Oracle® Commerce Retail Extension Module

Implementation Guide
Release 16.0
E79486-01

December 2016

Oracle® Commerce Retail Extension Module Implementation Guide, Release 16.0

E79485-01-01

Primary Author: Judith Meskill

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

(i) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(ii) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Mobile Store Inventory Management.

(iii) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(iv) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

Send Us Your Comments

Oracle Commerce Retail Extension Module Implementation Guide, Release 16.0

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

Note: Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the Online Documentation available on the Oracle Technology Network Web site. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

This Implementation Guide provides detailed information that can be useful for implementing and configuring the application.

Audience

This guide is for system administrators.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Commerce Retail Extension Module Release 16.0 documentation set:

- *Oracle Commerce Retail Extension Module Installation Guide*
- *Oracle Commerce Retail Extension Module Merchandising Implementation Guide*
- *Oracle Commerce Retail Extension Module Release Notes*
- *Oracle Commerce Retail Extension Module Security Guide*

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Improved Process for Oracle Retail Documentation Corrections

To more quickly address critical corrections to Oracle Retail documentation content, Oracle Retail documentation may be republished whenever a critical correction is needed. For critical corrections, the republication of an Oracle Retail document may at times **not** be attached to a numbered software release; instead, the Oracle Retail document will simply be replaced on the Oracle Technology Network Web site, or, in the case of Data Models, to the applicable My Oracle Support Documentation container where they reside.

An updated version of the applicable Oracle Retail document is indicated by Oracle part number, as well as print date (month and year). An updated version uses the same part number, with a higher-numbered suffix. For example, part number E123456-**02** is an updated version of a document with part number E123456-**01**.

If a more recent version of a document is available, that version supersedes all previous versions.

Oracle Retail Documentation on the Oracle Technology Network

Oracle Retail product documentation is available on the following web site:

`http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html`

Conventions

Navigate: This is a navigate statement. It tells you how to get to the start of the procedure and ends with a screen shot of the starting point and the statement “the Window Name window opens.”

```
This is a code sample
    It is used to display examples of code
```

1

Introduction

This chapter introduces Oracle Commerce Retail Extension Module (RXM).

RXM is a module for Oracle Commerce (OC) that converges store shopping concepts with e-commerce and provides commerce information to the stores. It requires Oracle Commerce, which is a platform that is modularized for high-extensibility. Oracle Commerce's module framework stacks like layers. RXM, as a Commerce module, is located between the base Commerce platform below and custom merchant modifications above. Layers above add to and override modules beneath. RXM layers above the DCS module, which defines the base e-commerce domain and functions. Any additional extensions and configurations required by the merchant's unique domain in order to build their webstore UI are also layered on the very top. However, with RXM, the merchant's module must extend RXM classes and override the RXM configuration, where it exists.

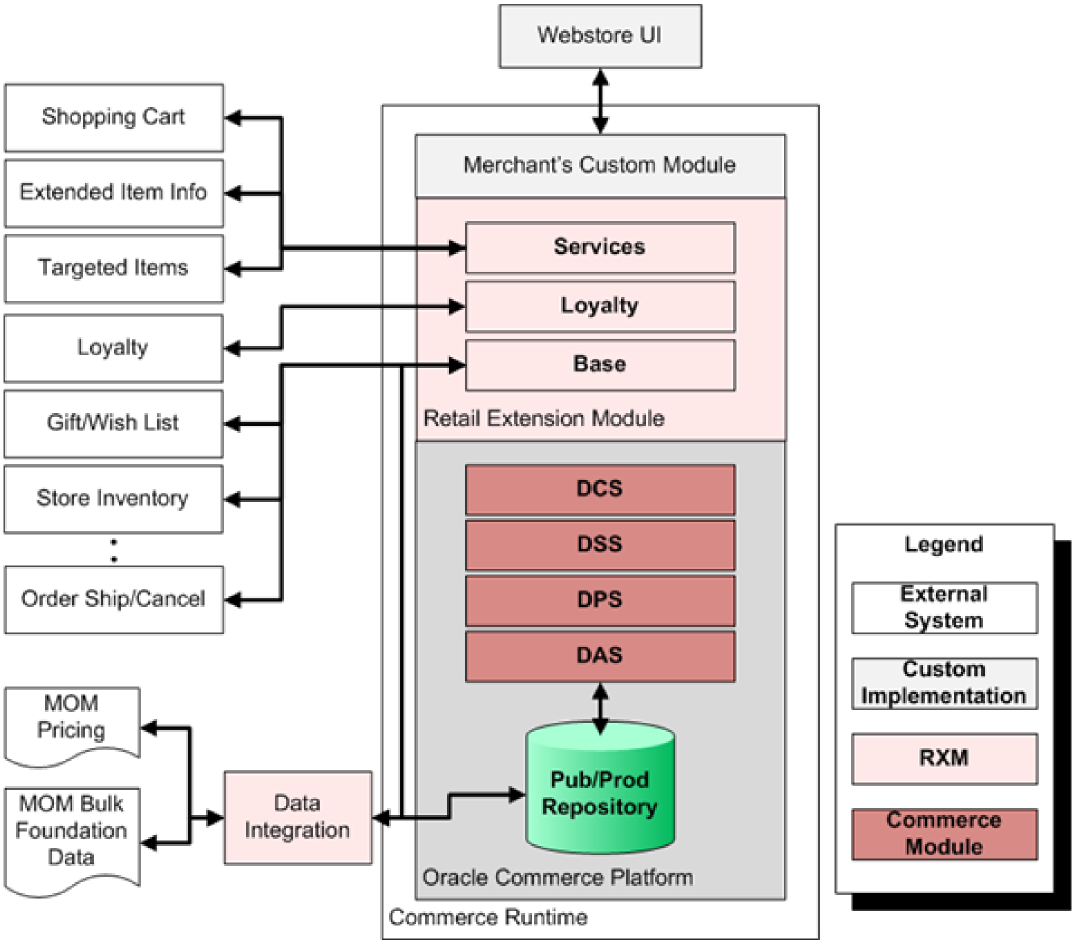

Figure 1: RXM Module

In Figure 1, RXM is divided into separate modules for Base features, Loyalty features, and Services provided to Xstore. This separation is for configuring OC for different deployments if system administrators want to separate loads or responsibility.

2

Design Approach

RXM provides integration with other retail systems. Out-of-the-box, these systems are expected to be Oracle Retail assets, such as Oracle Retail Customer Engagement, Oracle Retail Order Broker, Oracle Retail Order Management System, Oracle Retail Xstore Point of Service, and Merchandise Operations Management.

Many of RXM's integrations are SOAP web services. These web services are actually defined by the Oracle Retail Service Backbone (RSB). RSB is built upon Oracle Service Bus, which is an application for decoupling and virtualizing the merchant's enterprise applications. RSB comes as part of the Oracle Retail Integration Bus (RIB). RSB provides decorator paks for each endpoint application. These packs decorate and transform the service operation payloads. A pack for each endpoint application is required.

RSB provides not just abstraction; it also provides transformation between RSB's service definitions and the provider applications' definitions. Retailers are able to edit this transformation as well as extend existing service operations or define new service operations. Since RSB is built upon Oracle Service Bus (OSB) 12c, these development tasks are done using JDeveloper, which is bundled with OSB. For additional information on performing modifications to 16.0 RSB services that RXM uses, see the *Oracle Retail Service Backbone Development Guide.*

In regards to RXM as a consumer of SOAP web services, there is no integration framework for this in base Oracle Commerce. RXM uses Apache Camel paired with Apache CXF for decoupling the integration from the RXM application code and making the actual SOAP connections.

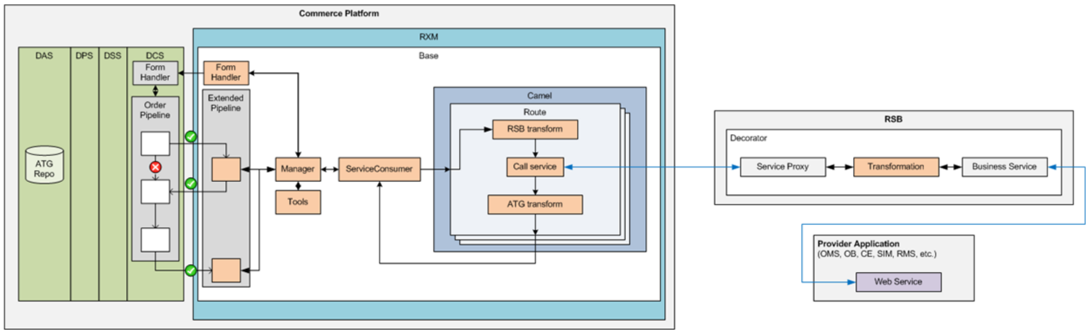

Figure 2: RXM Accessing SOAP

Figure 2 depicts some of the common ways RXM accesses a SOAP service. For example, using typical Oracle Commerce extension mechanisms, order processing through the order pipeline uses pipeline processors added for RXM. These processors can use a Manager object, which can access a service through a ServiceConsumer façade. Additionally, it is also possible for FormHandlers to access Managers directly.

ServiceConsumers

The ServiceConsumers are the facades that are responsible for separating business code from the Apache Camel context and its processing. Application code is not expected to call a consumer directly but it can. Usually the calls are handled by the manager. Each method call to the consumer is passed as a message to the Camel context, which is completely XML configurable. The context defines the message processing steps. Typical steps are: before connecting the service, the message is transformed to an RSB payload. Then, Camel uses CXF to call the SOAP service hosted at RSB. RSB also has its own transformation step to transform from the RSB payload to the provider application payload. Then it is handled by the provider.

Managers

Managers should be a familiar concept for Oracle Commerce implementers. RXM extends a few managers where needed (see subclasses of AbstractInventoryManager, CommerceItemManager, GiftlistManager, and OrderManager). Generally, they are used to provide methods that have business logic or access ServiceConsumers or Tools classes.

Tools

Tools should also be a familiar concept from base Oracle Commerce. In general, RXM created Tools classes where a utility must be performed around Repository access or manipulation. See subclasses of Tools for CatalogTools, OrderTools, GiftlistTools, CommerceProfileTools, and PricingTools.

3

Commerce Extensions

Using the pattern described in chapter 2 and other available Commerce frameworks, RXM extends the following functional areas.

RXM Profile Extensions

RXM enhances Oracle Commerce base user profile framework with synchronization to an external customer management system to use as a system of record. The default external application used for RXM 16.0 is Oracle Retail Customer Engagement (ORCE). Using RXM form handlers, profiles are synchronized upon login and updates using Oracle Commerce's atg.service.datacollection.DataCollector framework.

Available API

A large amount of profile API is concentrated in the component found at /retail/commerce/profile/ProfileManager. The implementing class implements this interface, oracle.retail.commerce.profile.ProfileManager. The manager is the primary API where clients can access the SOAP service consumer calls to ORCE. Through this manager interface, clients have access to the following:

- createCustomer(RepositoryItem)
- createCustomerAsync(RepositoryItem)
- mergeCustomer(RepositoryItem)

The component /atg/userprofiling/ProfileTools is updated with a new implementing class, oracle.retail.commerce.profile.RetailCommerceProfileTools. This class provides the ability to:

- addEmailAddress(RepositoryItem, String)
- addPhoneNumber(RepositoryItem, String)
- getExternalCustomerId(RepositoryItem)
- getExternalCustomerId(String)
- getProfileItemByCustomerId(String)

This tools class allows for managing the additional emails and phone numbers for a profile and dealing with a profile's external customer ID.

Commerce Extensions

The Oracle Commerce class atg.core.util.ContactInfo, which contains address information, is extended by oracle.retail.commerce.profile in order to provide an address name, type, external ID, and whether it is a primary address.

The oracle.retail.commerce.profile.RetailCommercePropertyManager is an extension to the base Oracle Commerce property manager in order to provide additional RXM property names for profiles.

Login and Create Handler

When profiles are created or accessed, RXM will retrieve information from ORCE and merge it with the profile if it is a known customer. This is done by adding the /retail/commerce/profile/event/SynchronizingProfileSwapEventListener component to the "swapEventListeners" property in Commerce's /atg/userprofiling/ProfileFormHandler component. Commerce sends an event upon login and create actions. This listener handles the event to use the ProfileManager to synchronize the information.

Repository Listener

When profile repository items are updated, RXM will notify ORCE about changes to the profile so that it can update its customer record. This is done through the /retail/commerce/profile/event/ProfileChangesCollector component using the profile manager and tools. This component is wired into the atg.repository.PropertiesChangedListenerService at /retail/commerce/profile/event/ProfileDataEventListener. This listener is configured to only listen for certain properties to change, for example firstName, lastName, and billingAddress. This listener is one of the initial services started by the RXM module.

Customer Transformer

RXM synchronizes customer and profile information using an external system. This system is accessed through SOAP web services hosted on the Retail Service Backbone (RSB). The profile components, such as the profile manager, access the service operations through the ProfileServiceConsumer component.

The service consumer uses the Apache Camel context for profile services defined at /retail/commerce/integration/profile/profile-context.xml. The context routes are configured to make SOAP calls to the RSB. Before and after each SOAP call, the payload of the route must be transformed. Primarily this is performed by the component at /retail/commerce/integration/profile/ProfileServiceTransformer. Methods in this component transform the profile and contact information repository objects to and from the JAXB objects in com.oracle.retail.integration.base.bo.customerdesc.v1 required by the RSB. However, in some cases, you will find examples where it is easier to use Camel's <simple> domain specific language to perform a transform or an object mutation. For example:

```
<!-- Transform RSB message to external customer id. -->
<transform><simple>${body[0].customerId}</simple></transform>
```

In the above example, the response body is transformed into just the customer ID string.

```
<!-- Invoke query customer RSB service -->
<doTry>
    <to uri="cxf:bean:CustomerServiceEndpoint?loggingFeatureEnabled=false" />
    <!-- Transform RSB message to response. -->
    <bean ref="ProfileServiceTransformer" method="toCustomerIds" />
    <doCatch>
        <exception>org.apache.cxf.binding.soap.SoapFault</exception>
        <onWhen>
            <simple>${exception.message} contains 'customer was not
found'</simple>
        </onWhen>
        <log message="No customer was found, catching SoapFault and returning
empty array of string"/>
        <!-- Set body to empty string array -->
        <setBody>
            <groovy>new String[0]</groovy>
        </setBody>
    </doCatch>
</doTry>
```

In the above example, SoapFaults from ORCE are caught. If they contain the phrase "customer was not found", then the fault is ignored and an empty string response is set into the Camel message body, which is returned by the ProfileServiceConsumer.

The transformer is specific to using the expected RSB web services, and in some cases, ORCE service provider behavior is also expected and coded for. For integrating to other non-RSB web services, the transformer component's class should be replaced with a custom implementation and the Camel route updated with appropriate transform calls. For services abstracted behind the RSB that behave slightly differently from ORCE, it is best to extend oracle.retail.commerce.integration.profile.ProfileServiceTransformer and update the component with the extended class.

ORCE Support

RXM integrates with ORCE Customer Services version 3.0. RXM is expected to communicate with ORCE through the RSB. The RSB application provides application-specific PAKs that contain additional transformation through XSLT transformations. Development or changes to these transforms requires Oracle Service Bus (OSB) with JDeveloper. See *Oracle Retail Service Backbone Development Guide* for information about how to develop with these paks.

Security Considerations

New profile synchronization functions have no new access roles defined.

Accessing remote loyalty service operations must be secured by default. Security properties are loaded by the loyalty Camel context. See the “Securing Commerce Extensions” section for additional information on these settings.

Customization

The RXM module takes advantage of both Nucleus and Spring to allow for easy customization and dependency injection. Class names can be found in the profile-context.xml as well as the various component properties files. See the *Oracle Commerce Platform Development Guide* for more information on how to customize Nucleus components.

RXM Loyalty Extensions

RXM adds loyalty functionality to Oracle Commerce using an external loyalty management system as the system of record. The default external application used for RXM 16.0 is Oracle Retail Customer Engagement (ORCE).

Web store developers can access most loyalty functionality though the LoyaltyManager API.

Available API

The primary access point for the loyalty API is through the component found at /retail/commerce/loyalty/LoyaltyManager. The implementing class implements this interface, oracle.retail.commerce.loyalty.LoyaltyManager. Through this interface, clients have access to:

- applySelectedAwards(RetailOrder, AwardCoupon...)
- calculatePoints(RetailOrder)
- createLoyaltyAccount(Profile)
- recoverPoints(Profile, RetailOrder)

- recoverPoints(Profile, String)
- retrieveLoyalty(String)
- retrieveLoyalty(String, boolean)

RXM provides a basic form handler that can be found at /retail/commerce/loyalty/LoyaltyFormHandler. The implementing class is oracle.retail.commerce.loyalty.LoyaltyFormHandler. This class uses LoyaltyManager and OrderManager, among others. It provides the ability to:

- handleOptIn(DynamoHttpServletRequest, DynamoHttpServletResponse)
- handleRecoverPointsForOrder(DynamoHttpServletRequest, DynamoHttpServletResponse)
- handleRecoverPointsForOrderId(DynamoHttpServletRequest, DynamoHttpServletResponse)

There is also a form handler at /retail/commerce/loyalty/LoyaltyAwardFormHandler that can:

- handleAddAwardsToOrder(DynamoHttpServletRequest, DynamoHttpServletResponse)

Two drops are available to make estimating points and account lookups easier. See /retail/commerce/loyalty/droplet/LoyaltyAccountLookupDroplet and /retail/commerce/loyalty/droplet/EstimateLoyaltyPointsDroplet.

The RXM.Loyalty module has a subclass OrderTools called oracle.retail.commerce.loyalty.LoyaltyOrderTools that can be found at /atg/commerce/order/OrderTools. It handles applying awards and estimated points to orders.

Commerce Extensions

RXM.Loyalty extends the order repository item in order to persist "estimatedLoyaltyPoints" repository items and multiple award repository items.

As part of the updateOrder and refresh order pipelines in commercepipeline.xml, additional processors are added to save and load the loyalty awards and points for orders.

The user repository item is extended to include a loyaltyCardNumber property for saving the profile's connection to the loyalty system.

Order pricing is updated using these new calculators. In /atg/commerce/pricing/OrderPricingEngine, the /retail/commerce/loyalty/pricing/calculators/LoyaltyEstimateResetCalculator and /retail/commerce/loyalty/pricing/calculators/LoyaltyDiscountCalculator are added. The first removes any previous estimate from the order upon a price recalculation in order to avoid the use of stale points. The second calculator inspects the order for awards that have been applied and discounts the order total by the award's specified amount.

Loyalty Transformer

RXM.Loyalty expects most of its functionality to be backed by an external system. This system is accessed through SOAP web services hosted on the Retail Service Backbone (RSB). The loyalty components, such as the loyalty manager, access the service operations through the LoyaltyServiceConsumer component.

The service consumer uses the Apache Camel context defined for loyalty services defined at /retail/commerce/integration/loyalty/loyalty-context.xml. The context routes are configured to make SOAP calls to the RSB. Before and after each SOAP call, the payload of the route must be transformed. Primarily this is performed by the component at /retail/commerce/integration/loyalty/LoyaltyServiceTransformer. Methods in this component transform the object model in oracle.retail.commerce.loyalty to and from the JAXB objects in com.oracle.retail.integration.base.bo.loyacctdesc.v1 required by the RSB. However, in some cases, you will find examples where it is easier to use Camel's <simple> domain specific language to perform a transform or an object mutation. For example:

```
<when>
    <!-- If order doesn't have id, set to zero to avoid ORCE rejecting it. -->
    <simple>${body.customerOrderId} == null ||
${body.customerOrderId.empty}</simple>
    <setBody>
        <groovy>request.body.customerOrderId = '0'
            return request.body
        </groovy>
    </setBody>
</when>
```

In the above example, the request body is set with an order ID of "0". Not only is the element required by the RSB WSDL, but it has to be numerical due to ORCE's expectations.

The transformer is specific to the use of the expected RSB web services, and in some cases, ORCE service provider behavior is also expected and coded for. For integrating with other non-RSB web services, the transformer component's class must be replaced with a custom implementation and the Camel route updated with appropriate transform calls. For services abstracted behind the RSB that behave slightly differently from ORCE, it is best to extend oracle.retail.commerce.integration.loyalty.LoyaltyServiceTransformer and update the component with the extended class.

ORCE Support

For Loyalty functionality, RXM uses the following ORCE services:

- Card Services 3.1
- Loyalty Account Services 3.1
- Stored Value Card Transaction Services 3.1

RXM is expected to communicate with ORCE through the RSB. The RSB application provides application-specific PAKs that contain additional transformation through XSLT transformations. Development or changes to these transformations requires Oracle Service Bus (OSB) with JDeveloper. *See Oracle Retail Service Backbone Development Guide* for how to develop with these paks.

Note: ORCE can be configured to award fractional loyalty points for things that have partial units such as a half pound of chocolate. RXM only handles whole points. The partial points awarded by ORCE are rounded down by RXM.

Security Considerations

Loyalty functions have no new access roles defined.

Accessing remote loyalty service operations must be secured by default. Security properties are loaded by the loyalty Camel context. See “Securing Commerce Extensions” section for additional information on these settings.

Customization

The RXM.Loyalty module takes advantage of both Nucleus and Spring for easy customization and dependency injection. Class names can be found in the loyalty-context.xml as well as the various component properties files. See the *Oracle Commerce Platform Development Guide* for more information on how to customize Nucleus components.

RXM Shopper Lists Extensions

The RXM shopper list extensions are designed to minimize impact to existing users of Commerce Gift and Wish lists.

Most extensions to the Commerce Gift Wish lists do not change the Commerce base class method signatures and can be used simply by updating the appropriate Commerce component property file's "$class" property and adding a few new properties.

Versions

RXM shopper lists extensions have been tested with the following versions of ORCE APIs:

ORCE Registry API version: 3.0

ORCE Customer API version: 3.0

Pipeline Extensions

No pipeline extensions have been extended or modified for RXM Shopper Lists.

Data Extensions

Two properties have been added to the Commerce gift-list item.

- **externalGiftlistID** - This field associates the Commerce gift/wish list with the external ORCE registry list.
- **expirationDate** - ORCE supports and requires an expiration date. A default offset can be set for this field if this field is not needed by the application.

The application can extend Commerce/RXM gift/wish lists repository objects using the standard Commerce data extension model.

See files: /config/atg/commerce/gifts/giftlists.xml and /sql/install/oracle/create_rxm_schema_ddl.sql

Class Extensions

The following RXM classes extend Commerce to keep Commerce and ORCE gift list and wish list data synchronized.

- RetailGiftlistTools extends atg.commerce.gifts.GiftlistTools
- RetailGiftlistManager extends atg.commerce.gifts.GiftlistManager
- RetailGiftlistSearchFormHandler extends atg.commerce.gifts.SearchFormHandler

The implementers can extend the above classes if additional properties are required or if different method behavior is desired.

RetailGiftlistTools extends atg.commerce.gifts.GiftlistTools

- **findCustomerForWishList()** Several other similar type tool methods are also contained in this class.

- **updateGiftlist()** Two overloaded updateGiftlist() methods used to update the RXM extended properties external gift list ID and expiration date.

RetailGiftlistManager extends atg.commerce.gifts.GiftlistManager

- **createGiftlist()** These methods are required to handle the saving of the external gift list ID (ORCE's gift list id). Several overloaded methods are provided to support expirationDate and the external gift list ID.
- **searchGiftlists()** This method is used to perform an ORCE gift list search. This is called before the gift list search form searches the Commerce repository. In this way the Commerce repository is populated with any matching gift lists from ORCE that did not exist in Commerce.
- **syncGiftlists()** This method is used to synchronize a customer's gift lists and wish list. The application must decide when is best to call this method.

RetailGiftlistSearchFormHandler extends atg.commerce.gifts.SearchFormHandler

- **preSearch()** This overrides the base class method and performs a "pre fetch" of matching gift lists from ORCE. This ensures the base class's doSearch() will find the most up-to-date set of gift lists from both the Commerce repository and ORCE.

Repository Listeners

Two repository listeners are part of RXM Shopper Lists. These listeners send modified properties to ORCE in near real time.

- GiftlistItemChangesCollector
- GiftlistChangesCollector

These classes can be extended if custom behavior is desired by the application.

This design sends create and update requests to ORCE when Commerce detects changes. The application may choose to implement a different mechanism that may, for example, queue up some changes and then send bulk updates to ORCE.

GiftlistServiceConsumer

The GiftlistServiceConsumer is the RXM interface to the external system of record, ORCE.

RetailGiftlistManager and the repository listeners use this interface to communicate with ORCE (through RSB).

This interface uses Camel as a router to the RSB web services.

The camel context file, giftlist-context.xml, maps the interfaces in GiftlistServiceConsumer to the specific transformers and RSB services for each gift list operation.

Shopper List Transformers

Below is the list of transformers used to transform Commerce Giftlist repository items to and from the standard RBO payload.

Each RBO gift list service has a request and result transformer (except for those results which are simply a status or a gift list ID).

Each of these methods then calls "to" methods to transform the various classes. For example, the createGiftListRequest() calls the following methods to transform all the various Commerce repository properties into the RBO payload objects.

```
public GiftListCreModVo createGiftListRequest(RepositoryItem giftlist)
```

```
{
    GiftListCreModVo giftlistCreModVo = toGiftListCreModVo(giftlist);
    GeoAddrDesc geoAddrDesc = toGeoAddrDesc(giftlist);
    if (geoAddrDesc != null)
    {
        ShippingAddrCreVo shippingAddrCreVo = new ShippingAddrCreVo();
        shippingAddrCreVo.setGeoAddrDesc(geoAddrDesc);
        giftlistCreModVo.setShippingAddrCreVo(shippingAddrCreVo);
    }
    return giftlistCreModVo;
}
```

Class: GiftlistServiceTransformer

- createGiftListRequest
- retrieveGiftListRequest
- retrieveWishListRequest
- retrieveGiftListResults
- retrieveGiftListItemsResults
- retrieveWishListResults
- retrieveWishListItemsResults
- updateGiftListRequest
- addWishListItemsRequest
- addGiftListItemsRequest
- deleteGiftListItemsRequest
- deleteGiftListRequest
- updateGiftListItemRequest
- deleteWishListItemsRequest
- queryGiftListsRequest
- queryGiftListsResults

Note: Whenever a transformer updates a gift list or gift list item, the transformer signals to the gift list change collector to not send an update request to ORCE for the next update for that particular update. This is to avoid sending unneeded update requests to ORCE during retrieve operations that merge changes made to ORCE into the Commerce repository. See Java Docs for GiftlistChangesCollector.ignoreNextChange().

RSB XSLT Transformers

Below is a list of RSB XSLT transformers used to transform the RBO payloads into the format required by ORCE.

Note that ORCE requires a "securityUserId" in all requests. The user ID must be defined by the application's ORCE administrator and the default ID "RXM" must be replaced in the XSLT transformers.

- AddGiftListItems.xsl
- AddWishListItemsResponse.xsl
- AddWishListItems.xsl
- CreateGiftListToAddOrUpdateRegistryResponse.xsl

- CreateGiftListToAddOrUpdateRegistry.xsl
- DeleteGiftListItems.xsl
- DeleteGiftListResponse.xsl
- DeleteGiftList.xsl
- DeleteWishListItemsResponse.xsl
- DeleteWishListItems.xsl
- QueryGiftListResponse.xsl
- QueryGiftList.xsl
- RetrieveGiftListItemsResponse.xsl
- RetrieveGiftListItems.xsl
- RetrieveGiftListResponse.xsl
- RetrieveGiftList.xsl
- RetrieveWishListItemsResponse.xsl
- RetrieveWishListItems.xsl
- UpdateGiftListDetailResponse.xsl
- UpdateGiftListItems.xsl
- UpdateGiftListResponse.xsl
- UpdateGiftList.xsl

Shopper List Sequence Diagrams

Create Gift List

Note that the Commerce gift list creation triggers an event in the change collector. The change collector then calls the consumer/camel/transformers to create the ORCE gift list. The external gift list ID is returned and saved.

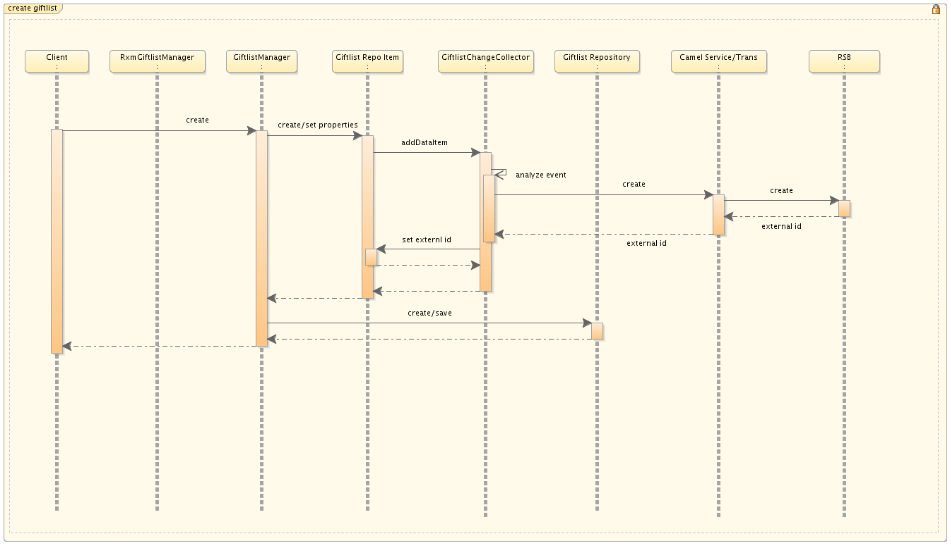

Figure 3: Create Gift List

Update Gift List

Any update to a gift list triggers an event in the change collector. The event then sends the update to RSB/ORCE.

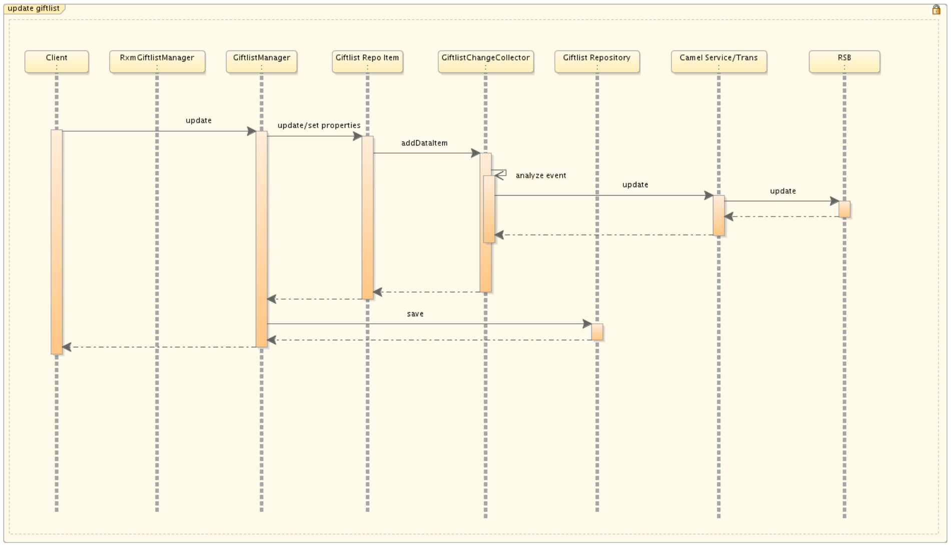

Figure 4: Update Gift List

Update Gift List Item

Note that gift list item changes trigger events in the GiftlistItemChangeCollector. Adding a gift list item to a gift list (not this sequence) triggers an event on the GiftlistChangeCollector.

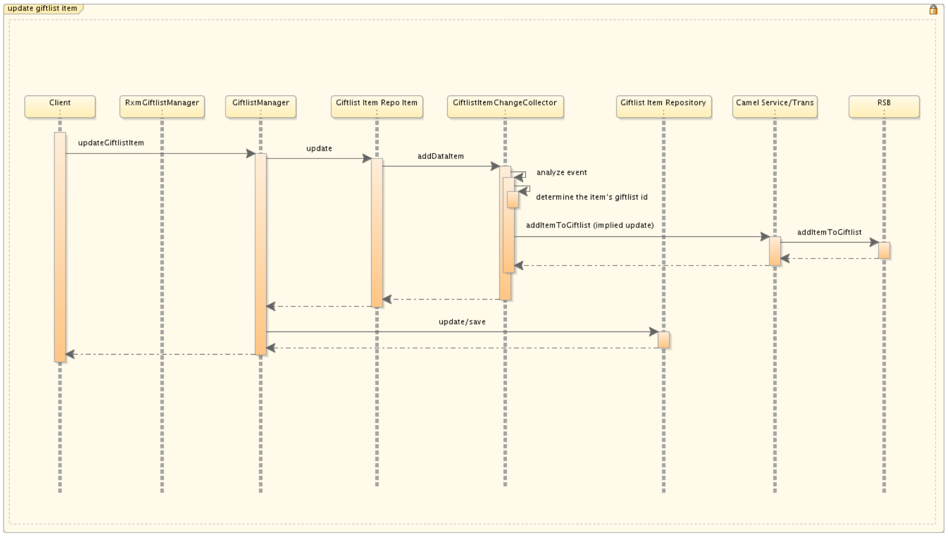

Figure 5: Update Gift List Item

Gift List Query

In this diagram a query request is sent to RSB/ORCE. A gift list is returned that is not found in the Commerce repository. The transformer creates the gift list in Commerce before returning the search results to the form handler.

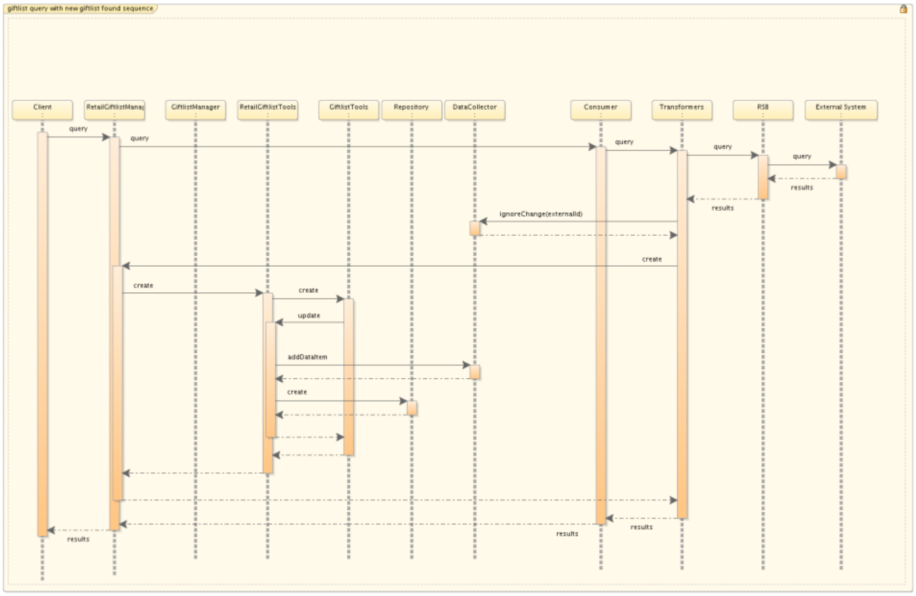

Figure 6: Gift List Query

Differences Between the Commerce/RXM and ORCE Shopper Lists

Gift List versus Registry

In version 3.0 of the ORCE APIs, gift lists are called registries and are accessible using the ORCE Registry API.

Wish Lists

In version 3.0 of the ORCE APIs, wish lists are fairly limited in function and are available using the ORCE Customer API. ORCE wish lists are available using the ORCE item visualizer API.

ORCE wish lists do not have a quantity associated with the item.

In later ORCE API revisions, wish lists behave more like registries. This level of API is not supported by RXM 16.0.

Published versus Private

An unpublished registry in ORCE is considered a private gift list in terms of Commerce nomenclature.

Addresses

Commerce supports a single shipping address for each gift list.

ORCE supports three event addresses and one contact address for each owner of the event.

ORCE gift list contact addresses/phones/emails are not supported by RXM.

Only the ORCE gift list's "before event shipping address" address is supported. This address is used for the Commerce gift list shipping address.

Gift List Event Types

Gift lists have an event type such as birthday and wedding.

The default implementations of Commerce, RSB, and ORCE all have unique event types.

These types are translated in the RXM translators and RSB XSLTs.

It is the responsibility of the application to define the supported types and configure Commerce, RSB, and ORCE to support the same set of event types. The RXM translators and RSB XSLTs must be updated to work properly with the application defined types.

The RXM translators have externalized the conversion between the types into /config/retail/commerce/integration/giftlist/GiftlistServiceTransformer.properties

Synchronizing a Customer's Shopper Lists

Repository listeners are used to detect Commerce shopper list modifications and to send those changes to ORCE in near real time.

Real time shopper list updates from ORCE to Commerce are not supported.

Whenever a shopper list is retrieved from Commerce using the RetailGiftlistManager class, the latest version of that shopper list is retrieved from ORCE and the latest version of the shopper list is returned.

A syncGiftlist() method is provided in RetailGiftlistManager that synchronizes all gift or wish lists for a specific customer.

An application may desire to synchronize a customer's gift lists whenever the customer logs in. To do this:

1. Create a class that extends atg.commerce.profile.CommerceProfileFormHandler
2. Add a component properties file that initializes the class extension (typically located at /config/atg/userprofiling/ProfileFormHandler.properties)
3. Override handleLogin() with something like the following and provide getter/setter for giftlistManager.

```
@Override
public boolean handleLogin(DynamoHttpServletRequest pRequest,
DynamoHttpServletResponse pResponse)
throws ServletException, IOException
{
boolean redirect = super.handleLogin(pRequest, pResponse);
if (getProfile() != null)
{
giftlistManager.syncGiftlists(getProfile());
}
return redirect;
}
```

A customer may desire to synchronize a gift list just before the editing it with a form handler. To do this:

1. Create a class that extends atg.repository.servlet.ItemLookupDroplet.
2. Add a component properties file that initializes the class extension (typically located at /atg/commerce/gifts/GiftlistLookupDroplet.properties).
3. Override service() with something like the following and provide getter/setter for giftlistManager.

```
@Override
public void service(DynamoHttpServletRequest pRequest, DynamoHttpServletResponse
pResponse) throws ServletException, IOException
{
String giftlistId = pRequest.getParameter(ID_PARAM);

RepositoryItem giftList = null;
try
{
giftList = giftlistManager.getGiftlist(giftlistId);
}
catch (CommerceException e)
{
vlogError("Exception occurred while retrieving giftlist {0} from external
system.", giftlistId, e);
}
if (giftList == null)
{
vlogError("Could not retrieve giftlist {0} from external system.", giftlistId);
}

super.service(pRequest, pResponse);
}
```

Use the droplet in your jsp

```
<dsp:droplet name="GiftitemLookupDroplet">
<dsp:param name="id" param="giftId"/>
<dsp:param name="elementName" value="giftitem"/>

...

</dsp:droplet>
```

Gift List Queries

The Commerce Gift List search operation searches the Commerce repository and returns the results as a collection of RepositoryItems.

RXM extends the atg.commerce.gifts.SearchFormHandler class and provides a form handler that "pre fetches" the search results from ORCE and synchronizes the results with the Commerce repository. In this way the results are available in the Commerce repository and are found by atg.commerce.gifts.SearchFormHandler.

For performance reasons, it is suggested that the search always be limited to a single customer profile. If the query scope is too large, ORCE queries will return an error code if the number of records matching the query criteria is too large.

The application may also decide to use atg.commerce.gifts.SearchFormHandler and search only the Commerce repository. This search does not "pre fetch" gift lists from ORCE. The application may choose to implement a custom mechanism to import bulk or real time gift list data from ORCE to improve performance while maintaining accurate (non-stale) results.

Note that ORCE query results do not return fully populated gift lists. The results do not return addresses or gift list items. So, be certain to fully populate a gift list repository item in the search results before allowing the user to view or edit the details of the gift list. This is done by calling RetailGiftlistManager.getGiftlist().

Retrieving ORCE Private Registries

If a gift list (registry) is defined only in ORCE and has not been published, then it will not be retrieved using the search operation.

A private/unpublished gift list can only be retrieved from ORCE using its external ID. It cannot be retrieved by RXM.

If the gift list has been created in Commerce/RXM, then the external ID is known and the private gift list can be synchronized.

RetailGiftlistManager.syncGiftlists()sync private gift lists that are known to RXM.

Security Considerations

Gift List Service Decorator Logging

The Gift List service decorator pipeline logs messages for debugging purposes. These messages can contain customer information such as addresses, e-mail, and phone numbers.

It is recommended that production deployments disable all logging in the Gift List pipeline to avoid exposing any sensitive customer information.

RXM Purchase History Extensions

RXM allows for customers to see their purchase history. This service either obtains the purchase history (completed orders and transactions) from the external system of record or in case the system of record is not available, it can obtain orders from local commerce repository as well. The default external system used is ORCE, which provides only completed order history and transaction history.

Pipeline Extensions

NA

Available API

Purchase history API exists in /oracle/retail/commerce/order/RetailOrderManager that extends commerce SimpleOrderManager. /oracle/retail/commerce/history/droplet/PurchaseSummaryLookup droplet calls getPurchaseHistory(PurchaseHistorySearchCriteria historySearchCriteria) method in RetailOrderManager, passing PurchaseHistorySearchCriteria.

PurchaseHistorySearchCriteria is criteria sent to external system/local repository to query for purchase history. It has the following fields:

Field Name	**Type**	**Purpose**	**Supported by ORCE**
externalCustomerId	string	Customer ID in system of record	Yes
startDate	date	Start date of purchase history period	Yes
endDate	date	End date of purchase history period	Yes
lineItemPopulation	int	Number of itmes to be returned per order or transaction	No
maxRecords	int	Maximum number of records to be fetched per page (if pagination is supported)	No
pageNumber	int	Page number of purchase history result	No
ascending	boolean	Order in which results can be displayed	No

RetailOrderManager has two configurable properties.

Name	Purpose
enableOrdersLocalLookup	If this is set to true, purchase history (only orders) will be retrieved from commerce local repository as well. Enable this only if external system is not available because data in commerce could be stale data. Default: false
defaultMonthSearchRange	In case no end date is mentioned in search criteria, default end date would be three months from start date. If start date is not mentioned in search criteria, then default start date is current date. Default: 3

If enableOrdersLocalLookup is true, getPurchaseHistory method will pass all criteria values to getOrdersForProfile method in /oracle/retail/commerce/order/RetailOrderQueries. In that method, commerce is being queried for incomplete orders for customer using RQL.

Purchase History Transformers

/oracle/retail/commerce/history/PurchaseHistoryServiceConsumer is an interface for consumers of purchase history to call the external service. Implementation class of this interface calls camel route, which calls the SOAP service on RSB to obtain purchase history.

/oracle/retail/commerce/integration/history/PurchaseHistoryServiceConsumerImpl uses camel context /config/retail/commerce/integration/history/purchasehistory-context.xml for purchase history. The context routes are configured to make SOAP calls to the RSB. Before and after each SOAP call, the payload of the route must be transformed. Primarily this is performed by the component at /retail/commerce/integration/history/PurchaseHistoryServiceTransformer. Methods in this component transform PurchaseHistorySearchCriteria to com.oracle.retail.integration.base.bo.purchhistcrivo.v1.PurchHistCriVo while going to RSB and transform response (com.oracle.retail.integration.base.bo.purchhistcoldesc.v1.PurchHistColDesc) from RSB to oracle.retail.commerce.history.PurchaseSummary.

purchasehistory-context does support multiple routes as well.

purchase history route definition

```
<multicast parallelProcessing="true" strategyRef="aggregatorStrategy">
    <!-- In case of multiple end points, add another URI here and add cxfEndpoint
accordingly(also update security.properties) -->
    <to uri="cxf:bean:PurchaseHistEndpoint?loggingFeatureEnabled=false"/>
</multicast>
```

The code above refers to aggregatorStrategy, which is defined in context xml like this

```
<bean id="aggregatorStrategy"
class="oracle.retail.commerce.integration.history.PurchaseHistoryAggregationStrate
gy"/>
```

PurchaseHistoryAggregationStrategy implements Apache Camel's AggregationStrategy class and it merge the results from two end points. So in purchase history route definition, if another end point is added, PurchaseHistoryAggregationStrategy will return the merge result from two end points.

Purchase history transformer is easily extendable. Objects such as PurchaseHistorySearchCriteria, PurchaseSummary and PurchaseItemSummary are created using /retail/commerce/history/PurchaseSummaryFactory.

ORCE Support

RXM is expected to communicate to ORCE through the RSB. The RSB application provides application-specific PAKs that contain additional transformation through XSLT transformations. Development or changes to these transformations require Oracle Service Bus (OSB) with JDeveloper. See *Oracle Retail Service Backbone Development Guide* for how to develop with these paks.

Here is the transformation mapping between RXM and ORCE for purchase history:

Request:

CustPurchHistCriVo maps to getTransactionHistory in CustomerServices(Ver 3.0)

Commerce (CustPurchHistCriVo)	ORCE(getTransactionHistory)
customer_id	customerId
start_date	startDate
end_date	endDate
line_item_population	
max_record	
page_number	

Response:

Commerce(CustPurchHistDesc)	ORCE(Transaction / Order) [TransactionReturnType]
initiate_loc_type	When RetailTransaction/LineItem/CustomerOrderForDelivery !=' ' or RetailTransaction/LineItem/CustomerOrderForPickup != ' ' Then initiate_loc_type = "W" Else initiate_loc_type = "S"
initiate_loc_id	Transactions/Transaction/RetailStoreID

Commerce(CustPurchHistDesc)	ORCE(Transaction / Order) [TransactionReturnType]
purchase_type	When RetailTransaction/LineItem/CustomerOrderForDelivery !=' ' or RetailTransaction/LineItem/CustomerOrderForPickup != ' ' Then purchase_type = "O" Else purchase_type = "T"
id	Transaction/SequenceNumber
status	Transaction/RetailTransaction/@TransactionStatu
currency_code	Transaction/CurrencyCode
grand_total	Transaction/RetailTransaction/Total [@TotalType='TransactionGrandAmount']
submitted_date	Transaction/EndDateTime
number_of_items	count(RetailTransaction/LineItem[not(Tender) and not(Tax)])

Line Item:

Commerce	ORCE (Transaction)
sequence_no	Transaction/RetailTransaction/LineItem/SequenceNumber
item_id	Transaction/RetailTransaction/LineItem/Sale/ItemID
item_description	
quantity	Transaction/RetailTransaction/LineItem/Sale/Quantity
currency_code	Transaction/CurrencyCode
extended_sell_price	Transaction/RetailTransaction/LineItem/Sale/ExtendedAmount

RXM Limitations:

In this release of RXM, all Xstore items have to exist in RXM in order to retrieve the purchase history for transactions placed in Xstore.

Security Considerations

Purchase History functions have no new access roles defined.

Accessing remote purchase history service operations must be secured by default. Security properties are loaded by the purchase history camel context. See “Securing Commerce Extensions” section for additional information on these settings.

Customization

The RXM module takes advantage of both Nucleus and Spring for easy customization and dependency injection. Class names can be found in the purchasehistory-context.xml as well as the various component properties files. See the *Oracle Commerce Platform Development Guide* for more information on how to customize Nucleus components.

RXM Order Extensions

RXM extends Oracle Commerce's order functionalities by providing additional APIs for picking up orders in the store, shipping to the store, shipping to the customer, canceling individual items, and retrieving order's status from an external Order Management System (OMS).

Pipeline Extensions

commercepipeline is extended to support submitting, saving, and retrieving of orders from an external OMS.

- **processOrder** pipeline is extended to add or overlay the following pipelinelinks and processors:
  - **setLineSequenceNumbers**

    This processor is added after sendGiftPurchasedMessage pipelinelink. It sets the line sequence numbers for all RetailShippingGroupCommerceItemRelationship instances.

Transactional Mode	TX_MANDATORY
Component	/retail/commerce/order/processor/SetLineSequenceNumbers
Object	oracle.retail.commerce.order.processor ProcSetLineSequenceNumbers
Transitions	Return value of 1 executes addOrderToRepository next

  - **sendFulfillmentMessage**
  - The object for this processor is replaced with oracle.retail.commerce.order.processor.ProcExportOrderMessage. This processor submits an order to external system by calling the submitOrder API in the RetailOrderManager.

SendFulfillmentMessage.properties

```
$class=oracle.retail.commerce.order.processor.ProcExportOrderMessage
retailOrderManager = /atg/commerce/order/OrderManager
```

- validateForCheckout pipeline is extended to add the following processors
  - validateOrderForStoreReserve
  - This processor is added after checkForDiscontinuedProducts pipelinelink. This processor verifies that the state of the reserveInStore and shipToStore flags is consistent.

Transactional Mode	TX_MANDATORY
Component	/retail/commerce/order/processor/ValidateOrderForStoreReserve
Object	oracle.retail.commerce.order.processor.ProcValidateOrderForStoreReserve
Transitions	Return value of 1 executes checkOrderInventory next

  - checkOrderInventory
  - This processor is added after validateOrderForStoreReserve pipelinelink. This processor verifies that there is available inventory for all commerce items in an Order.

Transactional Mode	TX_MANDATORY

Component	/retail/commerce/order/processor/CheckOrderInventory
Object	oracle.retail.commerce.order.processor.ProcCheckOrderInventory
Transitions	None, this is the last link in the chain and will cause the PipelineManager to return to the caller.

- updateOrder pipeline is extended to add or replace the following pipeline links and processors:
  - updateRelationshipObjects
  - This processor is replaced with ProcRetailSaveRelationshipObjects. This processor extends ProcSaveRelationshipObjects in order to save the associated line references from an external Order Management System.

Transactional Mode	TX_MANDATORY
Component	/retail/commerce/order/processor/SaveRelationshipObjects
Object	oracle.retail.commerce.order.processor.ProcRetailSaveRelationshipObjects
Transitions	Return value of 1 executes updatePriceInfoObjects

- updatePriceInfoObjects
- This processor is replaced with ProcSavePriceInfo. This processor extends ProcSavePriceInfoObjects in order to save extended order line item attributes to the repository.

Transactional Mode	TX_MANDATORY
Component	/retail/commerce/order/processor/SavePriceInfoObjects
Object	oracle.retail.commerce.order.processor.ProcSavePriceInfo
Transitions	Return value of 1 executes updateCostCenterObjects

Repository Extensions

orderrepository is extended to add or replace the following properties and item descriptors.

- originOfOrder property is replaced to add option value "store" to retrieve store originated orders.
- An auxiliary table "rxmpp_order" is added to "order" item descriptor. The property externalOrderId in the table is added to save the orderId generated by OMS.

orderrepository.xml

```
    <item-descriptor name="order" xml-combine="append">
        <table name="dcspp_order" xml-combine="append">
            <property name="originOfOrder" column-name="origin_of_order" data-
type="enumerated" default="default" expert="true"
              category-resource="categorySchedule" display-name-
resource="originOfOrder" xml-combine="replace">
               <attribute name="resourceBundle"
value="atg.commerce.OrderRepositoryTemplateResources"/>
               <attribute name="useCodeForValue" value="false"/>
                <option value="default" code="0"/>
                <option value="scheduledOrder" code="1"/>
                <option value="contactCenter" code="2"/>
                <option value="store" code="3"/>
                <attribute name="propertySortPriority" value="30"/>
```

```
          </property>
        </table>
        <table name="rxmpp_order" type="auxiliary" id-column-name="order_id">
            <property name="externalOrderId" column-name="ext_order_id" data-
type="string" expert="true" category-resource="categoryBasics" display-name-
resource="externalOrderId"/>
        </table>
    </item-descriptor>
```

- An item descriptor "inStorePickupShippingGroup" is added for the purpose of differentiating between pickup, ship to store, and reserve in store orders.
- "detailedItemPriceInfo" item descriptor is extended to include depositPaidAmount and depositPaidPercent properties in support of reserve in store orders.
- Auxiliary tables "rxmpp_shipitem_rel" and "rxm_ship_rel_ci_line_ref" are added to "shipItemRel" item descriptor to save additional order line item attributes.
- "commerceItemLineRef" item descriptor with properties lineRefLowBound and lineRefHighBound is added to capture line references from the OMS.

Class Extensions

The following extensions have been made to support submission, retrieval of store and web orders, and cancelling order items from the OMS.

- **RetailOrder, RetailOrderImpl**: This class extends atg.commerce.order.Order to add additional attributes to the order such as externalOrderId, and estimatedLoyaltyPoints .
- **RetailOrderQueries**: This class extends atg.commerce.order.OrderQueries and contains methods for executing various types of order queries, for example getOrderIdFromRepository(String) which queries the order repository to obtain the orderId associated with the external order ID passed as an input.
- **RetailOrderTools:** This class extends atg.commerce.order.OrderTools and provides order behavior unique to the RXM environment.
- **RetailShippingGroupCommerceItemRelationship:** This class extends the atg.commerce.order.ShippingGroupCommerceItemRelationship to capture of information mainly the line item number information retrieved from the OMS.
- **RetailCommerceItem, RetailCommerceItemImpl:** This class extends CommerceItem to add additional item attributes such as ItemDescription and CurrencyCode.
- **RetailCommerceItemManager:** This class extends CommerceItemManager to provide access to the protected methods.
- **RetailInStorePickupShippingGroup:** This class extends InStorePickupShippingGroup to add additional flags to indicate whether it is ship to store and reserve in store.
- **AddRetailCommerceItemInfo:** This class extends AddCommerceItemInfo and and adds additional properties that determines if an item needs to be shipped to store.
- **PickupAwarePurchaseProcessHelper:** This class extends PurchaseProcessHelper.
- **RetailDetailedItemPriceInfo:** This class extends DetailedItemPriceInfo in order to include additional order line item attributes.

Available API

OrderManager component is overlayed by oracle.retail.commerce.order.RetailOrderManager object.

OrderManager.properties

```
$class=oracle.retail.commerce.order.RetailOrderManager

enableOrdersLocalLookup=false
defaultMonthSearchRange=3

profileTools=/atg/userprofiling/ProfileTools

customerOrderServiceConsumer=/retail/commerce/integration/order/CustomerOrderServi
ceConsumer
purchaseHistoryServiceConsumer=/retail/commerce/integration/history/PurchaseHistor
yServiceConsumer
```

RetailOrderManager extends SimpleOrderManager and provides the following APIs for submitting, querying, and manipulating orders.

- retrieveOrder(String)
- cancelCommerceItems(RetailOrder, Map<String, Double>)
- submitOrder(RetailOrder)

Order Transformers

After the purchase process is successfully completed and the order is placed, the order must be submitted to the Order Management System to handle the fulfillment process.

CustomerOrderServiceConsumer component contains the operations to submit, retrieve, and cancel order items to the external system. These operations invoke the routes defined in /retail/commerce/integration/order/order-context.xml that calls the SOAP operations hosted at the Retail Service Backbone (RSB) with the required payload. To obtain the desired input/output payloads, transformer components are used to transform to and from RXM objects and Retail Business Objects (RBO).

createCustomerOrder - This route defined in the order-context xml, takes oracle.retail.commerce.order.RetailOrder transformed into a com.oracle.retail.integration.base.bo.custorderdesc.v1.CustOrderDesc as an input payload, invokes the "createCustomerOrder" service operation on the RSB and returns the CustOrderDesc payload which is transformed into RetailOrder object.

- OrderTransformer.toCustOrderDesc(RetailOrder)
- OrderTransformer.toRetailOrder(CustOrderDesc)

After the successful submission to the external system, OrderManager.replaceOrderContects(sourceOrder, targetOrder) is invoked to replace the contents of the submitted order with order returned as response by the external system. The properties to be ignored during this copy can be found in OrderTools.properties.

OrderTools.properties

```
#
# Properties ignored during bean copy for various beans
#
ignoredPricingAdjustmentProperties=adjustment
ignoredItemPriceInfoProperties=adjustments,currentPriceDetails
ignoredRetailDetailedItemPriceInfoProperties=
ignoredOrderPriceInfoProperties=taxableShippingItemsSubtotalPriceInfos,\
      nonTaxableShippingItemsSubtotalPriceInfos,shippingItemsSubtotalPriceInfos
ignoredTaxPriceInfoProperties=shippingItemsTaxPriceInfos
ignoredPaymentGroupProperties=id
ignoredPaymentGroupShippingGroupRelationshipProperties=
ignoredPaymentGroupCommerceItemRelationshipProperties=
ignoredOrderProperties=id,approvalSystemMessages,approverIds,approverMessages,\
      authorizedApproverIds,awards,estimatedLoyaltyPoints,externalReturnRequests,\
      manualAdjustments,priceInfo,quoteInfo,taxPriceInfo
```

queryCustomerOrder - This route is defined in the order-context xml, takes String orderId transformed into a com.oracle.retail.integration.base.bo.custordercrivo.v1.CustOrderCriVo as an input payload, invokes the "queryCustomerOrder" service operation on the RSB, and returns the CustOrderColDesc payload that is transformed into RetailOrder object.

- OrderTransformer.toCustOrderCriVo(String)
- OrderTransformer.toRetailOrder(CustOrderDesc)

order-context.xml

```
<route id="queryCustomerOrder">
    <from uri="direct:queryCustomerOrder"/>
    <bean ref="OrderTransformer" method="toCustOrderCriVo"/>
    <to uri="cxf:bean:CustomerOrderServiceEndpoint"/>
    <bean ref="OrderTransformer" method="toRetailOrder"/>
</route>
```

After the successful submission to the external system, OrderManager.replaceOrderContects(sourceOrder, targetOrder) is invoked to replace the order contents of submitted order with the order returned in response by the external system.

cancelCustomerOrderItems - This route, defined in the order-context xml, takes RetailOrder transformed into a com.oracle.retail.integration.base.bo.custorderpicvo.v1.CustOrderPicVo as an input payload, invokes the "pickupCustomerOrderItems" service operation on the RSB.

- OrderTransformer.toCustOrderPicVo(RetailOrder)

Components involved in transformations are

- OrderTransformer - Loads the required Managers and Tools to perform the transformations. orderStatusMap could be extended to map any additional statuses required by the OMS.

OrderTransformer#orderStatusMap

```
# CustomerOrderDesc order status to Oracle Commerce Order States map
orderStatusMap=\
NEW=2,\
PARTIAL=2,\
FILLED=2,\
CANCELED=4,\
COMPLETED=5,\
OPEN=1,\
HELD=7,\
ERROR=6,\
SUSPENDED=8
```

- **OrderLineItemTransformer** -This class contains the methods used to transform order line item information between the Retail Service Bus (RSB) and the Oracle Commerce formats.
- **PaymentTransformer** - This class provides the methods to convert PaymentGroup objects into com.oracle.retail.integration.base.bo.paymentdesc.v1.PaymentDesc. In this component's properties file, the cardTypeMap can be extended to map additional card types.
- **ProfileServiceTransformer** - To transform to and from com.oracle.retail.integration.base.bo.customerdesc.v1.CustomerDesc and RXM profile atg.repository.RepositoryItem.

OROMS Support

RXM integrates with the following OROMS 16.0 services:

- CWOrderIn
- CWMessageIn
- CWServiceIn

The messages (that is, XSDs) that the above services use are:

- CWCancel: 1.0
- CWCUSTHISTIN: 2.0
- CWORDERIN: 8.0
- CWOrderOut: 9.0

RXM is expected to communicate to OROMS through the RSB. The RSB application provides application-specific PAKs that contain additional transformations through XSLT transformations. Development or changes to these transformations require Oracle Service Bus (OSB) with JDeveloper. See *Oracle Retail Service Backbone Development Guide* on how to develop using these PAKs.

Some of the constants used in the XSLTs are "externalized" using JDeveloper variables. These variables values depend on how OROMS is set up.

CreateOrderToCWOrderIn.xsl

```
<xsl:variable name="MSG_DESTINATION">RDC</xsl:variable>
<xsl:variable name="MSG_TYPE">CWORDERIN</xsl:variable>
<xsl:variable name="COMPANY_CODE">17</xsl:variable>
<xsl:variable name="ORCE_INTEGRATED">N</xsl:variable>
<xsl:variable name="OMS_VISA_PAY_TYPE">4</xsl:variable>
<xsl:variable name="OMS_MC_PAY_TYPE">5</xsl:variable>
<xsl:variable name="OMS_CASH_CHECK_PAY_TYPE">1</xsl:variable>
<xsl:variable name="OMS_AMEX_PAY_TYPE">3</xsl:variable>
<xsl:variable name="OMS_DISCOVER_PAY_TYPE">6</xsl:variable>
```

OROMS Limitations:

- Fractional item quantities are not supported.
- Mixed orders are not supported.
- RXM handles the price, promotions, and tax calculations; therefore, this information is not sent to OROMS.
- Payment types supported are credit cards with tokenization, gift cards, and checks.

RXM Limitations:

In this release of RXM, cancel order operation is not supported. RXM supports only cancel of the individual order items. An order with all the order items canceled is canceled automatically by OMS.

In this release of RXM, split orders (with items to be picked up and shipped to a store) are not supported due to a limitation of OROMS.

This version of RXM warehouse inventory lookup was only tested with a simulator and not with RMS.

Security Considerations

There are some specific security concerns in the order handling with RXM.

Payment

Out of the box, RXM only sends credit card tokens to OMS. This means retailers are required to use a tokenization service. For more details on tokenization support in RXM, see the *Oracle Commerce Retail Extension Module Security Guide*.

Order ID Validation

RXM provides an ESAPI validator to format the Order ID generated by OMS. This is available through the oracle.retail.commerce.security.EsapiSecurityTools validate method. Use the constant oracle.retail.commerce.security.BaseSecurityTools.EXTERNAL_ORDER_ID_SECURITY_CONTEXT to access the correct validator.

To modify this validator, update the validationRules.properties file. This file is packaged with the RXM.Base module in /lib/oracle.retail.commerce.base-16.0.0.jar under /retail/commerce/security.

Customization

The Order Extensions provide several additional points of customization.

OMS Endpoint Configuration

The Order extensions are designed to submit the Commerce order to an external OMS through a web services call. The endpoint for that web service is configured in retail/commerce/integration/order/security.properties. This file is packaged as part of the oracle.retail.commerce.base-16.0.0.jar in the RXM.Base module's lib directory. No value is provided by default.

Order endpoint security.properties file

```
### Web Service options
### endpointUri can be changed to HTTP or HTTPS
endpointUri=
```

Nucleus Component Overrides

The RetailOrderManager and OrderTransformer follow the Commerce model for initializing components through Nucleus. As such, both components are available for extension and replacement through common Commerce development patterns. Additionally, each component wires in additional components that can be customized as well. This section provides a description of the configuration options.

RetailOrderManager

The RetailOrderManager has two additional configuration properties in its component definition:

- enableOrdersLocalLookup: The setting to control if the RetailOrderManager looks in the Commerce repository for customer Order history. If set to true, the RetailOrderManager will query the Commerce repository. If set to false, it will not. The default value is false.
- defaultMonthSearchRange: The setting to control how much purchase history the RetailOrderManager will retrieve from the system of record. The default is three months.

OrderTransformer

The OrderTransformer uses several delegate transformers to deal with the various parts of an order. These delegates are all configured through its Nucleus component properties file, /retail/commerce/integration/order/OrderTransformer.properties. Any or all of these delegates can be customized and replaced using the traditional Commerce development patterns.

Web Service Integration

All interactions with the OMS web service are done through the Apache Camel framework. As mentioned above, the Order extensions configuration is housed in the order-context.xml packaged as part of the RXM.Base module's config/config.jar under /retail/commerce/integration/order. This file configures the CXF endpoint, sets up the OrderTransformer, and defines the routes available to the Order extensions. By default, the follow routes are available:

- **createCustomerOrder** - Submits a Commerce order to the OMS endpoint for processing.
- **queryCustomerOrder** - Retrieves an order from the OMS and transforms it into a Commerce order.
- **cancelCustomerOrderItems** - Submits a request to the OMS to cancel a particular item on an existing order.
- **requestNewOrderId** - Submits a request to the OMS for a new order ID.
- **cancelNewOrderId** - Cancels a previous request for an order ID.

Each of these routes can be customized using methods described in the Apache Camel documentation. By default, these routes use the OrderTransformer to transform the Commerce order into the RSB order representation, call the OMS, and then transform the result back to a Commerce object.

To better understand how to secure the communication with the OMS, see the “Securing Commerce Extension”s section of this guide.

RXM Order Broker Extensions

The RXM Order Broker extensions have been designed to minimize the impact to existing users of Commerce's AbstractInventoryManagerImpl.

The Order Broker (OROB) supports two operations.

- queryAvailableToPromise()
- queryInventoryInformation()

All methods defined in AbstractInventoryManagerImpl ultimately call one of these two operations.

Versions

RXM order broker extensions have been tested with version 16.0 of the OROB APIs.

Pipeline Extensions

ProcCheckOrderInventory is called as part of the commerce pipeline. This processor calls the BrokeredStoreInventoryManager class, which in turn calls the OROB service to check if inventory is available to promise.

/atg/commerce/commercepipeline.xml

```
        <pipelinelink name="checkOrderInventory" transaction="TX_MANDATORY">
            <processor
jndi="/retail/commerce/order/processor/CheckOrderInventory"/>
        </pipelinelink>
```

Data Extensions

No data extensions are required for the Order Broker extensions.

Class Extensions

The following RXM classes extend Commerce to provide an interface to OROB.

- StoreInventoryManager extends atg.commerce.inventoryAbstractInventoryManagerImpl
- BrokeredStoreInventoryManager extends StoreInventoryManager

The application can extend the above classes if additional properties are required or if different method behavior is desired.

StoreInventoryManager extends atg.commerce.inventoryAbstractInventoryManagerImpl

- This class implements or extends the base Commerce class in a generic way that can potentially work with external systems other than OROB.

BrokeredStoreInventoryManager extends StoreInventoryManager

- This class overrides and customizes the following two methods to work specifically with OROB.
- queryAvailableToPromise()
- queryInventoryInformation()

Repository Listeners

No repository listeners are required for this RXM extension.

BrokeredStoreInventoryServiceConsumer

The BrokeredStoreInventoryServiceConsumer is the RXM interface to the external system of record, OROB. BrokeredStoreInventoryManager uses this interface to communicate with ORCE (through RSB). This interface uses Camel as a router to the RSB web services. The camel context file, inventory-context.xml, maps the interfaces in BrokeredStoreInventoryServiceConsumer to the specific transformers and RSB services for each inventory operation.

Store Inventory Transformers

Here is the list of transformers used to transform inventory requests and results to or from the standard RBO payloads.

Class: InventoryServiceTransformer

- toInvAvailCriVo()
- toInvAvToPromColCriVo()
- toSearchAreaDesc()
- toDistanceUnit()
- toInventoryInfos()

The class can be extended and these methods can be overridden if needed by the application.

RSB XSLT Transformers

Here is a list of RSB XSLT transformers used to transform the RBO payloads into the format required by OROB.

- LocateProductAvailResponseToStoreProductAvailResponse.xsl
- StoreProductAvailRequestToLocateProductAvailRequest.xsl

Differences Between the Commerce AbstractInventoryManagerImpl and OROB

OROB does not support unit of measure quantities. Therefore, OROB does not support floating point quantities, only "each" quantities (integer).

OROB returns inventory counts as sets of items per location. RSB returns inventory as a collection of item/location pairs.

Security Considerations

OROB requires basic authentication credentials in the request header.

4

Securing Commerce Extensions

This chapter provides a security overview for RXM. For additional information, see the *Oracle Commerce Retail Extension Module Security Guide*.

RXM integrates the Commerce Platform with several Oracle Retail assets, effectively making the platform a client to those services. As a client, it is important to secure the communications between the remote services and Oracle Commerce. The primary means of security is through SOAP policy enforcement, specifically Username Token over HTTPS, referred to here as Policy A.

Introduction

Policy A is relatively simple. The communication is over HTTPS but the UsernameToken is in plain text. A timestamp element is included as well. The client and server machine form a trust relationship through a public-private key relationship and the use of encrypted stores called key stores and trust stores. The following example shows a typical deployment of Commerce, with RXM communicating with services through the Oracle Retail Service Backbone (RSB).

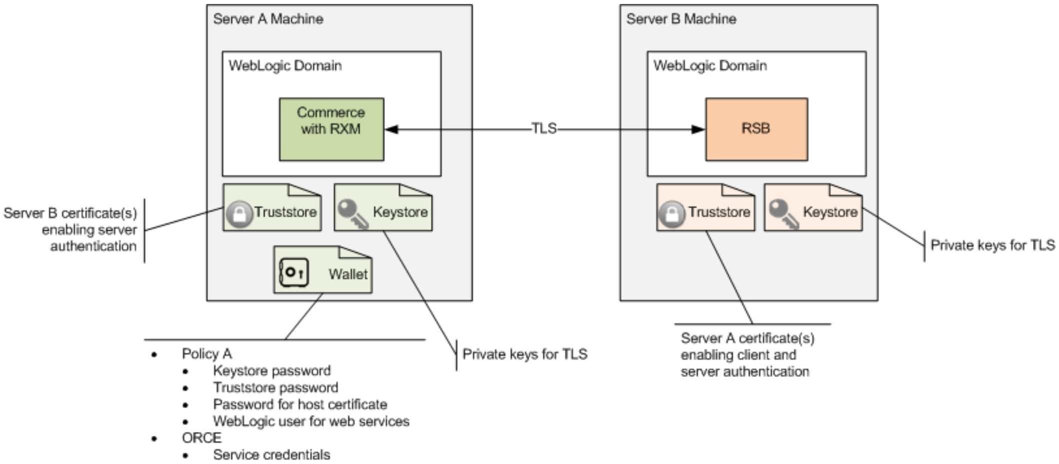

Figure 7: Typical Commerce Deployment

Configuration

The configuration of Policy A for RXM occurs in the server and in the client, as described this section.

Server Side Changes (RSB)

There are three major steps to configuring Policy A for the RSB deployed in WebLogic.

Configure SSL

Policy A requires that HTTPS be used. This step requires a Keystore and Truststore compatible with the Keystore and Truststore used by the client.

Create Policy A User

Policy A requires that each inbound message be authenticated. The SOAP header of each inbound message must contain a user name and password known to WebLogic. To accomplish this, the expected user credential must be added to the WebLogic security realm of the server the RSB is deployed on.

Apply Policy to Web Services

Configure each of the web services used by the clients with a Web Service Security Policy. There are two parts to this configuration:

- Configure a WebLogic WS Security Policy file on each web service. Policy A is defined in a single WebLogic xml file.
- Configure a user policy condition on each web service. This is how the WS Security framework knows which user names and passwords are valid for message authentication.

Client Side Changes (RXM)

Corresponding changes are required on the client side so the correct certificates and WS security values are sent with the request. Most of the security configurations are referenced in the camel context file (that is, loyalty-context.xml) but defined in security.xml. Properties are externalized to security.properties. Both of these files can be found in the root directory of the oracle.retail.commerce.base-16.0.0.jar located in RXM/Base/lib.

The security.xml file establishes the security components used by the various Camel endpoints. It declares five types of components:

- **KeyManagerBuilder:** This component provides access to a keystore. It is needed because RXM retrieves all the required user credential and certificate passwords from a wallet file instead of keeping them as plain text in the XML file.
- **TrustManagerBuilder:** This component provides access to a truststore. It is required because RXM retrieves the truststore password from a wallet file instead of keeping it plain text in the XML file.
- **WSS4JOutInterceptor:** This component configures WS Security. It is provided by the Apache CXF API.
- **WebServicePasswordHandler:** This component configures the password for WS Security. It is required because RXM retrieves the WebLogic user's password from a wallet file instead of keeping it as plain text in the XML.
- **BasicAuthenticationHeaderInterceptor:** This component handles the creation of any HTTP header authentication requirements.

The values used to create these components are all externalized into the security.properties file. It contains properties for Keystore, Truststore, and WS Security. It also contains a soap.mustunderstand property. Setting this to false allows making requests through HTTP, if the web service is not secured with Policy A.

As mentioned above, all passwords used by RXM for communications are stored in a CSF wallet file (cwallet.sso) and retrieved using the Credential Store Framework (CSF) API. For WebLogic 12.1.3 without JRF, the wallet and its associate files must be placed in the root of the domain containing RXM. These are the required files:

- cwallet.sso
- jazn-data.xml
- jps-config.xml

As an example, if RXM's domain is called rxm_domain, the files must be placed in D:\Oracle\Middleware\WLS.12.1.3\user_projects\domains\rxm_domain.

Sample Files

security.xml

```
    <bean id="keyManagerBuilder" 
class="oracle.retail.commerce.integration.security.KeyManagerBuilder" factory-
method="buildKeyManagers">
        <constructor-arg index="0" type="java.lang.String" 
value="${keystore.path}"/>
        <constructor-arg index="1" type="java.lang.String" 
value="${keystore.password.alias}"/>
        <constructor-arg index="2" type="java.lang.String" 
value="${keystore.cert.password.alias}"/>
        <constructor-arg index="3" type="java.lang.String" 
value="${keystore.type}"/>
    </bean>

    <bean id="trustManagerBuilder" 
class="oracle.retail.commerce.integration.security.TrustManagerBuilder" factory-
method="buildTrustManagers">
        <constructor-arg index="0" type="java.lang.String" 
value="${truststore.path}"/>
        <constructor-arg index="1" type="java.lang.String" 
value="${truststore.password.alias}"/>
        <constructor-arg index="2" type="java.lang.String" 
value="${truststore.type}"/>
    </bean>

    <bean id="wss4jOutInterceptor" 
class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
        <property name="properties">
            <map>
                <entry key="action" value="UsernameToken Timestamp"/>
                <entry key="passwordType" value="PasswordText"/>
                <entry key="user" value="${webservice.user}"/>
                <entry key="timeToLive" value="5000"/>
                <entry key="mustUnderstand" value="${soap.mustunderstand}"/>
                <entry key="passwordCallbackRef">
                    <ref bean="passwordCallbackHandler"/>
                </entry>
            </map>
        </property>
    </bean>

    <bean id="passwordCallbackHandler" 
class="oracle.retail.commerce.integration.security.WebServicePasswordHandler"/>

    <bean id="AuthenticationHeaderInterceptor" 
class="oracle.retail.commerce.integration.security.BasicAuthenticationHeaderInterc
eptor">
        <constructor-arg index="0" type="java.lang.String" 
value="${auth.userName}"/>
        <constructor-arg index="1" type="java.lang.String" 
value="${auth.password}"/>
        <constructor-arg index="2" type="java.lang.String" value="${auth.orgId}"/>
        <constructor-arg index="3" type="java.lang.String" 
value="${auth.scheme}"/>
    </bean>
    <bean id="OrderBrokerAuthenticationHeaderInterceptor"
class="oracle.retail.commerce.integration.security.BasicAuthenticationHeaderInterc
eptor">
        <constructor-arg index="0" type="java.lang.String"
value="${orob.auth.userName}"/>
        <constructor-arg index="1" type="java.lang.String"
value="${orob.auth.password}"/>
        <constructor-arg index="2" type="java.lang.String"
value="${orob.auth.orgId}"/>
        <constructor-arg index="3" type="java.lang.String"
value="${orob.auth.scheme}"/>
    </bean>

    <bean id="OMSAuthenticationHeaderInterceptor"
class="oracle.retail.commerce.integration.security.BasicAuthenticationHeaderInterc
eptor">
        <constructor-arg index="0" type="java.lang.String"
value="${oroms.auth.userName}"/>
        <constructor-arg index="1" type="java.lang.String"
value="${oroms.auth.password}"/>
        <constructor-arg index="2" type="java.lang.String"
value="${oroms.auth.orgId}"/>
        <constructor-arg index="3" type="java.lang.String"
value="${oroms.auth.scheme}"/>
    </bean>
```

security.properties – root

```
### The soap.mustunderstand flag should be set to false when using HTTP so SOAP on
the Server side
### does not need to understand Web Service Security related headers needed for
Policy A or B.
soap.mustunderstand=false

### Web Service User
webservice.user=

### Keystore Properties
### Example keystore.path=C://keystores//retail//webservices//keystore.jks
keystore.path=
keystore.password.alias=
keystore.cert.password.alias=
keystore.type=

### Truststore Properties
### Example: truststore.path=C://jdk1.7.0_65//jre//lib//security//cacerts
truststore.path=
truststore.password.alias=
truststore.type=

### HTTP Authentication properties for ORCE Web Services. ORCE does not use the
### default "Basic" authorization token scheme. Values are retrieved from wallet.
auth.userName=
auth.password=
auth.orgId=
### Example: auth.scheme=Basic
auth.scheme=


### HTTP Authentication properties for OROB Web Services.
orob.auth.userName=
orob.auth.password=
orob.auth.orgId=
orob.auth.scheme=

### HTTP Authentication properties for OROMS Web Services.
oroms.auth.userName=
oroms.auth.password=
oroms.auth.orgId=
oroms.auth.scheme=
```

security.properties - end point

```
### Web Service options
### endpointUri can be changed to HTTP or HTTPS
endpointUri=
```

5

RXM Web Services

Design Approach

RXM 16.0 provides a set of web services to enable access to web content to the store. These services are based on the RTG RSB service model and follow a similar implementation pattern. Each implements a ServiceInterface published as part of the RSB process. Behind that interface is a ServiceProviderImpl class that implements any exposed methods. For RXM, this ServiceProviderImpl class is a simple facade that delegates the implementation to pluggable Nucleus components. The ServiceProviderImpl invokes Nucleus to access the implementing component and delegates the call, passing all given parameters along. In most cases, these Nucleus components are global, but being global is not a requirement.

To enable Nucleus on these JAX-WS services, RXM has introduced a SOAP handler chain and associated it with each service. The handler chain is defined in the oraclecommercehandlers.xml file in the same CLASSPATH as the ServiceInterface. By default, RXM declares two handlers: ServletPipelineHandler and MessageLoggingSOAPHandler.

- **ServletPipelineHandler**: This SOAP handler is responsible for establishing the request context with Nucleus. It works very similarly to the Commerce request handling pipeline. The pipeline invoked by this handler always starts with a HeadPipelineServlet that, among other things, sets up the DynamoHttpServletRequest and DynamoHttpServletResponse. Once the incoming request is handled, JAX-WS also gives this handler a chance to handle the message, as the response is outbound. During this time, this handler cleans up any thread-sensitive state setup during the inbound phase.
- **MessageLoggingSOAPHandler**: Prints incoming and outgoing messages to sysout, (that is, System.out). To enable this feature, add this handler to the handler chain of the web service and set the System property or ServletContext init parameter with the key oracle.retail.commerce.service.handler.MessageLogging and value to "true". If the System property is true, it will override any value set into the servlet init params.

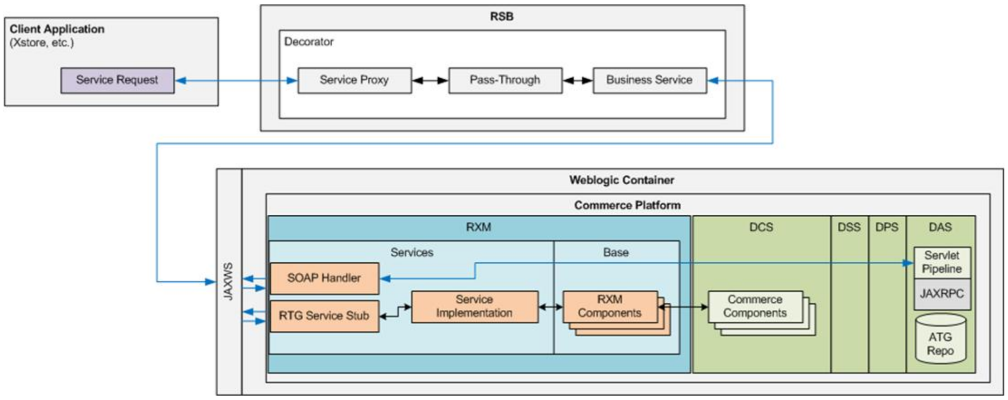

Figure 8: RXM Web Services

For specifics on each service, see the relevant section.

JAX-WS Configuration

The SOAP web services provided by RXM.Services use the JAX-WS API, which is different from the JAX-RPC web service (see Web Services Guide) framework provided by the Oracle Commerce Platform. These services provide RSB-based operations for use by other Oracle Retail (and potentially non-Oracle) applications.

The web services are provided with JAX-WS annotations on the appropriate Java class files.

- @WebService is used to mark classes as web services.
- @HandlerChain is used to configure each service with reference to the /com/oracle/retail/ooc/integration/services/<service>/v1/oraclecommercehandlers.xml path.

The use of annotations means that WebLogic (or other application server) must be told to look for JEE resources. The ooc-service-ejb-16.0.0.jar must be configured as a <module> in the EAR's /META-INF/application.xml for the application server to find the services. The oraclecommercehandlers.xml file configures each service with the JAX-WS SOAPHandler, oracle.retail.commerce.service.handler.ServletPipelineHandler. The ServletPipelineHandler is used by each service to execute the servlet request pipeline.

Services Provided

The following services, described in this chapter, are provided: Targeted Item Service, Shopping Cart Service, and Item Information Service.

Item Information Service

The Item Information Service (also known as the Extended Item Information Service) provides calling applications with additional Commerce information about Item SKUs and their related products. This service has a fairly simple API, taking in Item SKUs and returning a configurable set of Commerce attributes about those SKUs and their related products. The service also takes an optional language indicator telling the service what language to return the information in, if possible.

ItemInformationService API

```
public interface ItemInformationPortType {
    public ItmInfoColDesc retrieveItemInformation(ItmInfoCriVo itmInfoCriVo)
            throws EntityNotFoundWSFaultException, IllegalArgumentWSFaultException,
                    IllegalStateWSFaultException, ValidationWSFaultException;
}
```

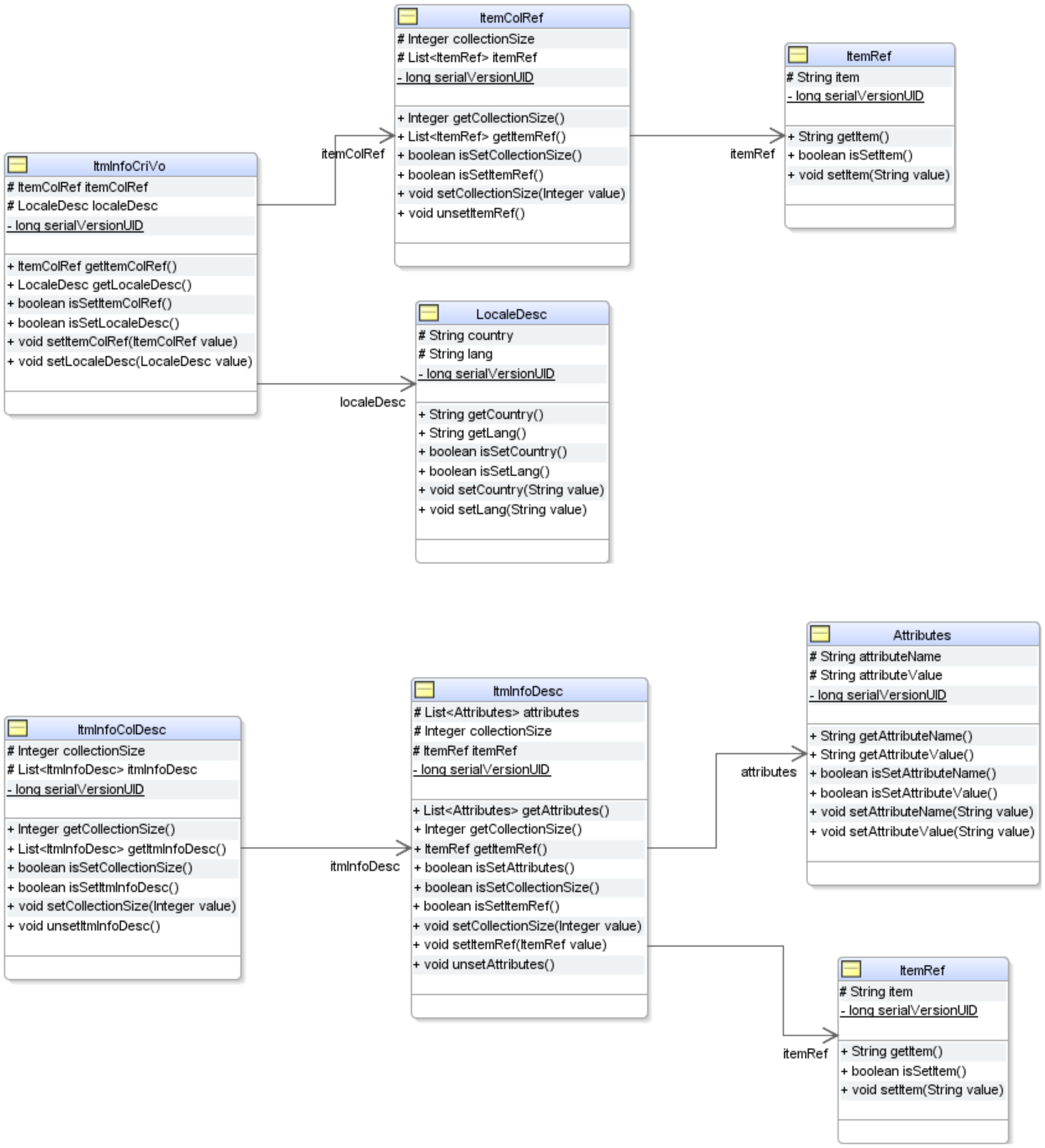
ItemColRef
# Integer collectionSize
# List<ItemRef> itemRef
- long serialVersionUID
+ Integer getCollectionSize()
+ List<ItemRef> getItemRef()
+ boolean isSetCollectionSize()
+ boolean isSetItemRef()
+ void setCollectionSize(Integer value)
+ void unsetItemRef()
ItemRef
# String item
- long serialVersionUID
+ String getItem()
+ boolean isSetItem()
+ void setItem(String value)
itemColRef
itemRef
ItmInfoCriVo
# ItemColRef itemColRef
# LocaleDesc localeDesc
- long serialVersionUID
+ ItemColRef getItemColRef()
+ LocaleDesc getLocaleDesc()
+ boolean isSetItemColRef()
+ boolean isSetLocaleDesc()
+ void setItemColRef(ItemColRef value)
+ void setLocaleDesc(LocaleDesc value)
LocaleDesc
# String country
# String lang
- long serialVersionUID
+ String getCountry()
+ String getLang()
+ boolean isSetCountry()
+ boolean isSetLang()
+ void setCountry(String value)
+ void setLang(String value)
localeDesc
Attributes
# String attributeName
# String attributeValue
- long serialVersionUID
+ String getAttributeName()
+ String getAttributeValue()
+ boolean isSetAttributeName()
+ boolean isSetAttributeValue()
+ void setAttributeName(String value)
+ void setAttributeValue(String value)
ItmInfoDesc
# List<Attributes> attributes
# Integer collectionSize
# ItemRef itemRef
- long serialVersionUID
+ List<Attributes> getAttributes()
+ Integer getCollectionSize()
+ ItemRef getItemRef()
+ boolean isSetAttributes()
+ boolean isSetCollectionSize()
+ boolean isSetItemRef()
+ void setCollectionSize(Integer value)
+ void setItemRef(ItemRef value)
+ void unsetAttributes()
attributes
ItmInfoColDesc
# Integer collectionSize
# List<ItmInfoDesc> itmInfoDesc
- long serialVersionUID
+ Integer getCollectionSize()
+ List<ItmInfoDesc> getItmInfoDesc()
+ boolean isSetCollectionSize()
+ boolean isSetItmInfoDesc()
+ void setCollectionSize(Integer value)
+ void unsetItmInfoDesc()
itmInfoDesc
ItemRef
# String item
- long serialVersionUID
+ String getItem()
+ boolean isSetItem()
+ void setItem(String value)
itemRef

Figure 9: Item Information Service

Configuration

The Item Information service is backed by the Nucleus component retail/commerce/service/catalog/ItemInformationService declared by the RXM.Services module. This component has two main configuration points: extendedProductAttributeNames and extendedSkuAttributeNames. Both configuration points take their inputs in the form of <attribute name>=<attribute handler component identifier>. This allows a variety of attribute types to be handled by a single service call. RXM provides the following set of property handler components:

Component	Purpose
`retail/commerce/service/catalog/handlers/StringPropertyTypeHandler`	Returns String properties with proper XML encoding

`retail/commerce/service/catalog/handlers/TimestampPropertyTypeHandler`	Returns Timestamp properties using the JDBC timestamp escape format
`retail/commerce/service/catalog/handlers/AuxiliaryMediaPropertyTypeHandler`	Returns a given RepositoryItem as an XML block, wrapped in a CDATA element. Use this handler to send the entire contents of a related repository item back to the caller.

`retail/commerce/service/catalog/handlers/ExternalMediaPropertyTypeHandler`	Returns the URL property values of the associated Auxiliary Media on the given repository item. Multiple URLs will be returned as a keyed list. For example, if the auxiliaryMedia has two external media instances associated with it, the result would look like this:
	`<key>1</key>`
	`<url><![CDATA[/docroot/content/image1.jpg]]></url>`
	`<key>2</key>`
	`<url><![CDATA[/docroot/content/image2.jpg]]></url>`

	Returns the URL property value of the externalMediaItem on the given repository item

By default, the following set of attributes is returned by the service:

- SKU
  - displayName
  - description
  - auxiliaryMedia
  - thumbnailImage
  - smallImage
  - largeImage
- Product
  - brand
  - longDescription
  - manufacturer
  - inStoreDetailPageUrl

Extension and Customization

The ItemInformationService is designed to be easily extended. There are several points of extension, each covered below.

Returning Additional Properties

The set of attributes returned is determined by Nucleus component properties. By default, the service accesses two repository items from the Catalog Repository: Product and SKU. To add more attributes to the set returned, update either the extendedProductAttributeNames property or the extendedSkuAttributeNames component property with the additional Repository Item properties to return. These are the Repository Item properties and their handlers provided by default.

```
extendedProductAttributeNames=\

brand=/retail/commerce/service/catalog/handlers/StringPropertyTypeHandler,\

longDescription=/retail/commerce/service/catalog/handlers/StringPropertyTypeHandle
r,\
manufacturer=/retail/commerce/service/catalog/handlers/RepositoryItemPropertyTypeH
andler,\

inStoreDetailPageUrl=/retail/commerce/service/catalog/handlers/StringPropertyTypeH
andler

extendedSkuAttributeNames=\

displayName=/retail/commerce/service/catalog/handlers/StringPropertyTypeHandler,\

description=/retail/commerce/service/catalog/handlers/StringPropertyTypeHandler,\

auxiliaryMedia=/retail/commerce/service/catalog/handlers/AuxiliaryMediaPropertyTyp
eHandler,\

thumbnailImage=/retail/commerce/service/catalog/handlers/ExternalMediaPropertyType
Handler,\

smallImage=/retail/commerce/service/catalog/handlers/ExternalMediaPropertyTypeHand
ler,\

largeImage=/retail/commerce/service/catalog/handlers/ExternalMediaPropertyTypeHand
ler
```

Handling Different Property Types

If the added property has a new property type, create a new Property Type Handler to return its String representation. To do this, create a new Java class that implements the oracle.retail.commerce.service.catalog.handlers.PropertyTypeHandler interface.

```
public interface PropertyTypeHandler {
    String getPropertyValueAsString(RepositoryItem repoItem, String propertyName,
Locale locale);
}
```

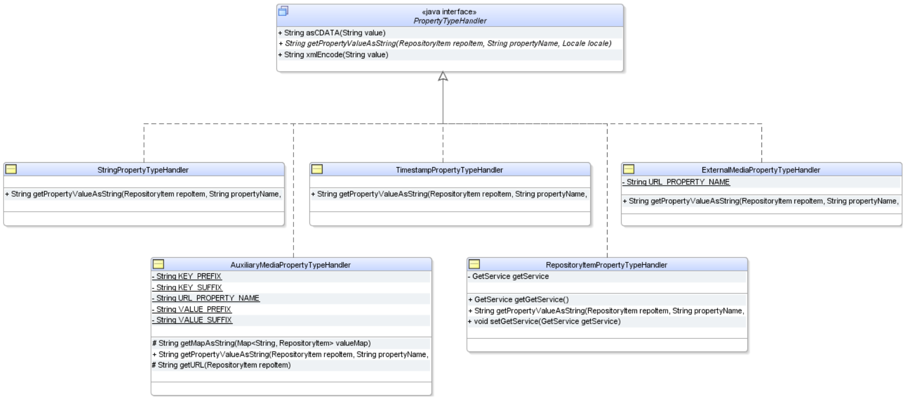
«java interface»
PropertyTypeHandler
+ String asCDATA(String value)
+ String getPropertyValueAsString(RepositoryItem repoItem, String propertyName, Locale locale)
+ String xmlEncode(String value)
StringPropertyTypeHandler
+ String getPropertyValueAsString(RepositoryItem repoItem, String propertyName,
TimestampPropertyTypeHandler
+ String getPropertyValueAsString(RepositoryItem repoItem, String propertyName,
ExternalMediaPropertyTypeHandler
- String URL_PROPERTY_NAME
+ String getPropertyValueAsString(RepositoryItem repoItem, String propertyName,
AuxiliaryMediaPropertyTypeHandler
- String KEY_PREFIX
- String KEY_SUFFIX
- String URL_PROPERTY_NAME
- String VALUE_PREFIX
- String VALUE_SUFFIX
# String getMapAsString(Map<String, RepositoryItem> valueMap)
+ String getPropertyValueAsString(RepositoryItem repoItem, String propertyName,
# String getURL(RepositoryItem repoItem)
RepositoryItemPropertyTypeHandler
- GetService getService
+ GetService getGetService()
+ String getPropertyValueAsString(RepositoryItem repoItem, String propertyName,
+ void setGetService(GetService getService)

Figure 10: Property Type Handler

As a practice, all String representations are wrapped in a CDATA element. The PropertyTypeHander interface provides default methods to assist in that process. All properties returned by the service are sent as a collection of name/value pairs, so no additional changes to the ItmInfoColDesc Retail Business Object are required.

Additional Extension Points

The Item Information Service follows the same implementation pattern that the other RXM services follow, so the extension is similar. The Nucleus component supporting the services can be updated to reference a customized implementation. The provided implementation, oracle.retail.commerce.service.catalog.ItemInformationServiceImpl, provides Java hook points for two of its processes: input validation and Locale processing.

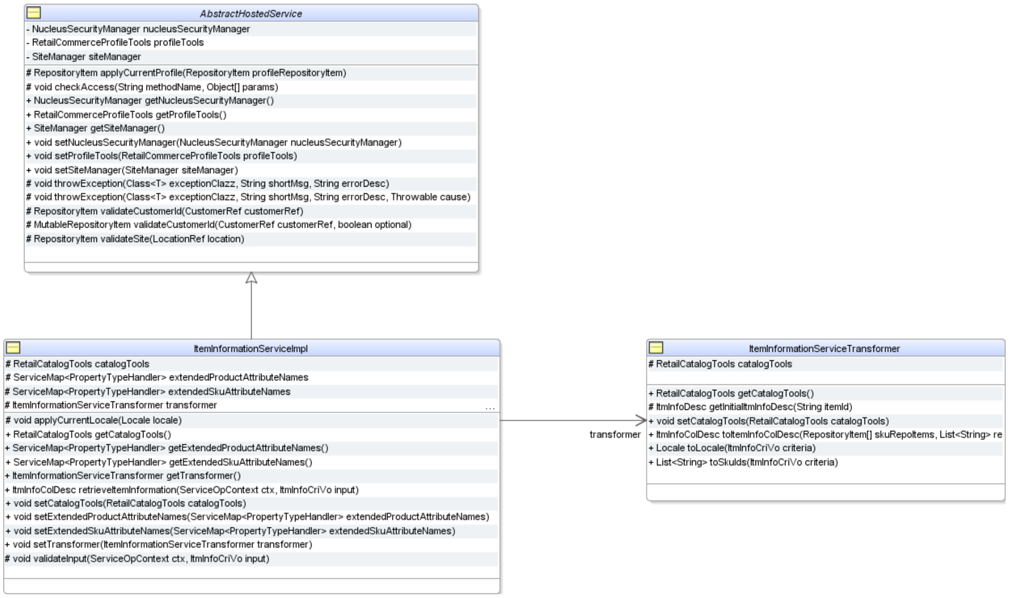
AbstractHostedService
- NucleusSecurityManager nucleusSecurityManager
- RetailCommerceProfileTools profileTools
- SiteManager siteManager
# RepositoryItem applyCurrentProfile(RepositoryItem profileRepositoryItem)
# void checkAccess(String methodName, Object[] params)
+ NucleusSecurityManager getNucleusSecurityManager()
+ RetailCommerceProfileTools getProfileTools()
+ SiteManager getSiteManager()
+ void setNucleusSecurityManager(NucleusSecurityManager nucleusSecurityManager)
+ void setProfileTools(RetailCommerceProfileTools profileTools)
+ void setSiteManager(SiteManager siteManager)
# void throwException(Class<T> exceptionClazz, String shortMsg, String errorDesc)
# void throwException(Class<T> exceptionClazz, String shortMsg, String errorDesc, Throwable cause)
# RepositoryItem validateCustomerId(CustomerRef customerRef)
# MutableRepositoryItem validateCustomerId(CustomerRef customerRef, boolean optional)
# RepositoryItem validateSite(LocationRef location)
ItemInformationServiceImpl
# RetailCatalogTools catalogTools
# ServiceMap<PropertyTypeHandler> extendedProductAttributeNames
# ServiceMap<PropertyTypeHandler> extendedSkuAttributeNames
# ItemInformationServiceTransformer transformer
...
# void applyCurrentLocale(Locale locale)
+ RetailCatalogTools getCatalogTools()
+ ServiceMap<PropertyTypeHandler> getExtendedProductAttributeNames()
+ ServiceMap<PropertyTypeHandler> getExtendedSkuAttributeNames()
+ ItemInformationServiceTransformer getTransformer()
+ ItmInfoColDesc retrieveItemInformation(ServiceOpContext ctx, ItmInfoCriVo input)
+ void setCatalogTools(RetailCatalogTools catalogTools)
+ void setExtendedProductAttributeNames(ServiceMap<PropertyTypeHandler> extendedProductAttributeNames)
+ void setExtendedSkuAttributeNames(ServiceMap<PropertyTypeHandler> extendedSkuAttributeNames)
+ void setTransformer(ItemInformationServiceTransformer transformer)
# void validateInput(ServiceOpContext ctx, ItmInfoCriVo input)
transformer
ItemInformationServiceTransformer
# RetailCatalogTools catalogTools
+ RetailCatalogTools getCatalogTools()
# ItmInfoDesc getInitialItmInfoDesc(String itemId)
+ void setCatalogTools(RetailCatalogTools catalogTools)
+ ItmInfoColDesc toItemInfoColDesc(RepositoryItem[] skuRepoItems, List<String> re
+ Locale toLocale(ItmInfoCriVo criteria)
+ List<String> toSkuIds(ItmInfoCriVo criteria)

Figure 11: Extension Points

To customize the input validation, override the protected void validateInput(ServiceOpContext ctx, ItmInfoCriVo input) method. By default, this method only validates that the caller provided a list of SKU IDs to query.

To customize the Locale handling, override the protected void applyCurrentLocale(Locale locale) method. By default, this method sets the locale property on the profile associated with the HTTP request to the language specified in the caller's input criteria.

To customize the format of the data returned by the service, replace the transformer associated with the service component. By default, the transformer property refers to the /retail/commerce/integration/catalog/ItemInformationServiceTransformer component backed by the oracle.retail.commerce.integration.catalog.ItemInformationServiceTransformer class. Any of the provided methods on this class can be overridden to provide custom behavior. See the class' Javadoc for more information.

Troubleshooting

This section covers the exceptions thrown by the service with possible causes.

Exception Thrown	**Possible Cause**
IllegalStateWSFaultException	An error occurred while attempting to access the product catalog. Generally wraps a Commerce RepositoryException. The input ItmInfoCriVo failed validation. Generally occurs when a need value missing.

Exception Thrown	**Possible Cause**
WebServiceException	Access denied. The Basic Authentication user associated with the request does not have permission to call the service.

Shopping Cart Service

Shopping Cart Service (also known as Online Shopping Cart Service) is one of the RXM's SOAP web services that can be called from external systems. This service provides customer's active shopping cart, based on customer ID.

There are three operations supported by this service:

- Retrieve shopping cart - This operation returns a cart with all items and their respective quantities.
- Remove item from shopping cart – Once a customer's shopping cart is retrieved, using this operation, items can be removed from shopping cart.
- Remove all items from shopping cart - This operation is used to remove all the items from a customer's shopping cart.

This service also supports multi-site. If multi-site is enabled in commerce, location ID (web-store ID) will also be a required input field. Multi-site is supported by all three operations.

ShoppingCart Service API

```
public ShoppingCartDesc retrieveShoppingCart(final ShoppingCartCriVo input)
 throws EntityNotFoundWSFaultException, IllegalArgumentWSFaultException,
 IllegalStateWSFaultException, ValidationWSFaultException;

public InvocationSuccess removeItemsFromCart(final ShoppingCartDelCriVo
shoppingcartdelcrivo)
 throws EntityNotFoundWSFaultException, IllegalArgumentWSFaultException,
 IllegalStateWSFaultException, ValidationWSFaultException;

public InvocationSuccess removeAllItemsFromCart(final ShoppingCartCriVo
shoppingcartcrivo)
 throws EntityNotFoundWSFaultException, IllegalArgumentWSFaultException,
 IllegalStateWSFaultException, ValidationWSFaultException;
```

Configuration

Implementation class for Shopping Cart Service interface is com.oracle.retail.ooc.integration.services.shoppingcartservice.v1.ShoppingCartServiceProviderImpl, which resides in 'Services' module. This class then delegates implementation to /oracle/retail/commerce/service/order/ShoppingCartService. This class, after validating the customer and site (if applicable), retrieves orders for that customer, and the first incomplete order is returned.

This class starts a transaction in case of removeItemsFromCart and removeAllItemsFromCart. This transaction removes items in order from repository. Because of this, whenever this service will be called, active shopping cart will go in bad state because of changes in repository. Always make sure to reload cart from repository.

This class has following properties:

Property Name	Property Function
cartOrderState	The state of the extracted order for customer depends on this property. Shopping cart represents orders in incomplete states. Default Value: incomplete
ascendingOrder	The order in which customer's orders will be extracted.Values are true for ascending order, false for descending. Default Value: false
sortOrdersByProperty	Order the results by the given item descriptor property. By default, orders are extracted by last modified date in descending order so that latest modified order can be displayed. Default Value: lastModifiedDate

This class also has few static final variables as well.

```
    /**
     * The index of the first order to return. (This is an index into the result
set return by the repository query.)
     */
    public static final int START_INDEX_OF_ORDER = 0;

    /**
     * The number of orders to return.
     */
    public static final int NUMBER_OF_ORDER = 1;
```

By default, the sort order is by last modified date in descending order, starting with the first order and returning one order.

Troubleshooting

Exceptions thrown by this service are:

Exception	Expected Release
com.oracle.retail.integration.services. exception.v1.IllegalArgumentWSFaultException	Input (com.oracle.retail.integration.base.bo. shoppingcartcrivo.v1.ShoppingCartCriVo) is null. In input, Customer is null. In input, location is null (in case of multisite) If item is to be removed, quantity is invalid.
com.oracle.retail.integration.services. exception.v1.EntityNotFoundWSFaultException	Customer does not exist or Profile is not found. No items in the shopping cart for the customer or no active shopping cart found. In case of remove item, item not found in repository. In case of multisite, location not found.
com.oracle.retail.integration.services exception.v1.IllegalStateWSFaultException	An error occurred accessing commerce repository. In case of remove items, error occurred while starting transaction or ending transaction.

Targeted Items Service

Note: Out of the box Oracle Commerce GSA IDs, such as sku IDs, are case sensitive.

In release 16.0, Xstore forces all primary keys (such as sku IDs) to uppercase in the database.

When Xstore is integrated with RXM, the forced-casing functionality has to be disabled in Xstore in order for the integration to work.

One of the SOAP web services that the RXM.Services module provides is the Targeted Items Service. This service relies on Oracle Commerce's framework for defining targeters and cross-sell/up-sell information. External systems can call this service. Based upon the input containing the current customer, the customer's current location (such as a retail store), and their current in-store purchase, RXM will call calculate what additional items are “targeted” to the customer’s online profile. This allows systems, such as Oracle Retail Xstore Point of Service, to show the customer the same “you may also like” content as they would see as if they were online.

In this case, the Commerce deployment acts as a service provider. In an Oracle Retail deployment, it is expected that Retail Service Backbone (RSB) is also involved, abstracting and virtualizing the service from consumers. No transformation of the SOAP payload in the RSB is needed because RXM hosts the same service contract for its services (or essentially RXM “speaks” RSB language for its hosted services).

Configuration

The Targeted Items Service is hosted by an RSB-generated JAXWS class, com.oracle.retail.ooc.integration.services.targeteditemsservice.v1.TargetedItemsServiceProviderImpl. This class delegates the service implementation to the /retail/commerce/service/targeting/TargetedItemsService component. This component first executes the targeter that is configured in its "targeterPath" property. The default targeter is /atg/registry/RepositoryTargeters/TargetedItems, which is provided out of the box (only as an example). The products or rules configured in the targeter are expected to be replaced by the merchant using Business Control Center (BCC).

The provider implementation class mentioned above is invoked by an RSB bean that implements the com.oracle.retail.ooc.integration.services.targeteditemsservice.v1.TargetedItemsServiceInterface interface. This interface is enhanced with a JAXWS @HandlerChain notation that specifies the oraclecommercehandlers.xml file to be found in the same package. This XML defines a list of handlers. The ServletPipelineHandler in this list invokes the Oracle Commerce Servlet Pipeline Chain before invoking the provider implementation class. This gives Nucleus knowledge of the current HTTP session and request. See “Web Services” section for more information.

Troubleshooting

A com.oracle.retail.integration.services.exception.v1.IllegalStateWSFaultException will occur if the /retail/commerce/service/targeting/TargetedItemsService cannot be found in the session scope or if the repository fails during the lookup of the customer, the location, or any items.

A com.oracle.retail.integration.services.exception.v1.IllegalArgumentWSFaultException will occur if the input is null.

A com.oracle.retail.integration.services.exception.v1.EntityNotFoundWSFaultException will occur if the customer ID or location ID is specified and it cannot be found using a query.

Security

For all hosted RXM services, the \atg\dynamo\servlet\dafpipeline\PathAuthenticationServlet Servlet Pipeline component is enabled. In this component the Targeted Items Service's context path is configured to use a /retail/commerce/service/security/WebServiceAuthenticator. This component is atg.servlet.pipeline.UserAuthorityAuthenticator that makes sure that users accessing the service have the webservices-user-group access role.

Also, the NucleusSecurityManager is used by the service to ensure the user has access to the service.

Security

Security for RXM is enabled by default. As stated in the *Oracle Commerce Retail Extension Module Security Guide*, web service security is comprised of four parts: authentication, authorization, confidentiality, and integrity. In the RXM.Services module, this primarily means application-level authorization. When fully configured, these four security aspects are accomplished.

PathAuthenticationServlet

RXM.Services by default enables the /atg/dynamo/servlet/dafpipeline/PathAuthenticationServlet Nucleus component within the servlet request pipeline. Additional authenticators are configured for each of the web service's context paths. Each is a UserAuthorityAuthenticator, which uses the database (see below) to check credentials and authorization.

NucleusSecurityManager

Each service component has the NucleusSecurityManager set.

- /retail/commerce/service/catalog/ItemInformationService
- /retail/commerce/service/order/ShoppingCartService
- /retail/commerce/service/targeting/TargetedItemsService

To disable the user authorization provided by these services, set the NucleusSecurityManager to null (that is, nucleusSecurityManager=). In this configuration, the service does not check functional access, so authorization for that service is disabled. Use this configuration if the application server (that is, WebLogic) is managing all user authentication and authorization duties.

Functional Access

Each service component checks the HTTP request's (see DynamoHttpServletRequest) user for access to that service's functional name. The user will not be set onto the DynamoHttpServletRequest unless PathAuthenticationServlet (see above) is enabled. In this case, access will be denied (unless NucleusSecurityManager is unset [see above]). The service's functional name is formatted as "Component Name <dot> Operation Name" (for example, TargetedItemsService.queryTargetedItems).

RXM.Services loads these functional names into the /atg/webservice/security/NucleusSecurityRepository during installation and database initialization. The database tables involved in the data setup of functions and users are:

- DAS_ACCOUNT
- DAS_GROUP_ASSOC
- DAS_NS_ACLS
- DAS_NUCL_SEC

User Setup

RXM.Services also loads a group (or role) called "webservices-user-group" into the /atg/dynamo/security/AdminSqlRepository during installation and database initialization. This group is given access to the above functional names.

The user that has the webservices-user-group role must be manually created. This user is not created by the installation or configuration process.

Extension

Developers wishing to extend these services may do so. However, consult the *Oracle Retail Service Backbone Development Guide* for how to extend the WSDLs and XSDs that define the contract of the service. Changes to the contract are reflected in a new version of ServiceProviderImpl and its payloads.

A copy of the generated ServiceInterface must be updated to include this line of code on its header to ensure the ServletPipelineHandler is invoked.

```
@HandlerChain(file="oraclecommercehandlers.xml")
```

Then the component at /retail/commerce/service/package/Service can be extended or have its class replaced to accept the different JAXB RSB objects.

The component at /retail/commerce/integration/package/ServiceTransformer must either then be replaced or extended per the extensions made to the service. Its job is to transform the RSB JAXB objects to and from service input and output, such as results from the product repository.

Troubleshooting

HTTP requests to the deployed JAX-WS SOAP web service musts provide a basic "Authorization" header token that matches the user setup in WebLogic and the same in the DAS_ACCOUNT table.

A bad username or password set up results in a "401 Unauthorized" response.

Here are additional options to set when making a request:

- Provide authorization credentials "preemptively". This assures the request has the "Authorization" header token when it is challenged by the app server and then passed to the application. If you do not provide an "Authorization" header preemptively, then you may see a response such as:

  ```
  <faultstring>Security token failed to validate.
  weblogic.xml.crypto.wss.SecurityTokenValidateResult@2971ce8d[status:
  false][msg UNT Error:A duplicated nonce is found!
  vRuyx7NIC2ChC8qHWlkheg==]</faultstring>
  ```

- Set the request option "WSS-Password Type" to "PasswordText". The application expects to be able to read the credentials as text. The credentials and request are encrypted by the TLS layer. If you do not set "PasswordText", then you may see a response such as:

```
<faultstring>Error on verifying message against security policy Error
codes: 1001 1021 Error codes: 1001 1021</faultstring>
```

- Set the request option "WSS TimeToLive" to 5000 (5 seconds) or similar. If you do not set "WSS TimeToLive", then you may see a response such as:

  ```
  <faultstring>Timestamp validation failed.</faultstring>
  ```