when . . .**
Agent adds item to order	Agent adds an item to an order.
Agent adds payment group	Agent adds a payment group to an order.
Agent adds shipping group	Agent adds a shipping group to an order.
Agent cancels order	Agent cancels an order.
Agent changes item quantity	Agent changes an item’s quantity.
Agent claims item	Agent claims a coupon, gift certificate, or store credit.
Agent create order	Agent creates a new order.
Agent creates an order comment	Agent creates a new order comment.
Agent edits payment group	Agent edits a payment group.
Agent edits shipping group	Agent edits a shipping group.

Event display name	Triggered when . . .
Agent exchanges order	Agent exchanges an order.
Agent issues store credit	Agent issues a store credit to a customer.
Agent overrides a price	Agent overrides the price of a shipping group or commerce item.
Agent receives a return item	Agent receives an item that has been returned.
Agent removes item from order	Agent removes an item from an order.
Agent returns order	Agent returns an order.
Agent splits a shipping group	Agent splits a shipping group’s commerce items between shipping groups.
Agent submits order	Agent submits an order.
Agent views credit card	Agent views a credit card.
Agent views order	Agent views an order.
Agent views order payment	Agent views an order’s payment information.
Agent views order returns	Agent views order return information.

Working with Scenario Managers

Scenarios are managed and run by the Scenario Manager services, `ScenarioManager` and `InternalScenarioManager`. Depending on the subject of the scenario, the agent or the customer, you must know which Scenario Manager to run.

ScenarioManager and InternalScenarioManager

Just as you need to be aware of whether your slots and scenarios are displaying content targeted to the agent or the customer they are helping, you need to be aware of where those scenarios need to be created. In a production environment, ATG applications run on an internal server cluster and your external website runs on an external server cluster. Each instance in the internal cluster uses two separate scenario managers: the standard scenario manager, `ScenarioManager`, which handles scenarios in which the main subject of the scenario is the customer and the internal scenario manager, `InternalScenarioManager`, which handles scenarios in which the main subject of the scenario is the agent. Each instance of the external cluster only has a standard scenario manager in which you create scenarios where the main subject of the scenario is the customer. When you author a scenario that populates a slot, the scenario must be created in the appropriate cluster (internal or external) depending on its use.

- Scenarios in which the customer is the subject of the scenario belong in the standard scenario manager. These scenarios should be created in the external cluster. (Examples include when the customer’s profile drives the scenario, or the customer’s behavior on the external website drives the scenario)
- Scenarios in which the agent is the subject of the scenario belong in the internal scenario manager. These scenarios should be created in the internal cluster. (Examples include when the agent’s profile drives the scenario, or if actions the agent takes in Service Center drives the scenario)

This means that you will have to create one set of scenarios for delivering content in which the agent is the subject of the scenario (in the `InternalScenarioManager` service) and one set of scenarios for delivering content in which the customer is the subject of the scenario (in the `ScenarioManager` service).

As with any environment in which you are using scenarios, one of the scenario instances in each cluster will have to be configured as the Scenario Editor Server. Internal scenarios are created and updated while pointing to the Scenario Editor Server designated for the internal cluster. External scenarios are created and updated while pointing to the Scenario Editor Server designated for the external cluster.

Scenario Managers in a Development Environment

If you are working in a development environment where the external and internal environments are running on the same machine, you will need to point your ACC to either the standard Scenario Manager or the `InternalScenarioManager` service. The ACC can only display one of these at a time; it is configured to show the Standard `ScenarioManager` by default. You must manually configure the ACC to point to the `InternalScenarioManager` to create scenarios for the internal environment. See Configuring the ACC to Point to the InternalScenarioManager (page 139) for instructions on how to do this.

Scenario Managers in a Production Environment

If you have a production environment running, you will have two clusters: one for your internal applications (such as the ATG application) and one for your external sites (such as your external website). In this situation, the ACC in the internal cluster can still point to both the `InternalScenarioManager` and the standard `ScenarioManager`. However, the external cluster, which uses only the standard `ScenarioManager`, should be used to create or update scenarios in which the customer is the main subject.

The `ScenarioManager` on the external cluster shares the same database as the `ScenarioManager` on the internal cluster. This enables you to create your customer-subject scenarios in just one place. The scenarios created in the `ScenarioManager` on the external cluster will not show up automatically in the `ScenarioManager` for the internal cluster. You must update the scenarios across the clusters.

Updating Scenarios Across Clusters

If you create or update any of the scenarios running on the `ScenarioManager` in the external cluster, you will need to manually update the `ScenarioManager` in the internal cluster. This update ensures that the two `ScenarioManagers` are not out of sync.

To make this update you will need to go to the Dynamo Server Admin console and navigate to the `ProcessUpdateService` component and invoke the `updateAllProcesses` method. This will update all scenarios in all instances of the `ScenarioManager` in the internal cluster.

The path to the `ProcessUpdateService` component is:

```
<ATG10dir>/nucleus/atg/svc/scenario/ProcessUpdateService/
```

If you are running in a development environment and/or are not using an external website, you do not need to worry about using the `ProcessUpdateService` to communicate scenario changes across clusters.

Configuring the ACC to Point to the InternalScenarioManager

In a development environment you must configure the ACC to point to the `InternalScenarioManager`. To do this you need to specifically include the `DSS.InternalUser.ACC` module when creating your environment.

For detailed information on working with scenarios, refer to the *ATG Personalization Programming Guide*.

Configuring Process Editor Servers

The agent-facing server is configured with two Scenario Manager Configuration files, where you specify the location of the Process Editor Server and any global servers:

- `/atg/scenario/scenarioManager.xml` – configuration for external users
- `/atg/scenario/internalScenarioManager.xml` – configuration for internal users

Configuring the Customer-Facing Scenario Manager

The management server is aware of scenarios only for external users (users of the customer-facing website). You configure the `scenarioManager.xml` file to have one Process Editor Server, which is an Service Center instance:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<process-manager-configuration>
 <process-editor-server>
  <server-name>ATG CSC_Host: 8851</server-name>
 </process-editor-server>
</process-manager-configuration>
```

Configuring the Agent-Facing Scenario Managers

The agent-facing server is aware of scenarios for both internal users and external users. You must configure the agent-facing server as follows:

- `internalScenarioManager.xml` – include the Service Center server as a Process Editor Server, so that you can create and edit scenarios for internal users:

  ```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
  <process-manager-configuration>
  <process-editor-server>
  <server-name>ATG CSC_Host:8851</server-name>
  </process-editor-server>
  </process-manager-configuration>
  ```

- `scenarioManager.xml` – include a management server as a Process Editor Server, so that you can create and edit scenarios for external users:

  ```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
  <process-manager-configuration>
  <process-editor-server>
  <server-name>ATG Customer-facing_Host:8851</server-name>
  </process-editor-server>
  </process-manager-configuration>
  ```

20 Reporting and Logging

Commerce Service Center provides the ability to configure reporting and logging of a number of customer and agent actions.

Commerce Service Center Reporting Framework

The reporting framework used by Commerce Service Center is similar to the framework used by Commerce. Commerce Service Center gathers log files that contains order and agent information and loads it into the warehouse. For information on setting up and configuring the Commerce reporting framework, refer to the *Preparing to Use Commerce Reporting* section in the *ATG Commerce Programming Guide*. For information on using Commerce Service Center reports, refer to the *ATG Reports Guide*.

Data Collection Overview

As with all reporting, Commerce Service Center reporting data collection starts with the firing of a log-worthy event. The `EventListener` listens for the events, gets the appropriate object and passes it to the `LogEntryQueueSink`. The `EventListener` Nucleus component is configured with the property `dataListeners=LogEntryQueueSink`.

The `LogEntryQueueSink` property is a `DataCollectorQueue` type property and ensures the correct timing of writing to the log files. The `LogEntryQueueSink` component is configured with the property `dataListeners=LogEntryGenerator` to ensure that message will be passed on to the `LogEntryGenerator`.

The `LogEntryGenerator` property is used to generate a `LogEntry` object. This object is passed to the `LogEntryLogger` that has been configured with the parameters `dataListeners=LogEntryFileLogger`. The `LogEntryLogger` component of the `RotationAwareFormattingFileLogger` class logs items to the named file, and then rotates the log file based on a schedule and data threshold. It also contains a `formatFields` property that is used to indicate properties that should be written to the file.

Returns and Exchanges Data Collection Properties

The returns and exchange data collection process starts with the firing of the return/exchange event. The `ReturnFormHandler` fires the `ReturnOrder` event. The `ReturnEventListener` listens to the events.

The `ReturnEventListener` file is configured as follows:

```
$class=atg.commerce.reporting.ReturnEventListener
enabled^=/atg/dynamo/service/DWDataCollectionConfig.enabled
```

```
dataListeners=ReturnLogEntryQueueSink
returnOrderJMSType=atg.commerce.csr.ReturnOrder
exchangeOrderJMSType=atg.commerce.csr.ExchangeOrder
```

The `ReturnLogEntryQueueSink` listens to messages from event listener and queues the log entries to avoid performance bottleneck. Calls made to this component are queued and then passed to the `ReturnLogEntryGenerator`. The `ReturnLogEntryQueue` configuration file is configured as follows:

```
$class=atg.service.datacollection.DataCollectorQueue
dataListeners=ReturnLogEntryGenerator
```

The `LogEntryQueue` passes the data to the `ReturnLogEntryGenerator` that generates the `ReturnLogEntry` and passes it to the `ReturnLogEntryLogger`. The `ReturnLogEntryGenerator` file is configured as follows:

```
$class=atg.commerce.reporting.ReturnLogEntryGenerator
dataListeners=ReturnFileLogger
enabled^=/atg/dynamo/service/DWDataCollectionConfig.enabled
```

The `ReturnFileLogger` component is responsible for writing logs items to the named file, as well as rotating log files based on schedule and data thresholds. The `ReturnFileLogger` component also has `formatFields` property that indicates which properties should be written to file. The `ReturnFileLoggerLogger` configuration file contains the following:

```
#class
$class=atg.service.datacollection.RotationAwareFormattingFileLogger
# directory and file name of log file
logFileName=csc_return_
# Rotate log files automatically every 1 hour
schedule=every 1 hour
# Or rotate when there are 10,000 records in the file
dataItemThreshold=10000
# The directory to place all the log data files
defaultRoot^=/atg/dynamo/service/DWDataCollectionConfig.defaultRoot
# The centralized Dynamo scheduler
scheduler=/atg/dynamo/service/Scheduler
# Add a timestamp to all the names of the log files
timestampLogFileName=true
# Use this extension after the timestamp
logFileExtension=.data
# Format the time stamp like so (month-day-year_hour-minute-second-
# millisecond)
timestampDateFormat=MM-dd-yyyy_HH-mm-ss-SS
# properties to log (in order)
formatFields=timestampAsDate: MM/dd/yyyy HH: mm: ss,
returnOrder.returnRequestId
enabled^=/atg/dynamo/service/DWDataCollectionConfig.enabled
# Add a Unique ID to all the names of the log files
UIDLogFileName=true
# IdGenerator
idGenerator=/atg/dynamo/service/IdGenerator
# The JMS message type
logRotationMessageType=atg.reporting.ReturnOrder
# The messageSource component to send log rotation message
messageSource=/atg/dynamo/service/LogRotationMessageSource
```

Returns and exchanges log files are written to the `/atg/dynamo/service/DWDataCollectionConfig.defaultRoot` directory.

Call Data Collection Properties

When a call is initiated, a unique call ID is generated and assigned for each call. The call ID is used when starting and ending call events. The `atg.agent.events.CallEvent` event extends the `atg.agent.events.AgentEvent` event by adding `callId`, `startTime` and `endTime` properties. The `startTime` and `endTime` properties are recorded to the database as audit logs, while the `callId` is added to the audit database record.

The `/atg/agent/logging/AgentAuditQueue` listens for all agent events and passes the control to the `AgentAuditLogger` and provides an additional `AgentFileLogger` listener. This listener writes the data item to the file system. The `TypedEventDataListener` contains the `AgentAuditLogger`, the `AgentFileLogger` and the `SelfServiceAuditLogger` components.

The `CallLogEntry` and `CallLogEntryGenerator` classes provide the ability to add additional data to the log processes.

The `CallFileLogger` logs the data to the file system and creates an entry for the end call event. This logger will not log an entry for the start call event. When an agent ends a call the end call event is fired. Should an agent forget to end the call, when the window is closed or the `CallState` component is out of scope, the `doStopService` method will end the call event.

The `CallFileLogger` is configured with the following:

```
#class
$class=atg.service.datacollection.RotationAwareFormattingFileLogger
# directory and file name of log file
logFileName=svc_end_call_
# Rotate log files automatically every 1 hour
schedule=every 1 hour
#Or rotate when there are 10,000 records in the file
dataItemThreshold=10000
# The directory to place all the log data files
defaultRoot^=/atg/dynamo/service/DWDataCollectionConfig.defaultRoot
# The centralized Dynamo scheduler
scheduler=/atg/dynamo/service/Scheduler
# Add a timestamp to all the names of the log files
timestampLogFileName=true
# Use this extension after the timestamp
logFileExtension=.data
# Format the time stamp like so (month-day-year_hour-minute-second-
# millisecond)
timestampDateFormat=MM-dd-yyyy_HH-mm-ss-SS
# properties to log (in order)
formatFields=timestampAsDate: MM/dd/yyyy HH: mm: ss, callId,
startTimeAsDate: MM/dd/yyyy HH: mm: ss, endTimeAsDate: MM/dd/yyyy
HH: mm:ss,customerId, agentId
enabled^=/atg/dynamo/service/DWDataCollectionConfig.enabled
# Add a Unique ID to all the names of the log files
UIDLogFileName=true
# IdGenerator
idGenerator=/atg/dynamo/service/IdGenerator
# The JMS message type
logRotationMessageType=atg.reporting.svc.Call
# The messageSource component to send log rotation message
messageSource=/atg/dynamo/service/LogRotationMessageSource
```

Loader Pipeline Overview

Loading the return item into a transactional fact table is a process that requires a number of stages to obtain and work with the data in different ways. To clarify this process, it is presented as a data flow in which data flows from one processor to the next. Along the way the data is transformed until the final processor records the data into the transactional fact table.

Data loading starts with the implementation of the `Loader`. The `atg.reporting.datawarehouse.loader.Loader.Loader` component uses the `queueName` property to point to a log file. The `Loader` runs using a scheduler, creates a log file reader for returned items and invokes the `processReader` method of the `PipelineDriver` component.

The `PipelineDriver` reads delimited lines from the log file then parses them and populates a pipeline parameter. The parameter is then sent down a pipeline chain. The `atg.reporting.datawarehouse.loader.FilePipelineDriver.PipelineDriver` component uses the properties `paramPropertyNames` and `paramClasses` properties to specify the names and types of parameters that are read.

The `PipelineManager` runs the pipeline chain described in the `pipeline.xml` file. Pipelines consist of several processors. These processors collect data, prepare line items and insert them in into the data warehouse.

Returns and Exchanges Pipeline Processors

The following processors are used in the Submit Return pipeline:

Pipeline Link	**Description**

`fetchReturn`
 	
`/atg/reporting/datawarehouse/processes/custsvc/FetchReturnProcessor`
 	
`$class=atg.reporting.datawarehouse.commerce.csr.FetchReturnProcessor`
	Uses the return ID to look up the return/exchange in the repository.

`returnRequestLookup`
 	
`atg/reporting/datawarehouse/process/custsvc/ReturnRequestLookupProcessor`
 	
`$class=atg.reporting.datawarehouse.process.LookupPipelineProcessor`
	This processor fetches all return items in the warehouse for the current return ID. If there are return items the parameter map entry is created for the `resultPropertyName` value.

`checkReturnExists`
 	
`/atg/reporting/datawarehouse/process/WarehouseItemExistsProcessor`
 	
`$class=atg.reporting.datawarehouse.process.WarehouseItemExistsProcessor`
	This processor allows a switch to be implemented in the pipeline by determining if a warehouse item exists in the properties. If a `warehouseItemPropertyName` value exists in the parameter map, the current log record does not need to be processed. If the map entry does not exist the pipeline will process the next link.

Pipeline Link	Description

`createReturnLineItems`
 	
`/atg/reporting/datawarehouse/process/custsvc/CreateReturnLineItemsProcessor`
 	
`$class=atg.reporting.datawarehouse.commerce.csr.CreateLineItemsProcessor`
	This processor creates an array or the returned items map.

`allocateOtherRefund`
 	
`/atg/reporting/datawarehouse/process/custsvc/OtherRefundAllocatorProcessor.properties`
 	
`$class=atg.reporting.datawarehouse.commerce.AmountAllocatorProcessor`
	This processor gets refund information from `returnRequest.actualOtherRefund` and then distributes the refund amount based on return item refund amount or quantity. The `ItemRefundLineItemAlocator` looks at the return item refund subtotal. If the subtotal is greater than zero, then it distributes the amount based on the item refund amount.

`allocateReturnFee`
 	
`/atg/reporting/datawarehouse/process/custsvc/ReturnFeeAllocatorProcessor.properties`
 	
`$class=atg.reporting.datawarehouse.commerce.AmountAllocatorProcessor`
	This processor gets the return fee from `returnRequest.returnFee` and uses the `ItemRefundLineItemAlocator` to distribute the return fee across all return items.

`calculateTotalAdjustments`
 	
`/atg/reporting/datawarehouse/process/custsvc/LineItemTotalAdjustmentsProcessor`
 	
`$class=atg.reporting.datawarehouse.commerce.ComputerLineItemTotalProcessor`
	This process sums up all adjustments for each returned item, such as shipping share, tax share, and other return fee allocation amounts. The suggested shares are used to calculate the actual share. If the suggested share is zero, the share will be calculated based on the return item's quantity.

`calculateTotal`
 	
`/atg/reporting/datawarehouse/process/custsvc/LineItemTotalRefundProcessor`
 	
`$class=atg.reporting.datawarehouse.commerce.ComputerLineItemTotalProcessor`
	This processor sums up all total adjustments and item refund items.

`localCurrencyLookup`
 	
`/atg/reporting/datawarehouse/process/custsvc/LocalCurrencyLookupProcessor`
 	
`$class=atg.reporting.datawarehouse.commerce.CurrencyConverterProcessor`
	This processor converts local currency to standard currency values.

Pipeline Link	Description

`CurrencyConverter`
 	
`/atg/reporting/datawarehouse/process/custsvc/CurrencyConverterProcessor`
 	
`$class=atg.reporting.datawarehouse.commerce.CurrencyConverterProcessor`
	This processor converts shipping, tax and other refunds, the return fee, total adjustments and refunds.

`dayLookup`
 	
`/atg/reporting/datawarehouse/process/custsvc/DayLookupPipelineProcessor`
 	
`$class=atg.reporting.datawarehouse.process.DayLookupProcessor`
	This processor looks for the ID of the day for a given time stamp. This processor uses the return request `createdDate` property to return the ID.

`timeLookup`
 	
`/atg/reporting/datawarehouse/process/custsvc/TimeLookupPipelineProcessor`
 	
`$class=atg.reporting.datawarehouse.process.TimeLookupProcessor`
	This processor looks for the ID of the time for a given time stamp. This processor uses the return request `createdDate` property to return the ID.

`customerLookup`
 	
`/atg/reporting/datawarehouse/process/custsvc/CustomerLookupProcessor`
 	
`$class=atg.reporting.datawarehouse.process.RepositoryItemLookupProcessor`
`$scope=global`
	This processor gets the customer ID from the `returnRequest.order.profileId`. If the customer is not found in the data warehouse, it will look in the production schema. If found in neither schemas, the processor will return `Unspecified`.

`agentLookup`
 	
`/atg/reporting/datawarehouse/process/custsvc/InternalUserLookupProcessor`
 	
`$class=atg.reporting.datawarehouse.process.InternalUserLookupProcessor`
`$scope=global`
	This processor gets the agent ID from the `returnRequest.agent.repositoryId.` If the agent is not found in the data warehouse, it will look in the production schema. If found in neither schemas, the processor will return `Unspecified`.

`returnSalesChannelLookup`
 	
`/atg/reporting/datawarehouse/process/custsvc/ReturnSalesChannelLookupProcessor`
 	
`$class=atg.reporting.datawarehouse.process.EnumeratedPropertyLookupProcessor`
	This processor looks up the return channel ID in the data warehouse.

Pipeline Link	Description

`runReturnLineItemPipelineChain`
 	
`/atg/reporting/datawarehouse/process/custsvc/ReturnItemPipelineProcessor`
	This processor runs the return item pipeline for each element of the `LineItems` array.

The following processors are available in the `returnItem` chain:

Pipeline Link	Description

`lookupReturnSku`
 	
`/atg/reporting/datawarehouse/process/custsvc/SkuLookupProcessor`
 	
`$class=atg.reporting.datawarehouse.process.LookupPipelineProcessor`
	This processor looks up the SKU ID for each return item in the data warehouse.

`lookupReturnProduct`
 	
`/atg/reporting/datawarehouse/process/custsvc/ProductLookupProcessor`
 	
`$class=atg.reporting.datawarehouse.process.LookupPipelineProcessor`
	This processor looks up the product ID for each return item in the data warehouse.

`lookupReturnReason`
 	
`/atg/reporting/datawarehouse/process/custsvc/ReturnReasonLookupProcessor`
 	
`$class=atg.reporting.datawarehouse.process.LookupPipelineProcessor`
	This processor looks up the return reason ID for each return item in the data warehouse.

`logReturnItem`
 	
`/atg/reporting/datawarehouse/process/custsvc/ReturnItemLoggerProcessor`
 	
`$class=atg.reporting.datawarehouse.process.RepositoryLoggerProcessor`
	This processor creates a repository item for the logged data that is based upon the lookup properties.

Calls Pipeline Processors

The following processors are used in the call pipeline chain:

Pipeline Link	Description

`callLookup`
 	
`/atg/reporting/datawarehouse/process /svc/CallLookupProcessor.properties`
 	
`$class=atg.reporting.datawarehouse. process.LookupPipelineProcessor`
	This processor obtains the call item for the call ID. If the call item exists, the parameter map entry is created for the `resultPropertyName`.

`checkCallExists`
 	
`/atg/reporting/datawarehouse/process/ WarehouseItemExistsProcessor`
 	
`$class=atg.reporting.datawarehouse. process.WarehouseItemExistsProcessor`
	This processor implements a switch in the pipeline by determining if a warehouse item exists in the properties. If a `warehouseItemPropertyName` value exists in the parameter map, the current log record does not need to be processed. If the map entry does not exist, the pipeline will process the next link.

`customerLookup`
 	
`/atg/reporting/datawarehouse/process/ custsvc/CustomerLookupProcessor`
 	
`$class=atg.reporting.datawarehouse. process.RepositoryItemLookupProcessor $scope=global`
	This processor gets the customer ID from the parameter map. If the customer is not found in the data warehouse, it will look in the production schema. If found in neither schemas, the processor will return `Unspecified`.

`agentLookup`
 	
`/atg/reporting/datawarehouse/process/ custsvc/InternalUserLookupProcessor`
 	
`$class=atg.reporting.datawarehouse. process.InternalUserLookupProcessor $scope=global`
	This processor gets the agent ID from the parameter map. If the agent is not found in the data warehouse, it will look in the production schema. If found in neither schemas, the processor will return `Unspecified`.

`dayLookup`
 	
`/atg/reporting/datawarehouse/process/ custsvc/DayLookupPipelineProcessor`
 	
`$class=atg.reporting.datawarehouse. process.DayLookupProcessor`
	This processor looks for the start time timestamp from the parameter map as well as the lookup for the day of the time stamp.

`timeLookup`
 	
`/atg/reporting/datawarehouse/process/ custsvc/TimeLookupPipelineProcessor`
 	
`$class=atg.reporting.datawarehouse. process.TimeLookupProcessor`
	This processor looks for the start time timestamp from the parameter map as well as the lookup for the time of the time stamp.

Pipeline Link	Description

`enddayLookup`
 	
`/atg/reporting/datawarehouse/process/custsvc/DayLookupPipelineProcessor`
 	
`$class=atg.reporting.datawarehouse.process.DayLookupProcessor`
	This processor looks for the end time timestamp from the parameter map as well as the lookup for the end day of the time stamp.

`endtimeLookup`
 	
`/atg/reporting/datawarehouse/process/custsvc/TimeLookupPipelineProcessor`
 	
`$class=atg.reporting.datawarehouse.process.TimeLookupProcessor`
	This processor looks for the end time timestamp from the parameter map as well as the lookup for the end time of the time stamp.

`totalTime`	This processor calculates the total amount of call time in seconds.

`callLogger`
 	
`$class=atg.reporting.datawarehouse.process.RepositoryLoggerProcessor`
	This processor creates a repository item for the logged data based on the lookup properties.

Configuring Audit Logging

Commerce Service Center uses audit logging to record actions performed by Commerce Service Center agents in the agent audit repository.

Audit log records are saved in a standard GSA repository `/atg/agent/logging/AuditRepository`. The audit log repository items provide an audit trail of actions performed by the agent.

Although audit log repository items can be added, update, or removed using the standard GSA repository API, a series of components and classes are available to standardize the process of adding new items to the repository.

Note: There is no public API, other than direct GSA access, to update or remove audit log repository items.

Viewing Audit Logs

Commerce Service Center uses a system called *audit logging* to record actions performed by agents. Each type of action has a corresponding repository item type. These repository item types are used to log agent activity in the agent audit repository.

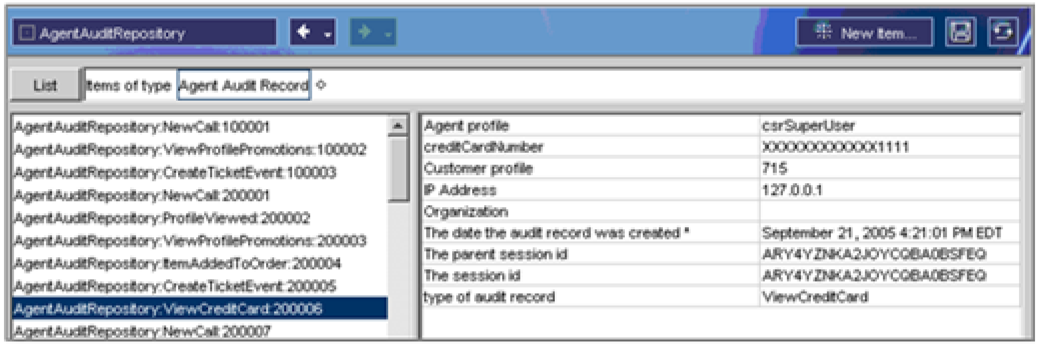
You can use the ACC to view these repository items:

1. Select the Content task area from the main ACC menu.
2. Select AuditAgentRepository from the submenu.
3. From the Items of Type drop-down list, select the type of item to view, and then click List.

If you select the first entry in the drop-down list, Agent Audit Record, the system displays all of the items in the repository, regardless of type. If you select any other entry, the system displays only the items of the specified type.

4. In the left pane, click the item to view.

   The system displays the log information in the right pane. For example:



AgentAuditRepository:NewCall:100001
AgentAuditRepository:ViewProfilePromotions:100002
AgentAuditRepository:CreateTicketEvent:100003
AgentAuditRepository:NewCall:200001
AgentAuditRepository:ProfileViewed:200002
AgentAuditRepository:ViewProfilePromotions:200003
AgentAuditRepository:ItemAddedToOrder:200004
AgentAuditRepository:CreateTicketEvent:200005
AgentAuditRepository:ViewCreditCard:200006
AgentAuditRepository:NewCall:200007

Agent profile	csrSuperUser
creditCardNumber	XXXXXXXXXXXX1111
Customer profile	715
IP Address	127.0.0.1
Organization	
The date the audit record was created *	September 21, 2005 4:21:01 PM EDT
The parent session id	ARY4YZNKA2JOYCQBA0BSFEQ
The session id	ARY4YZNKA2JOYCQBA0BSFEQ
type of audit record	ViewCreditCard

Adding a New Agent Audit Log Record

The following steps occur when creating an audit log record:

1. Generate an `atg.agent.events.AgentEvent` event object that contains all the relevant data to be recorded. The `AgentEvent` identifies the item-descriptor type of audit record and has properties containing relevant information for the log.

2. Fire the `AgentEvent` object through the `AgentMessageSource.sendAgentEventMessage` API. `AgentMessageSource` is defined as the message source for all `AgentEvent` events.

3. The `AgentAuditLogger` component is defined as a message sink and receives the `AgentEvent` and queues it for distribution to the appropriate `AgentAuditRecorder` instance.

4. `AuditLogRecorder` receives the `AgentEvent` from the `AgentAuditLogger` and creates the appropriate `RepositoryItem` sub-type of the `agent_audit` item-descriptor, as defined in the `AgentEvent's` type property. The `AuditLogRecorder` then, sets the item's properties from the `AgentEvent` data, and adds it to the `AuditRepository`.

When creating new audit log records, you must extend the `AgentEvent` class to pass your new event properties. You should then extend the `agent_audit` item-descriptor with a new sub-type to store your properties. Finally, you should extend the `AgentAuditRecorder.populateCustomProperties()` method to copy your custom `AgentEvent's` properties to your `agent_event RepositoryItem` sub-type.

Creating a New Agent Audit Log Record

The process outlined below uses creating a loyalty point redemption event as an example.

1. Add a new `AgentAudit` event subclass for the new audit record.

   This may not be necessary if the `AgentAudit` class has all the necessary properties. In this case, you only need to set the proper type of the event. Every event must extend `AgentEvent`. The following is an example of a `RedeemedLoyaltyPointsEvent`:

```
package atg.commerce.csr.events;
import atg.agent.events.AgentEvent;
```

```
public class RedeemedLoyaltyPointsEvent extends AgentEvent {
public static final String CLASS_VERSION = "$Id: $$Change: $";
private static final long serialVersionUID = -5747213631038504108L;
int mPointsRedeemed;
public void setPointsRedeemed(int pPointsRedeemed) {
mPointsRedeemed = pPointsRedeemed;
}
public int getPointsRedeemed() {
return mPointsRedeemed;
}
```

2. Add the new audit log repository item definition for the new type.

   Add an `item-descriptor` to the audit repository for the new agent audit in the`/atg/agent/logging/auditrepository.properties` file. Add an option to `auditType` property of the `agent_audit` item descriptor. For example:

```
//Add the item descriptor
<item-descriptor name="RedeemedLoyaltyPoints" super-type="agent_audit"
sub-type-value="RedeemedLoyaltyPoints">
<table name="csr_loyalty" id-column-name="id">
<property name="pointsRedeemed" data-type="int" column-
name="points_redeemed" display-name-resource="pointsRedeemed"/>
<property name="orderId" data-type="string" column-name="order_id"
display-name-resource="orderId"/>
</table>
</item-descriptor>
//Add the auditType
<option value="RedeemedLoyaltyPoints" code="1020"/>
```

3. Create an `AgentAuditLogger` component using one of the base classes provided or an extension. This component will receive the `AgentAudit` events of the new type. For example:

```
/atg/commerce/custsvc/logging/RedeemedLoyaltyPointsEventRecorder
$class=atg.agent.logging.ConfigurableAgentAuditRecorder
customProperties=\
pointsRedeemed=pointsRedeemed,\
orderId=orderId
```

   `ConfigurableAgentAuditRecorder` provides a convenient way of defining a recorder where the properties of the event map directly to the properties on the audit log repository item.

4. Configure the `AgentAuditLogger` with the new `AgentAuditRecorder` component mapped to the appropriate audit type. Add an entry into the `eventTypeToRecorderMap` property of the `/atg/agent/logging/`
   `AgentAuditLogger.properties` component to provide information on the new event recorder. For example,

```
eventTypeToListenerMap+=RedeemedLoyaltyPoints=/atg/commerce/custsvc/
logging/RedeemedLoyaltyPointsEventRecorder
```

5. Create an instance of the `AgentAudit` object and send it using the `AgentMessagesource.sendAgentEventMessage` API.

```
private void sendRedeemedLoyaltyPointsEvent
(
String pAgentId,
String pCustomerId,
String pOrderId,
int pPointsRedeemed,
String pTicketId
)
{
RedeemedLoyaltyPointsEvent redeemedLoyaltyPointsEvent = null;
redeemedLoyaltyPointsEvent =
getAgentMessagingTools().createRedeemedLoyaltyPointsEvent();
redeemedLoyaltyPointsEvent.setActivityType
CSRAgentMessagingTools.REDEEMED_LOYALTY_POINTS_TYPE);
redeemedLoyaltyPointsEvent.setPointsRedemeed (pPointsRedeemed);
getAgentMessagingTools().getAgentMessageSource().sendAgentEventMessage
(redeemedLoyaltyPointsEvent, null,
"myApp.events.RedeemedLoyaltyPoint");
return;
}
```

Disabling Audit Logging Events

To reduce the amount of data stored, you may want the audit logging system to record only certain activities and not others.

You can disable logging of an individual activity type by adding it to the `disabledTypes` property of the `/atg/agent/logging/AgentAuditLogger` component. This property is an array of the names of the activity types for which recording is disabled. For example, if you want to disable the recording of profile view events and order view events, you could set this property to `ProfileViewed, ViewOrder`.

If you want to disable audit logging entirely, set the `logEvents` property of `AgentAuditLogger` to `false`. The following is a list of the `eventTypeToListenerMap` properties:

Event Type	**Recorder in /atg/commerce/custsvc/logging/**
`ActivateScheduledOrder`	`ScheduledOrderEventRecorder`
`AddOrderFixedAmountAdjustment`	`OrderManualAdjustmentRecorder`
`AddPaymentGroup`	`PaymentGroupEventRecorder`
`AddShippingGroup`	`ShippingGroupEventRecorder`
`ApproveOrder`	`OrderEventRecorder`
`CancelOrder`	`OrderEventRecorder`
`ClaimItem`	`ClaimItemRecorder`
`CreateGiftlist`	`GiftlistEventRecorder`
`CreateOrder`	`OrderEventRecorder`

Event Type	Recorder in /atg/commerce/custsvc/ logging/
CreateOrderComment	CreateOrderCommentRecorder
DeactivateScheduledOrder	ScheduledOrderEventRecorder
EditPaymentGroup	PaymentGroupEventRecorder
EditShippingGroup	ShippingGroupEventRecorder
ExchangeOrder	ReturnOrderRecorder
GiftItemAddedToGiftlist	GiftitemEventRecorder
GiftItemQuantityChanged	GiftitemEventRecorder
GiftItemRemovedFromGiftList	GiftitemEventRecorder
GrantAppeasement	GrantAppeasementRecorder
GrantPromotionEvent	GrantPromotionEventRecorder
IgnorePromotionEvent	IgnorePromotionEventRecorder
ItemAddedToOrder	CommerceItemEventRecorder
ItemQuantityChanged	CommerceItemEventRecorder
ItemRemovedFromOrder	CommerceItemEventRecorder
ModifyGiftlist	GiftlistEventRecorder
OrderApprovalAddedEvent	OrderApprovalRecorder
OrderApprovalApprovedEvent	OrderApprovalRecorder
OrderApprovalRejectedEvent	OrderApprovalRecorder
PriceOverride	PriceOverrideRecorder
PriceReturnItem	PriceReturnItemRecorder
ReceiveReturnItem	ReceiveReturnItemRecorder
RejectOrder	OrderEventRecorder
RemoveOrderFixedAmountAdjustment	OrderManualAdjustmentRecorder
RemovePaymentGroup	PaymentGroupEventRecorder
RemoveShippingGroup	ShippingGroupEventRecorder
ReturnOrder	ReturnOrderRecorder
SaveOrder	OrderEventRecorder

Event Type	Recorder in /atg/commerce/custsvc/logging/
`SplitCostCenter`	`SplitCostCenterRecorder`
`SplitShippingGroup`	`SplitShippingGroupRecorder`
`SubmitOrder`	`OrderEventRecorder`
`UpdateScheduledOrder`	`ScheduledOrderEventRecorder`
`ViewCreditCard`	`ViewCreditCardRecorder`
`ViewOrder`	`OrderEventRecorder`
`ViewOrderCostCenters`	`OrderEventRecorder`
`ViewOrderPayment`	`OrderEventRecorder`
`ViewOrderPromotions`	`OrderEventRecorder`
`ViewOrderReturns`	`OrderEventRecorder`
`ViewOrderShipping`	`OrderEventRecorder`

Using Window Scoped Failover

With window scoped failover, upon server failure, an agent working on Commerce Service Center remains logged in with the current order, ticket, and profile information displayed in the global context area. The agent sees no change because of the failover.

Properties of window scoped components can be targeted for window backup by adding them to the `windowBackupPropertyList` property of the `/atg/dynamo/servlet/pipeline/WindowScopeManager` component. Upon session failover, the configured property values will be restored. These component property values must implement `Serializable` to be properly backed up.

The following components are failed over by default:

- `/atg/commerce/custsvc/order/ShoppingCart.current`
- `/atg/commerce/custsvc/order/ShoppingCart.restorableOrders`
- `/atg/commerce/custsvc/order/ShoppingCart.cloneEditState`
- `/atg/commerce/custsvc/order/ShoppingCart.loadTime`
- `/atg/commerce/custsvc/order/ShoppingCart.returnRequest`
- `/atg/commerce/custsvc/order/ViewOrderHolder.current`
- `/atg/commerce/custsvc/order/ViewOrderHolder.restorableOrders`
- `/atg/commerce/custsvc/order/ConfirmationInfo.order`

- `/atg/commerce/custsvc/order/ConfirmationInfo.profile`
- `/atg/commerce/custsvc/order/ConfirmationInfo.toEmailAddress`
- `/atg/commerce/custsvc/order/ConfirmationInfo.templateName`
- `/atg/commerce/custsvc/order/ConfirmationInfo.autoConfirmationEmailAddress`
- `/atg/commerce/custsvc/order/ConfirmationInfo.extraData`
- `/atg/commerce/custsvc/returns/ReturnsDataHolder.returnRequestID`
- `/atg/commerce/custsvc/environment/CurrentPriceListHolder`
- `/atg/commerce/custsvc/environment/CurrentCatalogHolder`
- `/atg/commerce/custsvc/profile/AddressHolder.addresses`
- `/atg/svc/agent/environment/CurrentSiteHolder.currentSite`

The following are also window scoped:

- `/atg/commerce/order/purchase/PaymentGroupContainerService`
- `/atg/commerce/order/purchase/ShippingGroupContainerService`

Adding Additional Components

Note: Making modifications to the existing window scoped and session scoped properties files may cause failover to not work as expected.

Add failover components for any customized code that you have written by modifying the `WindowScopeManager.properties` file in your custom module to add your customized components into the `windowBackupPropertyList+=`*`/new/custom/component.properties`* list.

Appendix A. Commerce Service Center Database Tables

This appendix describes the database tables used by Commerce Service Center.

Commerce Service Center Core Tables

Commerce Service Center uses the following database tables to store customer service information. These tables are installed when you run the CIM script.

The `csr_order_cmts` table is used by the Commerce order repository. The Nucleus component for this repository is `/atg/commerce/order/OrderRepository`. The rest of the tables described in this section are used by the Commerce Service Center returns and exchanges repository (`/atg/commerce/custsvc/CsrRepository`).

csr_order_cmts

This table stores agent comments associated with orders.

Column	Data Type	Constraint	Description
comment_id *(primary key)*	varchar(40)	not null	The unique ID associated with the comment.
order_id *(primary key)*	varchar(40)	not null	The ID of the order the comment is associated with. References `dcspp_order (order_id)`.
agent_id	varchar(40)	null	The profile ID of the agent who submitted the comment.
comment_data	varchar (2500)	not null	The text of the comment.
creation_date	timestamp	null	The date and time when the comment was submitted.
version	integer	not null	The GSA version of the repository item.

Commerce Service Center Order Approval Tables

Commerce Service Center uses the following database tables to store order approval information. These are stored in the approvals repository located in `/atg/commerce/custsvc/approvals/approvals-repository`.

csr_approval

This table stores information about the approval process.

Column	Data Type	Constraint	Description
approval_id *(primary key)*	varchar (40)	not null	The unique ID of the approval.
ticket_id	varchar (40)	not null	The ID of the ticket.
agent_id	varchar (40)	not null	The ID of the agent.
approver_id	varchar (40)	not null	The unique ID of the approver.
type	int	not null	The type of approval. The value is `orderApproval`.

approval_state	int	not null	The state of the approval. Values can be identified by code with the `useCodeForValue` attribute:
			
			`17011 PENDING`
			`17012 APPROVED`
			`12013 REJECTED`

site_id	varchar (40)	null	The unique ID of the site.
customer_id	varchar (40)	null	The unique ID of the customer.
creation_date	timestamp	not null	The date the approval was created.
completion_date	timestamp	null	The date the approval was completed.

csr_order_approval

This table stores information specific to an order approval.

Column	Data Type	Constraint	Description
approval_id *(primary key)*	varchar (40)	not null	The ID of the approval.
order_id	varchar (40)	not null	The ID of the order.

Column	Data Type	Constraint	Description
customer_email *(primary key)*	varchar (40)	null	The e-mail of the customer.
appeasement_total	double-precision	not null	The total appeasements.
order_total *(primary key)*	double-precision	not null	The total of the order.

Commerce Service Center Profile Tables

Commerce Service Center uses the following database tables to store agent approval information.

These tables are installed by the `<ATG10dir>/CSC10.2/DCS-CSR/sql/db_components/`*`database-vendor`*`/DCS-CSR_profile_ddl.sql` script.

csr_agent_app_limit

This table stores the approval limits for agents.

Column	Data Type	Constraint	Description
agent_id	varchar(40)	not null	The ID of the agent.
currency	varchar(3)	not null	The ID of the order.
app_limit *(primary key)*	numeric(19)	null	The limit that an agent can approve.

Commerce Service Center Logging Tables

Commerce Service Center uses the following database tables to store audit logging information.

These tables are installed by the `<ATG10dir>/CSC10.2/DCS-CSR/sql/db_components/`*`database-vendor`*`/DCS-CSR_logging_ddl.sql` script.

The tables described in this section are all used by the agent audit repository. The Nucleus component for this repository is `/atg/agent/logging/AuditRepository`.

csr_grant_appease

This table stores log records about the granting of appeasements (store credits).

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
appeasement_id	varchar(40)	null	The ID of the store credit.
amount	double precision	null	The amount of the store credit.

csr_price_override

This table stores log records of manual price overrides.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
order_id	varchar(40)	null	The ID of the order containing the price override.
component_id	varchar(40)	null	The ID of the order component (item or shipping group) whose price was overridden.
component_type	varchar(40)	null	The type of the order component whose price was overridden. Possible values: commerceItem, shippingGroup.
old_price	double precision	null	The previous price of the item or shipping group.
new_price	double precision	null	The new price of the item or shipping group.

csr_order_event

This table stores log records about orders that have been modified.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
order_id	varchar(40)	null	The ID of the modified order.
amount	double precision	null	The total cost of the order.

csr_return_order

This table stores log records of returned orders.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
ret_req_id	varchar(40)	null	The ID of the return request.
repl_order_id	varchar(40)	null	The ID of the replacement order (for an exchange).

csr_recv_rtrn_item

This table stores log records of receipt of returned items.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
item_id	varchar(40)	null	The commerce item ID of the returned item.
quantity	integer	null	The quantity of the item that was returned.

csr_claim_item

This table stores log records of the claiming of coupons, gift certificates, and store credits.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
claimable_id	varchar(40)	null	The ID of the coupon, gift certificate, or store credit.
claimable_type	varchar(40)	null	Type of item claimed (coupon, gift certificate, or store credit).

csr_ci_event

This table stores log records about changes to the quantities of items in orders.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
item_id	varchar(40)	null	The ID of the commerce item whose quantity was changed.
old_quantity	integer	null	The previous quantity of the item.
new_quantity	integer	null	The new quantity of the item.

csr_pg_event

This table stores log records about changes to payment groups in orders.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
pay_group_id	varchar(40)	null	The ID of the payment group that was changed.
update_type	one digit	not null	The type of update performed. 0=modify 1=remove 2=add

csr_split_sg

This table stores log records about splitting up items in orders among multiple shipping groups.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
src_ship_group_id	varchar(40)	null	The shipping group the item was moved from.
dest_ship_group_id	varchar(40)	null	The shipping group the item was moved to.
commerce_item_id	varchar(40)	null	The ID of the commerce item that was moved.
quantity	integer	not null	The quantity of the item that was moved.

csr_split_cc

This table stores log records about splitting up item costs among multiple cost centers.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of this log record.
src_cost_ctr_id	varchar(40)	null	The cost center that the item's costs were moved from.
dest_cost_ctr_id	varchar(40)	null	The cost center that the item's costs were moved into.
commerce_ident_id	varchar(40)	null	The ID of the commerce item whose costs were moved.
quantity	integer	not null	The quantity of the item whose costs were moved.

csr_sg_event

This table stores log records about changes to shipping groups in orders.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
ship_group_id	varchar(40)	null	The ID of the shipping group that was changed.
update_type	integer	not null	The type of update performed. 0=modify 1=remove 2=add

csr_upd_props

This table stores log records about changes to properties of repository items.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.

Column	Data Type	Constraint	Description
audit_id	varchar(40)	null	The ID of the log record this change is associated with. (For example, if the change is to a shipping group, this value is the ID of the log record in the csr_sg_event table.)
property_name	varchar(40)	null	The name of the property that was modified.
old_value	varchar(255)	null	The previous value of the property.
new_value	varchar(255)	null	The new value of the property.
version	integer	not null	The GSA version of the modified property.

csr_order_comment

This table stores log records about order comments submitted by agents.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
comment_id	varchar(40)	null	The ID of the comment.

csr_view_card

This table stores log records of agents viewing customers' credit card information.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
cc_number	varchar(20)	null	The credit card number of the card that was viewed.

csr_oma_event

This table stores log records about order adjustments submitted by agents.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
man_adj_id	varchar(40)	null	The unique ID of the manual adjustment.
adjustment_type	Integer	not null	The type of adjustment.
update_type	integer	null	The type of update performed.
reason	one-digit	null	The reason for the adjustment

csr_schd_event

This table stores log records of scheduled events.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
sch_order_id	varchar(40)	null	The unique ID of the scheduled order.
update_type	one-digit	not null	The type of update.

csr_appr_event

This table stores log records about order approvals requested by agents.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
approval_id	varchar(40)	not null	The unique ID of the approval request.
update_type	Integer	null	The type of update performed.

csr_order_appr_event

This table stores log records about order adjustments requested by agents.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.

Column	Data Type	Constraint	Description
order_id	varchar(40)	not null	The unique ID of the order.

csr_grt_prom_event

This table stores log records about order promotions.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
promo_id	varchar(40)	null	The unique ID of the promotion.

csr_ign_prom_event

This table stores log records about order promotions.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
promo_id	varchar(40)	null	The unique ID of the promotion.
order_id	varchar(40)	null	The unique ID of the order.

csr_gl_event

This table stores log records about gift list events.

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	not null	The unique ID of the log record.
giftlist_id	varchar(40)	null	The unique ID of the gift list.
event_name	varchar(64)	null	The type of event.

csr_gi_event

This table stores log records about gift items

Column	Data Type	Constraint	Description
id *(primary key)*	varchar(40)	null	The unique ID of the log record.
catalog_ref_id	varchar(40)	not null	The unique ID of the catalog.
old_quantity	Integer	not null	The amount of the previous quantity.
new_quantity	integer	not null	The amount of the new quantity.

Appendix B. Commerce Service Center Access Rights

The following table displays the access rights for the Commerce Service Center roles:

Rights	**csr Ticketing**	**csr Manager**	**csr Order**	**csr Profile**
`cmcAddProductByIdP`	x	x	x	x
`Allowed to Create New Profile`		x		x
`cmcApprovals`		x		
`cmcBillingP`	x	x	x	x
`commerce-custsvc-adjust-price-privilege`		x		
`commerce-custsvc-browse-promotions-privilege`		x	x	
`commerce-custsvc-issue-credit-privilege`		x		
`commerceDesignTab`	x	x	x	x
`commerceTab`	x	x	x	x
`cmcCompleteExchangeP`	x	x	x	x
`cmcCompleteOrderP`	x	x	x	x
`cmcCompleteReturnP`	x	x	x	x
`cmcConfirmExchangeP`	x	x	x	x
`cmcConfirmNewScheduleP`		x	x	
`cmcConfirmOrderP`	x	x	x	x
`cmcConfirmReturnP`	x	x	x	x

Rights	csr Ticketing	csr Manager	csr Order	csr Profile
`cmcConfirmOrderP`	x	x	x	x
`cmcConfirmOrderP`	x	x	x	x
`cmcConfirmUpdateScheduleP`		x	x	
`cmcCrossSellP`	x	x	x	x
`cmcCustomerAccountPanel`	x	x	x	x
`cmcCustomerCreateP`	x	x	x	x
`cmcCustomerInfoP`	x	x	x	x
`CustomerInformationPanel`	x	x	x	x
`CustomerOrderHistoryPanel`	x	x	x	x
`cmcCustomerP`	x	x	x	x
`cmcCustomerResultsP`	x	x	x	x
`CustomerResultsPanel`	x	x	x	x
`cmcCustomerSearchP`	x	x	x	x
`CustomerSearchPanel`	x	x	x	x
`customersTab`	x	x	x	x
`CustomerTicketHistoryPanel`	x	x	x	x
`cmcExchangeSummaryP`	x	x	x	x
`cmcExistingOrdcerP`	x	x	x	x
`cmcExistingSchedulcedOrderP`	x	x	x	x
`cmcGiftlistSearchP`		x		x
`cmcGiftlistsViewP`		x		x
`cmcGiftlistViewPurchaseModeP`		x		x
`GlobalPanel`	x	x	x	x
`HelpfulOpenByIDPanel`	x	x	x	x
`HelpfulRecentTicketsPanel`	x	x	x	x
`HelpfulTicketHistoryPanel`	x	x	x	x
`HelpfulTicketSummary`	x	x	x	x

Rights	csr Ticketing	csr Manager	csr Order	csr Profile
cmcMoreCatalogsP	x	x	x	x
cmcMorePriceListsP	x	x	x	x
cmcMultisiteSelectionPickerP	x	x	x	x
cmcOrderHistoryP	x	x	x	x
cmcOrderResultsP	x	x	x	x
cmcOrderReturnsP	x	x	x	x
cmcOrderSearchP	x	x	x	x
cmcProductCatalogBrowseP	x	x	x	x
cmcProductCatalogSearchP	x	x	x	x
cmcProductViewP	x	x	x	x
cmcPromotionsP	x	x	x	x
cmcPurchasedItemsHistoryP	x	x	x	x
cmcPurchaseHistoryP	x	x	x	x
cmcRefundTypeP	x	x	x	x
cmcRelatedOrdersP	x	x	x	x
cmcRelatedTicketsP	x	x	x	x
RespondComposeMessagePanel		x		x
respondTab		x		x
cmcReturnDetailsP	x	x	x	x
cmcReturnItemsP	x	x	x	x
cmcReturnsHistoryP	x	x	x	x
cmcReturnSummaryP	x	x	x	x
cmcScheduleCreateP		x	x	
cmcScheduledOrdersP	x	x	x	x
cmcSchedulesP	x	x	x	x
cmcScheduleUpdateP		x	x	
cmcShippingAddressP	x	x	x	x

Rights	csr Ticketing	csr Manager	csr Order	csr Profile
`cmcShippingMethodP`	x	x	x	x
`cmcShoppingCartP`	x	x	x	x
`cmcSubmittedOrdersP`	x	x	x	x
`TasksAllTicketsPanel`	x	x	x	x
`TasksMyTicketsPanel`	x	x	x	x
`tasksTab`	x	x	x	x
`TicketActivityPanel`	x	x	x	x
`cmcTicketHistoryP`	x	x	x	x
`TicketsCustomerInformationPanel`	x	x	x	x
`TicketsResultsPanel`	x	x	x	x
`TicketsSearchPanel`	x	x	x	x
`TicketsSummaryPanel`	x	x	x	x
`ticketsTab`	x	x	x	x
`workspaceLogin`	x	x	x	x

For additional information, refer to the *Setting Up Internal Access Control* (page 77) chapter.

Appendix C. CIM Configuration Components

The Configuration and Installation Manager script configures your Commerce Service Center environment.

When using CIM, you have the option to install a number of ATG applications, including Commerce Service Center. Depending on your requirements, you can run a single Commerce Service Center installation or install additional components to run your Commerce Service Center environment.

CIM also allows you to configure servers, including staging, preview and dedicated lock servers.

Available Added Functionality

The following functionality add-ons can also be configured when installing Commerce Service Center.

- Oracle Endeca Commerce Catalog Search
- Oracle Click-to-Connect On Demand
- Data Warehouse (Reporting)
- Multisite
- Publishing deployment and switching data sources for catalogs and price lists
- Live Index Server horizontal sharding of the Search index for orders

Server Instances

The following server instances are configured when running CIM:

Server Instance	Module List	Required Data Sources

Agent	`Fulfillment`
	`DCS.AbandonedOrderServices`
	`DCS-CSR-UI`
	`DAF.Search.Base.QueryConsole`
		`JTDataSource`
		`JTDataSource_production`

Production	`DCS-CSR-UI` `DCS.AbandonedOrderServices`	`JTDataSource`

Production and Management	`DCS-CSR.Management`
	`DCS.Search.Order.Index`
	`DCS.Search.Order.LiveIndex`
		`JTDataSource`
		`JTDataSource_production`

Add On Modifications

The following modifications occur to the server instances when using these optional add-ons:

Add On	Server Instance	Changes to Server Instance
Data Warehouse	Data Warehouse and Data Warehouse Loader	Adds `DCS.DW`, `Service.DW` and `DCS-CSR.DW` DDLs Adds `DCS-CSR.DW` to module list This adds the `atg/reporting/datawarehouse/JTDataSource` and `atg/reporting/datawarehouse/loader/JTDataSource` values.
Price Lists	Agent	Sets `/atg/commerce/custsvc/util/CSRConfigurator.usingPriceLists=true` and `/atg/commerce/custsvc/util/CSRConfigurator.usingSalePriceLists=true`
Endeca Catalog Search	Agent	Adds the `DCS-CSR-UI.Endeca` module, which enables communication with Endeca MDEX servers for catalog search.
ClickToConnect	Agent and Production	Adds the `DCS-CSR.ClickToConnect` module and the `JTDataSource` and `JTDataSource_production` values.
Merch UI	Publishing	If using Oracle ATG Web Commerce Merchandising, Commerce Service Center must be added to the deployment topology as a deployment target. Additionally, if switching data sources are enabled in Commerce, ATG Publishing must be aware of this to perform a switch on deployment
Multisite	All	Sets multisite capabilities by enabling multisite and setting the default Site ID and default Catalog ID in the `/atg/commerce/custsvc/util/CSRConfigurator`.

Add On	Server Instance	Changes to Server Instance
Sharding	All	Enables horizontal sharding of the search index for orders by setting the `/atg/commerce/search/`'s `shardingEnabled` property to `true`.

Data Source Configuration

The following data source information is configured for each of the server instances.

JTDataSource for Agent

The following information is configured for the agent `JTDataSource`. The `JTDataSource` information for the agent is stored in the `DCS-CSR` startup module.

SQL File

The SQL file used for the JTDataSource is `<ATG10dir>/CSC10.2/DCS-CSR/sql/db_components/`*`vendor`*`/DCS-CSR_logging_ddl.sql.`

Data Imports

The following files are imported using the `DCS-CSR` module:

Repository	Imported File	Versioned
`/atg/userprofiling/InternalProfileRepository`	`<ATG10dir>/CSC/DCS-CSR/install/data/internalUserData.xml`	No
`/atg/svc/option/OptionRepository`	`<ATG10dir>/CSC/DCS-CSR/install/data/csrOptions.xml`	Yes
`/atg/svc/option/OptionRepository_production`	`<ATG10dir>/CSC/DCS-CSR/install/data/csrOptions.xml`	No

JTDataSource for Production

The following information is configured for the production `JTDataSource`, which is the `JTDataSource_production` data source in the agent instance. The `JTDataSource` information for production is stored in the `DCS-CSR` startup module.

SQL File

The SQL files used for the JTDataSource are:
`<ATG10dir>/CSC/DCS-CSR/sql/db_components/`*`vendor`*`/DCS-CSR_ddl.sql` and `<ATG10dir>/CSC/DCS-CSR/sql/db_components/`*`vendor`*`/DCS-CSR_ticketing_ddl.sql`

Switching Data Source

The following information is configured for the switching data sources, which configures the `JTDataSource` data source in the production instance. The `switchingdatasource` add-on adds the switching `switchingCore` data source, as well as `switchingA` and `switchingB` data sources.

Note: Using a switching data source will also impact the catalog and pricing data sources for Commerce Service Center.

Data Imports

The following files are imported:

Repository	**Imported File**
`/atg/userprofiling/InternalProfileRepository`	`<ATG10dir>/CSC/DCS-CSR/install/data/internalUserData.xml`
`/atg/svc/option/OptionRepository`	`<ATG10dir>/CSC/DCS-CSR/install/data/csrOptions.xml`

CIM File Configuration

When running CIM, the following files are configured using these default settings:

Production Server File Configurations

The following property file configurations are set for the production server.

Search Property File Configuration

The `LaunchingService.properties` file `searchEngine` and `deployShare` properties are installed on the Commerce Service Center production server.

The `IndexingPeriodicService.properties` file is set to `enable=true` to identify the production server as a server that will be used by the indexing server to queue profile and order indexing requests.

The `/atg/userprofiling/search/ProfileOutputConfig.incrementalUpdateSeconds` and the `/atg/commerce/search/OrderOutputConfig.incrementalUpdateSeconds` properties, which determine the frequency for live indexing requests, are set to 5 seconds.

Note: If the server on which this is being configured is not the indexing server, set the `incrementalUpdateSeconds` to -1. It is best to configure a non-DRP server as your indexing server.

Agent Server File Configurations

The following configuration changes are made on the agent server.

Search Property File Configuration

The `LaunchingService.properties` file `searchEngine` and `deployShare` properties are installed on the Commerce Service Center agent server.

The `IndexingPeriodicService.properties` file is set to `enable=true` to identify the management server as a server that will perform queuing of index changes.

The `/atg/userprofiling/search/ProfileOutputConfig.incrementalUpdateSeconds` and the `/atg/commerce/search/OrderOutputConfig.incrementalUpdateSeconds` properties, which determine the frequency for live indexing requests, are set to 5 seconds.

Note: If the server on which this is being configured is not the indexing server, set the `incrementalUpdateSeconds` to -1. It is best to configure a non-DRP server as your indexing server.

Price Lists Add On Configuration

The following information is configured if installing Price Lists. The following information is stored in the `/atg/commerce/custsvc/util/CSRConfigurator.properties` file:

```
usingPriceLists=true
usingSalesPriceLists=true
```

Prerequisites for Running CIM

You will need the following Commerce Service Center information before running the CIM process:

1. Four Oracle database accounts, such as: `agent`, `production`, `publishing` and `reporting _loader_data-warehouse`
2. If you are configuring a switching data source, you will need two additional Oracle database accounts: `Switching_A` and `Switching_B`
3. Ensure that the supported version of your Web application server is installed.
4. Ensure that the supported version of the Java JDK is installed.
5. Ensure that the correct product versions have been installed in your environment using the ATG Installer. For information on installing these products, refer to your ATG documentation:
    - Oracle 11g client / drivers `(ojdbc5.jar` or `ojdbc6.jar`)
    - The Oracle ATG Web Commerce platform
    - Oracle ATG Web Commerce Reference Store or Oracle ATG Web Commerce
    - Oracle ATG Web Commerce Search

Note: If Commerce Service Center has previously been installed, you must drop each of the schemas before recreating them. You can do this by using `cim.sh` in the existing installation to drop the schemas. You will also need to rename the old ATG directory if this exists.

You will also need to remove any servers you have previously created in your Web application, as running CIM will create new server instances.

Running CIM

When running the CIM script, you will be prompted to install all products or just Commerce Service Center. For information on installing ATG applications, refer to your application documentation, and the *ATG Installation and Configuration Guide*.

CIM presents you with a sequence of menus and menu options that allow you to configure installed products. Multiple selections can be separated by a space. You can confirm your selections by typing D for Done.

To start CIM in the current window, run the following command:

```
./cim.sh or cim.bat
```

Once the products have been downloaded and installed, use CIM to configure them. If you wish to keep a record of your selections for future reference, CIM can create an output file.

When you run the CIM script, use the `–record` option to create an output file that records your selections that you can use to recreate your environment.

Appendix D. Configuring Oracle Click-to-Call On Demand

Oracle Click-to-Call On Demand is an optional application that, when integrated with Commerce Service Center, initiates and manages telephone communication between agents and customers.

Overview

A Click-to-Call integration with Commerce Service Center enables customers to initiate phone calls with agents. Once configured, Click-to-Call uses a page instrumentation engine to add JavaScript into your customer-facing Web pages. The JavaScript presents an icon and/or link to the customer who uses this link to enter a phone number that the Live Help On Demand Webcare system or an optional telephony (CTI) system uses to return their call. Commerce Service Center presents the agent with the customer's order information and profile, allowing the agent to assist the customer with their order.

Detailed information regarding the configuration and set up of Live Help On Demand features discussed in this section can be found in the Live Help On Demand documentation.

Initiating a Call

When the customer initiates a call, their customer and order information is written to the database and the customer's phone number is sent to Webcare. This information is transmitted using a token that creates a unique identifier that correlates the call with the customer's call data. The token is passed to the Click-to-Call system and is used later to obtain the data for the call, including the appropriate customer and order information, as well as the appropriate landing page for the agent.

Once Click-to-Call has obtained the token it establishes a call with both the telephony system (CTI) and the customer. The telephony system initiates the call with the agent.

Using a CTI System

Click-to-Call and Commerce Service Center can be configured to work with or without a CTI system. With a CTI system, once the unique call information has been obtained, the CTI system initiates a browser screen. The browser invokes a URL that passes the caller ID as a query parameter. Commerce Service Center looks up the call token from Webcare using the caller ID, looks up the customer information using the call token, and configures the agent's environment with the customer and order information.

Commerce Service Center also provides a mechanism for your CTI system to authenticate an agent. The CTI system can be configured to pass the a concatenation of the user name, the `FTCallID`, the caller's telephone number and the hash key to Commerce Service Center for authentication. The authentication process requires that this hash key is the same on both servers. If you do not wish to configure authentication, agents can log in to Commerce Service Center manually using the Commerce Service Center log in screen.

You can integrate your CTI system using the Live Help On Demand Agent Console to obtain the caller ID information from a screen scrape. To use the Agent Console, you need to install and configure it on each agent's desktop.

Specifying Links and Pop-Ups

Webcare creates and configures links that are presented to the customer. These pop-up icons and call information screens can be customized using logos, graphics and other UI components that are stored locally on your servers.

Links and pop-ups can be static or based upon rules that you create using the Live Help On Demand Engagement Engine Editor or the Oracle ATG Web Commerce rule process. For example, you can create a rule that presents a link when a customer is from a specific country, or that pops up a Click-to-Call icon if a customer's shopping cart has reached a specific amount.

Automatic Initialization of the Agent's Working Environment

Part of the process of receiving a call in Commerce Service Center is to initialize the agent's working environment with the correct set of objects.

By default, the following values are saved when a call is initiated on the storefront and then used to pre-populate the agent's working environment with the correct set of the objects:

- Profile ID – Used to load the active customer
- Order ID – Used to load the active order
- List Pricelist ID – Used to load the active list price list
- Sale Pricelist ID – Used to load the active sale price list
- Catalog ID – Used to load the active catalog
- Site ID – Used to load the active site if using multisite

Click-to-Call Requirements

To use Click-to-Call, you must have an active Click-to-Call account that includes access to the Webcare Engagement Engine Editor and the Live Help On Demand Agent Console. Contact your Oracle Support representative for further information.

The following modules are required to run Click-to-Call and should be installed on the agent-facing server:

```
DCS-CSR.ClickToConnect
```

```
Service.ClickToConnect
```

The following modules should be installed on the customer-facing server:

```
DCS.ClickToConnect
```

Refer to the Oracle Live Help On Demand documentation for information on Oracle Live Help On Demand Webcare and Oracle Live Help On Demand Agent Console requirements.

IBM WebSphere Requirements

If you are using Click-to-Call with an IBM WebSphere application server over SSL, you must ensure that the correct signing certificates are provided to WAS from Click-to-Call. To do this, you must set up the remote host and SSL port information from which WAS will retrieve the signer certificates. Refer to the IBM WebSphere application server documentation for information on obtaining certificate identification.

Oracle WebLogic Requirements

If you are using Click-to-Call with an Oracle WebLogic application server over SSL, you must ensure that the correct signing certificates are provided to allow WebLogic to recognize the Click-to-Call certificate. You must set up the Oracle Certificate Signing Request to obtain a certificate for the WebLogic server. Refer to the Oracle WebLogic documentation for information on obtaining certificate identification.

Configuring the Click-to-Call Account

Before you can set up Click-to-Call on either your store-facing pages or Commerce Service Center, you must provide Commerce Service Center with your account information.

Note: Before completing these steps, you must have a valid and active account for Live Help On Demand. Refer to your Oracle Support representative for information.

1. Configure Commerce Service Center to recognize your account information by editing the `localconfig/atg/clicktoconnect/Configuration.properties` file.
2. Add your account number, user name and password to the file.

   ```
   # Your ClickToConnect account ID
   accountId=Click to Connect Account ID

   # The ClickToConnect username
   username=Click to Connect User Name

   # The ClickToConnect password
   password=Click to Connect Password
   ```

3. Save the file when you have finished.

Adding an Agent to Commerce Service Center

Agents who use the Live Help On Demand Agent Console must be added to Commerce Service Center. Note that the login name for agents must be the same on both for Click-to-Call and Commerce Service Center. Use the Business Control Center to add agents to Commerce Service Center. Refer to the *ATG Business Control Center User's Guide* for additional information on creating agents.

Adding Agent Phone Numbers

When creating pop-ups and links using the Live Help On Demand Engagement Engine Editor, you need to provide the phone number of the call center, and if necessary, extension information. Refer to the Live Help On Demand Engagement Engine Editor documentation for additional information.

Configuring Commerce Service Center Pages

To run Click-to-Call with Commerce Service Center, you must add automatic page instrumentation filters. To create static links, you must create the button or link tag that will access Click-to-Call on your customer-facing store Web pages.

Configuring Automatic Page Instrumentation

Automatic page instrumentation allows you to automatically tag each of your customer-facing Web pages with the ATG JavaScript library references, as well as to output other parameters that are used for data persistence purposes, without changing any code for your site. When configuring automatic page instrumentation, set the page instrumentation filters that are used to add the Click-to-Call code to your pages.

The page instrumentation engine writes JavaScript variables to the Web page. These variables are used to identify the customer's environment and to store information that allows you to create customized rules that display links and pop-ups based on specific criteria. For example:

```
var _atg_estara_call_token\=100001-1234;
var _atg_estara_locale=en_US;
```

Adding Automatic Page Instrumentation

To add automatic page instrumentation, add the following to your customer-facing store's `web.xml` file:

```
<filter>
  <filter-name>ADCDataInsertFilter</filter-name>
  <filter-class>atg.adc.filter.ADCDataInsertFilter</filter-class>
  <init-param>
    <param-name>loggingDebug</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param>
    <param-name>loggingWarning</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
```

```
    <param-name>loggingError</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>loggingInfo</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>ADCDataInsertFilter</filter-name>
  <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

Configuring a Static Link

To configure a static link, or a link that remains within a navigational pane or frame within your Web pages, you must add a `<div>` tag in the page HTML where the link should appear. This section holds information that you create in the Live Help On Demand Engagement Engine Editor. The following example inserts a `<div>` that has a specific border style.

```
<div id="atg_ClickToCall_Link" style="border: 1px solid rgb(0, 0, 0);"></div>
```

Once you have created a `<div>`, the `<div>` tag ID is used by the link definitions that you define in the Live Help On Demand Engagement Engine Editor. Refer to the Live Help On Demand Engagement Engine Editor documentation for detailed information.

Disabling the Orphaned Session Service

The `OrphanedSessionService` removes orphaned session items from the `ClickToConnect` session repository. By default, the service is configured with the `DCS.ClickToConnect` module set to `enabled=true` and the `Service.ClickToConnect` configuration set to `enabled=false`. This service should be started on any instance where you have installed the `DCS.ClickToConnect` module and disabled on any instance configured with the `Service.ClickToConnect` module.

You can disable the service by setting the `/atg/clicktoconnect/OrphanedSessionService.properties` file `enabled` property to `false` in your `/localconfig` directory.

Creating Click-to-Call Links

Using the Live Help On Demand Engagement Engine Editor, you create links that customers use to initiate a call. Click-to-Call links can be generic, which present the same links to all of your customers, or they can be customized for specific audiences or locations.

Creating a Link

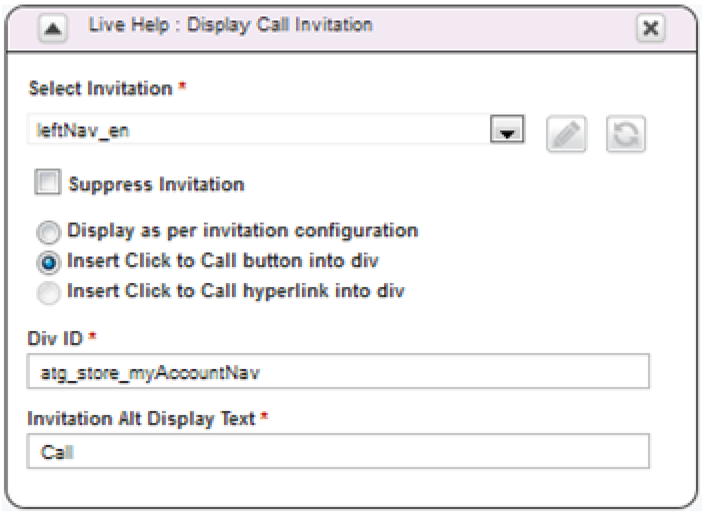
Click-to-Call links live within the HTML of your customer-facing Web page. For example, you can place a static link in one of your existing navigational menus using a `<div>` tag.

When creating a static Click-to-Call link, you create a Timeout Link that has the agent phone number and a time out and close length of zero (0), indicating that the link is displayed whenever the page is rendered.

For detailed information on creating links, refer to the Live Help On Demand Engagement Engine Editor documentation.

Creating a Rule

Once you have create a link, you must create a rule that will display your newly created link. Using the Engagement Engine Editor, create a new rule with the Rule Evaluation Order of 1. Set the Do This Actions to Live Help and Display Call Invitation. You can insert a button by selecting your link and using the Insert Click-to-Call Button in Div radio button. Enter the ID of your `<div>` tag and Alt Display Text.


Live Help : Display Call Invitation

Select Invitation *: leftNav_en

☐ Suppress Invitation

○ Display as per invitation configuration
● Insert Click to Call button into div
○ Insert Click to Call hyperlink into div

Div ID *: atg_store_myAccountNav

Invitation Alt Display Text *: Call

For detailed information on creating links, refer to the Live Help On Demand Engagement Engine Editor documentation.

Creating a Site

After you have set up the links and rules that will display the links for Click-to-Call, you must add a site using the Engagement Engine Editor. The site controls when a page that contains a pre-defined URL, IP address or page title will initiate a rule.

For detailed information on creating and publishing sites, refer to the Live Help On Demand Engagement Engine Editor documentation.

Configuring the Click to Connect Token

Using the Live Help On Demand Engagement Engine Editor create the following rule that will save Click to Connect information and pass the token to Commerce Service Center. This rule configures the following information:

- `clickToConnectSave()` - The `clickToConnectSave()` function is a JavaScript call that is located on each page of your customer-facing site. The function persists both order and token information in Commerce Service Center. The `clickToConnectSave` function is created automatically when you run page instrumentation and does not need to be added manually to every page. However, the `clickToConnectSave` function can also be configured using the Live Help On Demand Engagement Engine Editor.
- `ATGCallToken` - When the customer clicks the Click-to-Call link or button, a unique token ID is generated and stored using the `clickToConnectSave` function. This token, which is stored in the `_atg_estara_call_token` identifies the information that the customer was viewing.

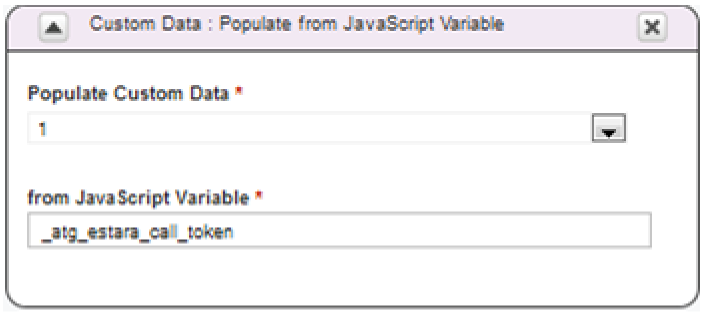
For additional information on creating rules, refer to the Live Help On Demand Engagement Engine Editor documentation.

Creating the Rule

1. Log into Webcare.

   Note: In order to log into Webcare, you must have an existing account.

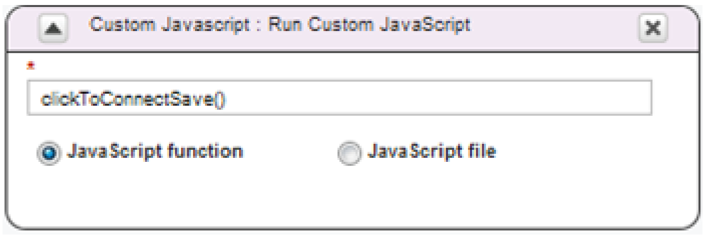
2. From the Setup menu, select Engagement Engine Editor.
3. Click the Add a New Rule button.
4. In the Name field enter `ClickToConnectSave`.
5. Set the Rule Evaluation Order to 2.
6. In the Do This section, select Custom Data and then Populate from JavaScript Variable.
7. Set the Populate Custom Data Dropdown to 1, and enter the name of the call token, which is `_atg_estara_call_token`.



Populate Custom Data *
1

from JavaScript Variable *
_atg_estara_call_token

8. Create another Do This action. Select Custom JavaScript and then Run Custom JavaScript. Enter `clickToConnectSave()` and select the JavaScript Function radio button.



*
clickToConnectSave()

[✓ JavaScript function] JavaScript file

9. Add a condition that sets Live Help: Invitation. Add the Click-to-Call link you created earlier and select Has Been Offered in the drop down menu.

10.Save and then publish the rule in the Engagement Engine Editor.

Example: Creating a Link based on Locale

The following example outlines how to display a French language link to a customer in France. Remember that page instrumentation adds JavaScript variables to your Web page, and one of the variables added is the `_atg_estara_locale` variable, which identifies the customer's locale.

1. Create a link that displays the French call invitation.
2. Use the Webcare Engagement Engine Editor to create a new rule that determines when the French language links are displayed.
3. In the Conditions section, select Web Page Content and then JavaScript Variable.
4. Select Text Variable from the drop down menu, and enter `_atg_estara_locale` to identify the variable, and then `fr_FR` to identify the value.
5. Create a Do This action that selects Live Help and then Display Call Invitation. Select the French link from the Select Invitation drop down menu.
6. Save and deploy the rule.

For detailed information on customizing links and rules, refer to the Live Help On Demand Engagement Engine Editor documentation.

Using Live Help On Demand Agent Console

The Live Help On Demand Agent Console is the agent-facing console that manages agent and customer interactions. The Agent Console can be used with an existing CTI infrastructure to allow your agents to interact with customers.

Integrating with the Live Help On Demand Agent Console

Using client-side integration, a browser window that points to a Commerce Service Center URL is displayed within the Agent Console. Integration between the Agent Console and Commerce Service Center is regulated using a hash key.

To create the browser window that links your Live Help On Demand system to both the Commerce Service Center and Click-to-Call systems, you must define a call integration panel in the Agent Console. Refer to the Live Help On Demand documentation for creating an integration panel.

To point the integration panel to Commerce Service Center, enter the URL of the Commerce Service Center server in the following format:

```
http://CSC_Server_Name:CSC_Server_Port/agent/main.jsp?
clickToCallInit=true&UserTelephoneNumber=[%CALLERNUMBER%]&username=
  [%CSRUSERNAME%]
&estara_fsguid=[%ESTARAFSGUID%]&callid=[%CALLERID%]&hash=[%CSCAUTHTOKEN%]
&token=[%VARIABLEFIELD1%]&referrer=[%ORIGINURL%]
```

Refer to the Oracle Live Help On Demand Agent Console User Guide for detailed information on configuring Commerce Service Center integration with authentication and hash functions.

Integrating without Live Help On Demand Agent Console

When you integrate Click-to-Call and Commerce Service Center with an external CTI system, you need to provide the CTI system with a URL that accesses the Click-to-Call information. Before you can configure the Commerce Service Center portion of the configuration, ensure that your external CTI system can produce a hash that Commerce Service Center will be able to reproduce.

The hash key can be set manually or set in a system repository. By default, hash keys are not available in clear text in a configuration file. Hash keys that are set in a system repository can be obtained using the `C2CTools` property from the Dynamo Server Admin.

One you have the hash key, generate a script. For example:

```
/**
  *
  * Generates the URL that is used to spawn a CSC instance passing the required
  * parameters for a Click To Connect Session
  *
  * @param pURL
  *    The initial part of the CSC URL e.g. http: //foo: 8080/agent/main
  * @param pUsername
  *    The username of the CSC user
  * @param pCallerID
  *    The caller id of the customer
  * @return The fully qualified URL
  *
  */
public URL generateURL(String pURL, String pUsername, String pCallerID) throws \
MalformedURLException, UnsupportedEncodingException {

// hashKey is the secret key used to salt the hash, this secret key is also
//configured within CSC and should be identical to what is used here.
final String hashKey = "mySecretKey";

// hashingAlgorithim is the algorithm used to create the hash. The algorithm
//should be the same as is configured within CSC in order for CSC to recreate the
//hash.
final String hashingAlgoritim = "SHA1";

// hashText is the information that the hash is created from, the Agent username,
//customer telephone number and the hashKey.
String hashText = pUsername + pCallerID + hashKey;
String clickToCallURLString = "";
String hashString = "";

/*
 * Create hash of the username, callerID and the hashKey. The Hashing algorithm
 * used is SHA1, matching the hashing algorithm that CSC is configured to use.
 */
try {
  MessageDigest m = MessageDigest.getInstance(hashingAlgoritim);
  m.update(hashText.getBytes(), 0, hashText.length());
  hashString = new BigInteger(1, m.digest()).toString(16).trim();
} catch (NoSuchAlgorithmException nsae) {
}
/*
 * The URL is created from the initial CSC URL e.g http: //foo: 8080/agent, the
 * clickToCallInit parameter, the Agent username, the hash value, and the
 * telephone number of the shopper who initiated the Click-to-Call session
 */
clickToCallURLString += pURL + "?clickToCallInit=true&username="
  + URLEncoder.encode(pUsername, "UTF-8") + "&hash="
  + URLEncoder.encode(hashString, "UTF-8") + "&UserTelephoneNumber="
  + URLEncoder.encode(pCallerID, "UTF-8");

URL clickToCallURL = new URL(clickToCallURLString);

return clickToCallURL;
}
```

Once you have verified that your CTI system can produce the necessary hash and URL, create a script that integrates your system. The script should:

1. Get the username, user telephone number, secret key, and Commerce Service Center URL variables.
2. Create a hash of the username, user telephone number, and the secret key.
3. Set the `checkFTCallID` property of the `/atg/svc/clicktoconnect/C2CTools` component to `false` on the Commerce Service Center server.
4. Generate a Web browser with a URL made up of the following:

   Commerce Service Center URL (`http://`*`cscserver`*`:` *`port`*`/agent/main.jsp`)

   The `clickToCallInit=true` variable

   - The agent user name
   - The hash made earlier
   - The customer telephone number
5. The following is an example of a URL:

   ```
   http://mycommerceservicecenterserver:8090/main.jsp?clickToCallInit=
   true&username=bsmith&hash=5dd9701wtf3f&estara_fsguid=03891CD515C19&
   callid=newgui_58478%3A122.1.34%2A80%3A05504.2118&
   UserTelephoneNumber=4411411414
   ```

Configuring Commerce Service Center Authentication with the Agent Console

Commerce Service Center authenticates the data request and the agent user ID before transferring any data. Commerce Service Center authentication uses values that are calculated based on values in the request and a secret key value. You specify the secret key value when working with Oracle Support to set up your Oracle On Demand Live Help account.

Note: Ensure that your Webcare account is linked to your Live Help On Demand Agent Console

Enabling Commerce Service Center Auto-Authentication

You can enable automatic authentication by performing the following:

1. You can configure Commerce Service Center to automatically authenticate an agent setting the `atg/dynamo/servlet/daf/pipeline/AuthenticationServlet.properties` file property `enabled` to `true`.

   The `AuthenticationServlet.properties` file also contains the following:

   ```
   # Login parameter name
   loginParamName=username
   # caller ID parameter name, this is the user telephone number
   callerIDParamName=UserTelphoneNumber
   # Unique caller ID, this is not the telephone number
   FTCallIDParamName=called
   # Hash parameter name
   hashParamName=hash
   # salt for the hash that's generated
   hashKey=secretKey
   # enable the AuthenticationServlet
   enabled=true
   # enable the FTCallID check
   checkFTCallID=true
   ```

2. A salted hash is generated by your CTI application or by Live Help On Demand and then verified by Click-to-Call to ensure authenticity. The secret key must be configured identically on both Commerce Service Center and your CTI or Live Help On Demand system. The property `secretKeyForHashCompare` in the `/atg/svc/clicktoconnect/C2CTools` file should be set to the value of this key.

   ```
   # Key that is included when calculating the has value...
   secretKeyForHashCompare=secretKey
   ```

   Note: If no secret key has been generated, Commerce Service Center will generate its own key, which can be found using the Dynamo Server Admin, and selecting `atg/svc/clicktoconnect/C2CTools`.

3. Live Help On Demand agents must be added with an identical login name for both Commerce Service Center and Webcare for automatic authentication. Use the Business Control Center and Webcare to ensure that agent names are the same in both applications. Ensure that you have a Customer Service Representative who has access to Commerce Service Center.

   Note: If you are not using the Agent Console, set the `checkFTCallID` property to `false` in the `/localconfig/atg/svc/clicktoconnect/C2CTools` file.

4. Use Webcare to add the agents. Note that each agent must have access to the Agent Console that is used to route telephone calls on their system. For information on creating Webcare agents, refer to your Live Help On Demand documentation.

Configuring Commerce Service Center Integration

Once you have obtained the secret key, you must configure the authenticated hash information so that Agents can access CSC through the Live Help On Demand Agent Console.

1. Log into Live Help On Demand In-House Administration. For detailed information on working with Live Help On Demand administration, refer to the Live Help On Demand Administration documentation.

2. Search for and then select your account number. In the CSC Integration section, enter the authenticated shared secret key.
3. From the Hash Function drop down menu, select `SHA-1`.
4. Save the changes and log out.

Disabling Commerce Service Center Auto-Authentication

You can disable Commerce Service Center authentication by setting the `enabled` property in the `atg/dynamo/servlet/daf/pipeline/AuthenticaionServlet.properties` to `false`. Once authentication has been disabled, the agent must log in for each session of Commerce Service Center.

Configuring Commerce Service Center Landing Page Components

Landing pages are used to control which page is initially viewed by the agent at the start of the Click-to-Call session. Typically, landing pages are determined based on the URL viewed on the storefront when the call was initiated. However, the Landing Page components provide a generic interface that allows you to determine a landing page based on other values, and not just the storefront URL.

Because most of the time the landing page is determined by the storefront URL at the time the call was initiated by the customer, Commerce Service Center provides a number of `LandingPage` and `LandingPageHandlers` pre-configured to land on specific views within the Commerce Service Center application. This includes a default landing page that is used when no other landing page is provided.

`LandingPage` configurations are available for landing the follow views in Commerce Service Center. Any one of the provided `LandingPages` can be mapped to one or more storefront URLs.

- The shopping cart view of the active order
- The customer profile view of the active customer
- The existing order view for a given order
- The product view for a given product
- The main catalog browse view

Default Landing Page

The Default Landing Page determines which page is initially viewed when no other landing page can be determined. By default, the Default Landing Page is configured with the shopping cart view.

Customizing Landing Page Components

You can change the default tab or panel that is presented to the agent when Click-to-Call presents a new call. The following section describes the components and configurations for this feature.

The landing page configuration consists of the `LandingPageManager`, the `LandingPageHandler`, and the `LandingPage` components.

The LandingPageManager Component

The `/atg/svc/clicktoconnect/LandingPageManager` component contains the configurations for all of the `LandingPageHandlers` and executes them to determine the correct landing page for the Click-to-Call request. It polls all `LandingPageHandlers` until one of them returns a `LandingPage` object. If no landing page is returned, the `defaultLandingPageHandler` is used.

These `LandingPageManager` properties are configurable as follows. If the `LandingPageManager` does not successfully determine a `LandingPage`, the agent's normal login landing page will be the default landing page:

- `landingPageHandlers` – an array of `LandingPageHandler` components
- `defaultLandingPageHandler` – the `LandingPageHandler` that is executed when no other `LandingPageHandler` has provided a landing page

For example:

```
landingPageHandlers+=
  /atg/commerce/custsvc/clicktoconnect/ProductViewLandingPageHandler,\
  /atg/commerce/custsvc/clicktoconnect/OrderViewLandingPageHandler,\
  /atg/commerce/custsvc/clicktoconnect/CategoryViewLandingPageHandler
defaultLandingPageHandler=/atg/commerce/custsvc/clicktoconnect/
    DefaultLandingPageHandler
```

The LandingPageHandler Components

The `LandingPageHandler` components, which are integrated with `LandingPageManager`, return a `LandingPage` based on the values of the `Click-to-CallRequestData` object.

The following `LandingPageHandler` properties are configurable:

- `landingPage` – the `landingPage` object that is returned by the `LandingPageHandler`
- `URIMatches` – this property is available to `LandingPageHandlers` that determine the landing page based on the current storefront URL

  Note: When configuring the `URIMatches` property, you must supply the full file path, for example `/commerce2/myaccount/profile.jsp`.

Component	Description
`/atg/svc/clicktoconnect/ProfileViewLandingPageHandler`	This component returns a landing page if the storefront URL starts with any one the strings specified by the `URIMatches` property. The profile viewed will be the same as the profile saved when initializing the call on the storefront. For example: `URIMatches=/ondemand/myaccount/profile.jsp`

Component	Description
`/atg/commerce/custsvc/clicktoconnect/CategoryViewLandingPageHandler`	This component returns a landing page if the storefront URL starts with any one the strings specified by the `URIMatches` property. This component will land the agent on the catalog browse main page. For example: `URIMatches=/ondemand/subcategory.jsp?categoryId=`
`/atg/commerce/custsvc/clicktoconnect/OrderViewLandingPageHandler`	This component returns a landing page if the storefront URL starts with any one the strings specified by the `URIMatches` property. The order ID is identified by a parameter on the storefront URL with the name specified by the property `orderIdParameterName`. This component has a `LandingPage` configuration for standard orders and another for scheduled orders. For example: `URIMatches=/ondemand/myaccount/myOrders.jsp`
`/atg/commerce/custsvc/clicktoconnect/ProductViewLandingPageHandler`	This component returns a landing page if the storefront URL starts with any one the strings specified by the `URIMatches` property. The product ID is identified by a parameter on the storefront URL with the name specified by the property `productIdParameterName`. For example: `URIMatches=/ondemand/productDetailWithPicker.jsp?productId=`

`/atg/commerce/custsvc/clicktoconnect/DefaultLandingPageHandler`	This component returns a landing page when no other landing page is provided. If the order is modifiable, the cart view page is used, otherwise the order view page is used.
	
	For example:
	`scheduledOrderViewLandingPage=ScheduledOrderViewLandingPage`
	
	`orderViewLandingPage=OrderViewLandingPage`
	`cartViewLandingPage=CartViewLandingPage`

The LandingPage Component

A `LandingPage` object provides all of the information required for the `LandinPageManager` to initialize the agent UI on a specific view. Commerce Service Center provides a number of pre-configuration `LandingPage` components that are used by the `LandingPageHandlers`. However, `LandingPage` objects can be constructed dynamically at runtime.

The following properties are configurable in the `LandingPage` component:

- `tabId`
- `panelStackIds`
- `panelIds`

- `dynamicIncludes`
- `treeTableIds`

The following `LandingPage` components can be configured:

Component	Description
`/atg/svc/clicktoconnect/ProfileViewLandingPage`	Identifies the values required to land on the customer tab’s profile view page.
`/atg/commerce/custsvc/clicktoconnect/CategoryViewLandingPage`	Identifies the values required to land on the commerce tab’s catalog view page.
`/atg/commerce/custsvc/clicktoconnect/ProductViewLandingPage`	Identifies the values required to land on the commerce tab’s product view page.
`/atg/commerce/custsvc/clicktoconnect/OrderViewLandingPage`	Provides the values needed to land on the commerce tab’s order view page.
`/atg/commerce/custsvc/clicktoconnect/ScheduledOrderViewLandingPage`	Identifies the values needed to land on the commerce tab’s scheduled order view page.
`/atg/commerce/custsvc/clicktoconnect/CartViewLandingPage`	Provides the values required to land on the commerce tab’s cart page.

Index