

# Integrating Oracle CX Commerce and Oracle Retail Order Management System



F32076-01  
August 2020



Integrating Oracle CX Commerce and Oracle Retail Order Management System,

F32076-01

Copyright © 2019, 2020, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

# Contents

## 1 Introduction

---

Audience	1-1
Features	1-1
Architectural Overview	1-1
Additional Documentation	1-2

## 2 Prerequisites

---

Access Rights	2-1
Data Configuration	2-1

## 3 Setting Up the Integration

---

Commerce Configuration	3-1
Accessing the Oracle Integrations Console	3-1
Configuring the Integration	3-2

# 1

## Introduction

Oracle Retail Order Management System Cloud Service (OROMS) is an order management system that supports retail transactions, including fulfillment, warehousing/inventory control, customer service, merchandising, marketing and finance.

Commerce and Oracle Retail Order Management can be used together to provide a robust commerce architecture.

## Audience

This document is written for Commerce and Oracle Retail Order Management administrators who need to set up and configure the integration between these two systems.

Readers of this document should have experience with both Commerce and Oracle Retail Order Management administration. This document does not provide instructions on configuring aspects other than integration for Commerce and Oracle Retail Order Management. For that information, refer to the product documentation.

## Features

The integration between Commerce and Oracle Retail Order Management provides a solution that combines the capabilities of these two products.

This integration provides the following features:

- Turning integration on and off using the Commerce administration interface
- Providing configuration based on your environment
- Pushing completed Commerce orders to Oracle Retail Order Management for fulfillment
- Retrieving and displaying Oracle Retail Order Management order status in Commerce

This integration provides retailers with an opportunity to manage order and fulfillment information.

## Architectural Overview

Shoppers use the Commerce storefront to place an order.

When a retailer enables integration, the order created in the storefront is sent to Oracle Retail Order Management where it is fulfilled. Commerce can obtain the details of the order from Oracle Retail Order Management and display the status in the customer storefront or the Agent Console of Commerce.

Commerce manages the promotions and discounts for the order and passes the final price of the order to Oracle Retail Order Management. The shipping methods in both systems are synchronized, ensuring that the customer is choosing from shipping methods that are available from Oracle Retail Order Management.

To perform data synchronization, Commerce communicates with Oracle Retail Order Management using REST. Commerce provides data output in JSON, while Oracle Retail Order Management exposes REST services that accept XML. Integration services take the JSON data from Commerce, convert it to XML using XSLT transformers, and send it to all Oracle Retail Order Management systems. In turn, XML data sent from Oracle Retail Order Management is read by the integration service and converted to JSON. The webhook target that communicates with Oracle Retail Order Management is set using the administration interface. If the integration is disabled, the transformation logic is skipped and the webhook behaves as a standard webhook.

Webhooks submit JSON, however, Serenade accepts only XML. When an order is submit, it is converted from JSON to XML. Webhooks support multiple target orders, and if the target contains

Integration services are configured using the Commerce administration interface.

When an order or order detail is queried in Commerce, a service call is made to Oracle Retail Order Management to get the latest status for that order. Orders that are created in Commerce are synchronized to Oracle Retail Order Management, and order status updates regarding fulfillment from Oracle Retail Order Management are requested by Commerce on demand.

## Additional Documentation

For additional information on Commerce:

For information on using Commerce, see the [Oracle CX Commerce documentation](#).

For information on customizing and extending Commerce, see the [Oracle CX Commerce documentation](#).

# 2

## Prerequisites

This section contains information on prerequisites needed before configuring the integration.

Before you begin the integration process, you must ensure that you have met the following conditions.

### Access Rights

To configure integration, you need to have administrator access to Commerce.

This allows you to configure the integration settings using the administration interface.

### Data Configuration

The product and SKU information should be in sync on both the Commerce and Oracle Retail Order Management servers.

The following describes the Item and SKU fields and the mapping between them:

#### Item Configuration

Oracle Retail Order Management allows the `ITEM_ID` field to contain a maximum of 12 alphanumeric characters. Therefore, items created with Commerce should contain no more than 12 alphanumeric characters, for example, `Item001`, `wallet`, etc.

#### SKU ID Configuration

The Oracle Retail Order Management system generated `short_sku_number` number is unique for a company as well as a site. As such, the Commerce SKU ID should also have a unique ID for the site. Note that the SKU ID must be numeric and limited to 7 characters.

#### Variant Configuration

Oracle Retail Order Management allows a maximum of three SKU attributes, for example, `color`, `size` and `collar`. Note that all items will have the same elements. For example, both the shirts and the shoe SKUs will contain the same variants, for example, `color`, `size`, and `collar`.

Commerce allows more than three variants, for example, `color`, `size`, `collar`, and `sleeve`. However, Oracle Retail Order Management cannot recognize more than three variants. If the SKU attribute in Oracle Retail Order Management and the SKU variant in Commerce both contain fewer than three entries, the SKU attribute will be the same.

While creating items and SKUs in Commerce and Oracle Retail Order Management, consider the following:

- When creating SKUs, limit variants to less than three

- Create SKUs in Oracle Retail Order Management. The same SKUs with the `short_sku_number` field can be manually exported and created in Commerce
- Create variants that are as general as possible so that they can be shared across items
- Before the integration process begins, the Commerce and Oracle Retail Order Management inventories should be synchronized

For detailed information on configuring and managing catalogs and SKUs, refer to the [Oracle CX Commerce documentation](#).

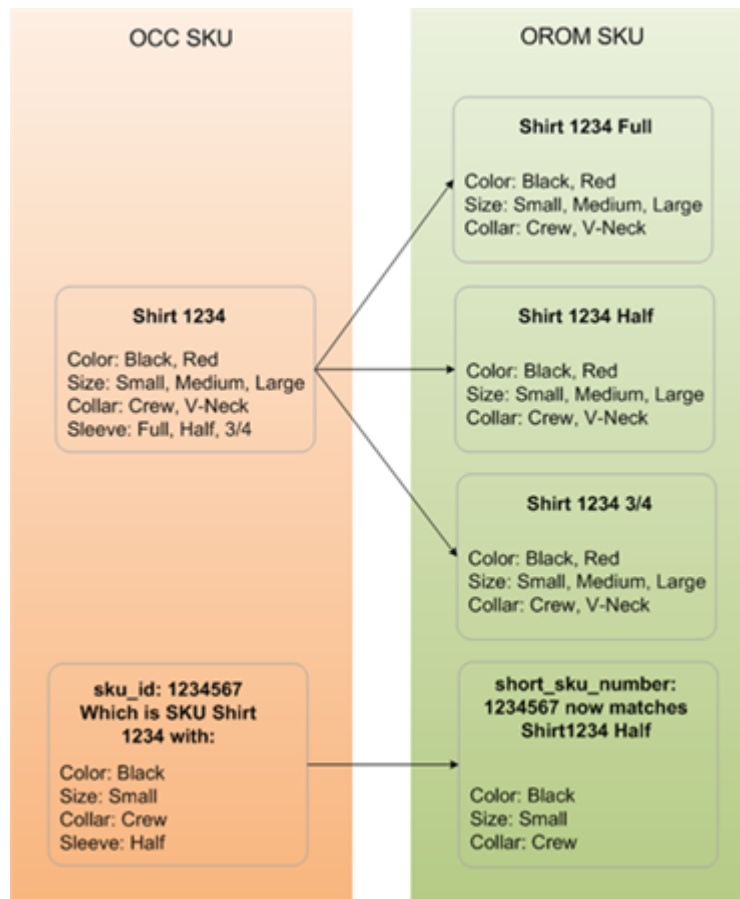
### Working with More Than Three Variants

Commerce allows you to configure SKUs so that they contain variants, such as `color`, `size`, `collar`, etc. However, Oracle Retail Order Management can accept only up to three of these variants. In the following example, the SKU for a shirt, as it is defined in Commerce, contains the variants for `color`, `size`, `sleeve` and `collar`. Because Oracle Retail Order Management can only accept three of these variants to be part of the SKU, the SKU is could be modified.

**Note:** If a Commerce SKU contains three or less variants, it is converted into the Oracle Retail Order Management system without any changes.

In the following example, the Commerce Shirt 1234 item contains the following variants: `color`, `size`, `collar`, and `sleeve`. A SKU with the `sku_id` 1234567 is a Shirt1234 item that has the variants values of `black`, `small`, `crew`, `half`.

Because the Commerce SKU has more variants than Oracle Retail Order Management can accept, you might configure your SKU so that the last variant, `sleeve`, is removed, creating three separate items: Shirt1234 Full, Shirt 1234 Half and Shirt 1234 3/4. The Commerce SKU now matches the Oracle Retail Order Management Shirt1234 Half item. The SKU with the Commerce `sku_id` 1234567 receives the system-generated `short_sku_number` 1234567.



The above diagram shows how the Commerce item Shirt 1234 can be converted into three separate Oracle Retail Order Management items based on the fourth variant, sleeve. The diagram also shows how the Commerce SKU Shirt1234 relates to the Oracle Retail Order Management SKU 1234567.



# 3

## Setting Up the Integration

The following section provides information on configuring and accessing integration.

This includes configuring both the Oracle CX Commerce and the Production Integration.

### Commerce Configuration

Before you set up the Oracle Retail Order Management integration with Oracle CX Commerce ensure that the necessary integration software is running on your server.

#### Configuring Webhooks

As described in the Architectural Overview section, the integration service is based on the Web API settings in Commerce.

Web APIs allow you to subscribe to events for your products and orders by creating webhooks that push JSON notifications to a URL you specify. For additional information on webhooks, refer to Use Webhooks.

Before you can configure the integration settings, the webhook must be configured. To do this, ensure that the Order Submit Event API has been configured with the URL necessary to connect to the Oracle Retail Order Management server.

#### To Configure the Order Submit Event API

1. From the Commerce administration interface, select the **Settings** menu.
2. Open the Web API page and select the **Webhook** tab.
3. Open the Order Submit Event API.
4. Provide the URL of the server that will be accepting the order, including the server name and port used for the CWOrderIn service. For example:  
`https://my.example.com:8443/SerenadeSeam/sxrs/SerenadeREST/CWOrderIn`
5. Provide a user name and password for accessing the server.
6. If you are using HMAC authentication, you can view or reset the key.
7. Click **Save** to save your changes.

### Accessing the Oracle Integrations Console

The Oracle Integrations page is accessed through the Commerce administration interface.

1. In the Commerce administration interface, click the **Settings** icon.
2. Click **Oracle Integrations** in the left navigation pane.
3. Select OROMS (Oracle Retail Order Management) from the Oracle Integrations menu.

- To enable the integration, select the **Enable integration** checkbox.

## Configuring the Integration

The following sections describe how to configure components of the integration procedure.

### Production Configuration

The Production Configuration section configures a number of details as well as the mappings for payment types and shipping methods. These values are passed to Oracle Retail Order Management allowing the orders to be identified.

**Note:** These settings are duplicated in the other Oracle Retail Order Management Integration setting, Preview Configuration.

The following properties are set in the fields provided by the Production Configuration page. Note that all fields are required unless otherwise noted.

Field	Description
XSLT Path	Name of the custom XSLT file to load.
Source	Identifies the source of the XML message. The source should default to IDC.
Target	Identifies the target of the XML message. The target should default to RDC.  If multiple webhooks are specified in the Target field, all of the systems receive the same data. Transformation for specific URLs can be performed by adding the URLs to both the webhook and integration settings.
Company Code	Identifies the company for the order. The company code is validated against the Company table.
Source Code	Updates the source code field in the Order Header table. The source code provides information about the currency code. You must provide the Commerce currency code to the Oracle Retail Order Management System source code mapping using the Currency Code mapping table.
Order Type	Updates the order type field in the Order Header table.
URL	The URL path to the Oracle Retail Order Management Web Service CWMessageIn. For example:  <code>https://my.example.com:8443/SerenadeSeam/sxrs/SerenadeREST/CWMessageIn</code>  <b>Note:</b> Use the same hostname and port number that you provided for the Order Submit URL on the Web APIs Webhook page.
User Name	The name of the user who should have access to the Web Service.

Field	Description
Password	The password associated with the user name. To see the password, click the Reveal button to display the password.

### Orders and multiple site configurations

The Commerce currency codes are mapped to the Oracle Retail Order Management Source Code. For example:

Commerce Currency Code	Oracle Retail Order Management Source Code
USD	CC_USD
EUR	CC_EUR

In an environment with multiple sites, you could configure the settings like this:

Site	Commerce Currency Code	Oracle Retail Order Management Source Code
USA	USD	CC_USD
UK	GBP	CC-UK_GBP
UK	EUR	CC_UK_EUR
Germany	EUR	CC_DE_EUR

Select the site with the site picker and select the appropriate Oracle Retail Order Management source code. Each source code points to a single offer record, which contains the currency code. This is how the offer record can provide the currency representation for an order.

### Using XSLT

You can use the customized XSLT capability to extend the default integration. The system contains an internal XSLT file that maps all attributes. The merchant does not have access to this XSLT file; however, you can provide a path to the XSLT. The XSLT should contain the logic that customizes the Oracle Retail Order Management `createOrder` payload so that it includes dynamic properties or makes mapping changes.

This XSLT workflow is active only when the Oracle Retail Order Management integration is enabled. The merchant must upload the XSLT using Oracle CX Commerce's file upload endpoints. If no customized XSLT file is uploaded, the system uses the default XSLT file to pass orders, ignoring any custom attributes.

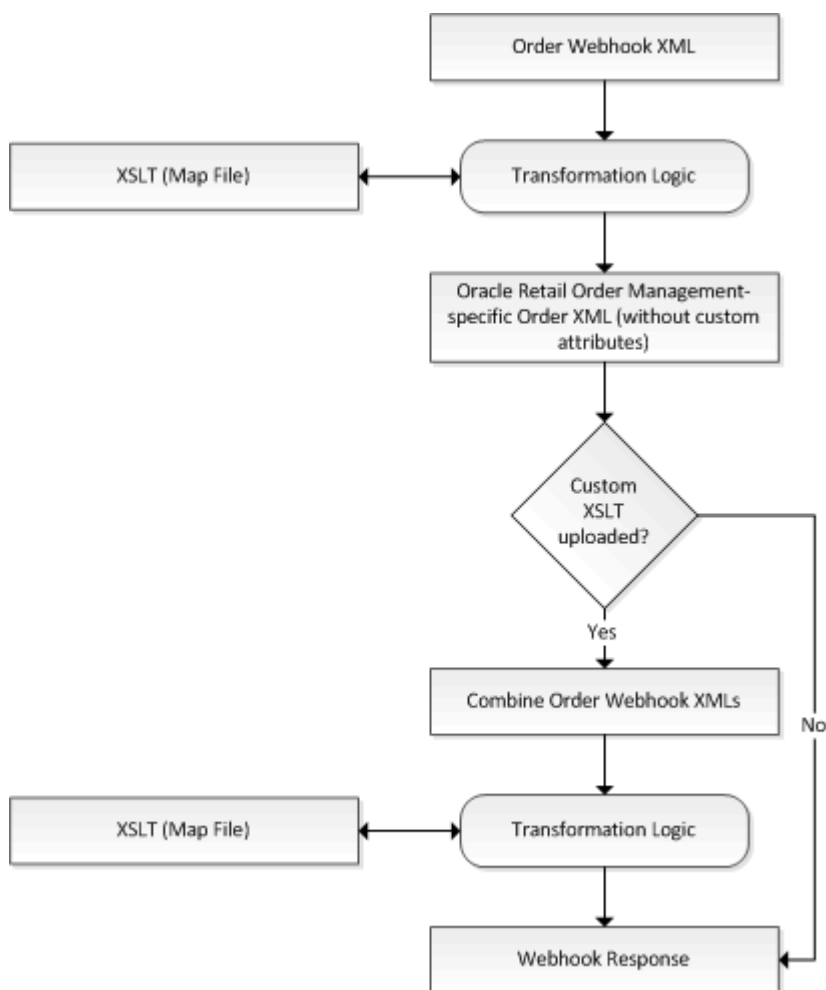
When creating a custom XSLT file:

- The default mapping can be overridden and mapped to other attributes as necessary.
- The merchant can use dynamic attributes created with Oracle CX Commerce to map attributes of the Oracle Retail Order Management.
- Because mapping is configured in the customized XSLT file, the merchant can map custom attributes that were created for an Order header level in Commerce to an order header or order line status in Oracle Retail Order Management.

To use a customized XSLT file, do the following:

1. Create a new XSLT file mapping.
2. Upload the XSLT file to Commerce using the file endpoints.
3. Use the Oracle Retail Order Management Integration Settings to provide the name of the uploaded XSLT.

The following illustration defines the way that the merchant's transformation is applied when the XSLT path is provided.



### XSLT Example

The following is an example of an XSLT file:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0"
  xmlns:str="http://exslt.org/strings" xmlns:exsl="http://exslt.org/
common"
  exclude-result-prefixes="str exsl">
<xsl:output indent="yes" method="xml" omit-xml-declaration="yes" />
<xsl:template match="/">
  <xsl:variable name="vOrder" select="request/atgResponse/order" />
  <Message>

```

```

<xsl:copy-of select="request/Message/@*" />
<Header>
  <xsl:copy-of select="request/Message/Header/@*" />
<Payments>
  <xsl:for-each select="request/Message/Header/Payments/
Payment">
    <Payment>
      <xsl:copy-of select="@*" />
    </Payment>
  </xsl:for-each>
</Payments>
<ShipTos>
  <xsl:for-each select="request/Message/Header/ShipTos/ShipTo">
    <ShipTo>
      <xsl:copy-of select="@*" />
      <Items>
        <xsl:for-each select="Items/Item">
          <Item>
            <xsl:copy-of select="@*" />
<!-- *****
Updating an existing mapping
***** -->
<!-- To change the mapping of an existing OROMS attribute, comment the
line
(<xsl:copy-of select="@*" />) and uncomment below line and replace
<oroms_attribute> with the attribute name required in output and
<occ_attribute>
with attribute name in occ XML map an attribute in oroms XML to a
different value,
comment the above line and uncomment below line and replace
<oroms_attribute> with
the attribute name required in output and the <occ_attribute> with
attribute name
in occ XML -->

<!-- <xsl:copy-of select="@*[name()!='<oroms_attribute>']" />
<xsl:attribute
name="tax_override"> <xsl:value-of select="//request/atgResponse/order/
<occ_attribute>" /> </xsl:attribute> -->

<!-- *****
Mapping a dynamic attribute of OCC to a new attribute in OROMS
***** -->
<!-- To add a new attribute "category" at item level in oroms XML,
which reads the
data from the dynamic attribute shopperCategory. Replace
<occ_attribute> with the
dynamic attribute name in occ. -->

<!-- <xsl:attribute name="category"> <xsl:value-of select="//request/
atgResponse/
order/<occ_attribute>" /> </xsl:attribute> -->

<!-- *****
Mapping a dynamic attribute of OCC with comma separated item level

```

```

data to a new attribute in OROMS
***** -->
<!-- To map a dynamic attribute in occ in format skuId1-value1,skuId2-
value2.
Replace <occ_attribute> with the dynamic attribute name in occ and
<oroms_attribute> with oroms attribute name. -->

<!-- <xsl:variable name="vOromsAttribute" select="@short_sku_number"/>

<xsl:for-each select="str:tokenize($vOrder/<occ_attribute>','')">
<xsl:variable name="temp" select="str:tokenize(.,'-')"/>
<xsl:if test="$temp[1]=$vOromsAttribute">
<xsl:attribute name="<oroms_attribute>"> <xsl:value-of
select="$temp[2]" />
</xsl:attribute>
</xsl:if>

</xsl:for-each> -->

<!-- *****
Mapping a dynamic attribute of OCC in JSON with a new attribute in OROMS
***** -->
<!-- To map a dynamic attribute in occ in json format (sample json
format is given
below ). Replace <occ_attribute> with the dynamic attribute name in occ,
<oroms_attribute> with oroms attribute name and
<dynamicAttributeFieldName> with
specific field name in the dynamic attribute json -->

<!-- Sample JSON:
[
  {
    "giftwraplineId":"gift-wrap-item-gwprod1001-834215",
    "giftWrapSkuId":"gwprod1001",
    "giftWrapSkuDescription":"Gift Wrap Product",
    "giftWrapSkuPrice":5,
    "skuId":"834215",
    "skuDescription":"Opal Innocence Silver 8\" Salad Plate",
    "quantity":1
  },
  {
    "giftwraplineId":"gift-wrap-item-gwprod1001-845353",
    "giftWrapSkuId":"gwprod1001",
    "giftWrapSkuDescription":"Gift Wrap Product",
    "giftWrapSkuPrice":5,
    "skuId":"845353",
    "skuDescription":"Bald Eagle Figurine",
    "quantity":1
  }
]
-->

<!-- <xsl:variable name="vOromsAttribute" select="@short_sku_number" />
<xsl:attribute name="<oroms_attribute>">
  <xsl:call-template name="readCustomProperty">

```

```

        <xsl:with-param name="json" select="$vOrder/<occ_attribute>" />
        <xsl:with-param name="skuId" select="$vOroomsAttribute" />
        <xsl:with-param name="dynamicAttributeName" select="
            <dynamicAttributeFieldName>" />
    </xsl:call-template>
</xsl:attribute>
-->
            </Item>
        </xsl:for-each>
    </Items>
</ShipTo>
</xsl:for-each>
</ShipTos>
</Header>
</Message>
</xsl:template>

<xsl:variable name="quot" select="'&quot;'" />
<xsl:variable name="skuIdWithQuots" select="concat('skuId\',$quot,
':\''
    $quot)" />

<xsl:template name="readCustomProperty">
    <xsl:param name="json" />
    <xsl:param name="skuId" />
    <xsl:param name="dynamicAttributeName" />

    <xsl:variable name="temp" select="normalize-space
        (substring-after($json,'{'})" />
    <xsl:variable name="fullSkuJson" select="normalize-space
        (substring-before($temp,'}'))" />
    <xsl:variable name="remainingJson" select="normalize-space
        (substring-after($temp,'}'))" />

    <xsl:call-template name="readCustomPropertyBySkuId">
        <xsl:with-param name="fullSkuJson" select="$fullSkuJson" />
        <xsl:with-param name="remainingJson" select="$remainingJson" />
        <xsl:with-param name="skuId" select="$skuId" />
        <xsl:with-param name="dynamicAttributeName" select="
            "$dynamicAttributeName" />
    </xsl:call-template>
</xsl:template>

<xsl:template name="readCustomPropertyBySkuId">
    <xsl:param name="fullSkuJson" />
    <xsl:param name="remainingJson" />
    <xsl:param name="dynamicAttributeName" />
    <xsl:param name="skuId" />
    <xsl:variable name="temp" select="normalize-space
        (substring-after($fullSkuJson,$skuIdWithQuots))" />
    <xsl:variable name="skuIdValue" select="normalize-space
        (substring-before($temp,'\'))" />

    <xsl:if test='$remainingJson'>
        <xsl:variable name="temp1" select="normalize-space

```

```

        (substring-after($remainingJson,'{'))" />
<xsl:variable name="fullSkuJson1" select="normalize-space
        (substring-before($temp1,'}'))" />
<xsl:variable name="remainingJson1" select="normalize-space
        (substring-after($temp1,'}'))" />
<xsl:call-template name="readCustomPropertyBySkuId">
    <xsl:with-param name="fullSkuJson" select="$fullSkuJson1" />
    <xsl:with-param name="remainingJson"
select="$remainingJson1" />
    <xsl:with-param name="skuId" select="$skuId" />
    <xsl:with-param name="dynamicAttributeName"
        select="$dynamicAttributeName" />
</xsl:call-template>
</xsl:if>

<xsl:if test='$skuId = $skuIdValue'>
    <xsl:variable name="attributeNameWithQuotes"
        select="concat($dynamicAttributeName, '\', $quot, ':\',
$quot)" />
    <xsl:variable name="temp2" select="normalize-space
        (substring-after($fullSkuJson,$attributeNameWithQuotes))" />
    <xsl:variable name="attributeValue" select="normalize-space
        (substring-before($temp2,'\'))" />
    <xsl:value-of select="$attributeValue" />
</xsl:if>
</xsl:template>
</xsl:stylesheet>

```

### Order Creation

Orders created in Commerce are sent to Oracle Retail Order Management using the Submit Order webhook.

Note that the Oracle Retail Order Management webhook overrides the Submit Order Webhook to send XML messages that contain the entire XML payload. If the webhook is configured to send to multiple destinations, all of the destinations will receive this XML payload.

The following is an XML example of an order creation message received by Oracle Retail Order Management.

### Sample Order Creation Message

```

<Message source="IDC" target="RDC" type="CWOrderIn">
  <Header order_number="o60412" order_type="Y" company_code="51"
order_channel="I"
    source_code="A123_USD" payment_only="N" response_type="E"
    order_date="08222016"
    sold_to_prefix="sold_to_fname="Kim" sold_to_lname="Anderson"
sold_to_suffix="sold_to_busres="R" sold_to_address1="21 Cedar Ave"
    sold_to_address2="" sold_to_address3="" sold_to_address4=""
    sold_to_city="Syracuse" sold_to_state="NY" sold_to_zip="13202"
    sold_to_country="US" sold_to_email_update="N"
sold_to_day_phone="212-555-977"
    sold_to_eve_phone="212-555-1977" sold_to_address_update="Y"
pay_incl="Y"

```



```

bill_to_prefix=""bill_to_fname="Kim" bill_to_lname="Anderson"
bill_to_suffix=""bill_to_address1="21 Cedar Ave" bill_to_address2=""
bill_to_address3=""bill_to_address4="" bill_to_city="Syracuse"
bill_to_state="NY"bill_to_zip="13202" bill_to_country="US"
bill_to_day_phone="212-555-1977" bill_to_eve_phone="212-555-1977"
bill_to_fax_phone=""bill_to_email="kim@example.com"
bill_to_company_name=""
ind_number=""order_email="kim@example.com"
alternate_sold_to_id="se-570031">
  <Payments>
    <Payment payment_type=""cc_number="9997000108950573"
cc_exp_month="03"
cc_exp_year="2018" auth_amount="75.88" auth_date="01011970"
ecommerce_indicator="Y" already_tokenized="Y"
transaction_id="lni4eg211lj6iqt6097hopidv7"
vendor_response="100"/>
  </Payments>
  <ShipTos>
    <ShipTo shipping_method="01" freight_tax_override="Y"
freight_tax_amount="4.22" calc_frt="N" ship_to_fax_phone=""
ship_to_evening_phone="212-555-1977"
ship_to_day_phone="212-555-1977"
ship_to_email=kim@example.com ship_to_zip="13202"
ship_to_state="NY"
ship_to_country="US" ship_to_city="Syracuse" ship_to_address3=""
ship_to_address2="" ship_to_address1="21 Cedar Ave"
ship_to_company=""
ship_to_suffix=""ship_to_lname="Anderson" ship_to_fname="Kim"
ship_to_prefix="" freight="50"contact_name="KimAnderson">
    <Items>
      <Item tax_override="Y" price_override="Y"
short_sku_number="130"
item_id="LAPTOP" actual_price="6.67" tax_amount="1.11"
quantity="2"/>
      <Item tax_override="Y" price_override="Y"
short_sku_number="130"
item_id="LAPTOP" actual_price="6.66" tax_amount="0.55"
quantity="1"/>
    </Items>
  </ShipTo>
</ShipTos>
</Header>
</Message>

```

### Order Status Updates

Commerce retrieves the order status and tracking information from the Oracle Retail Order Management System to display in the client. The status, which is obtained when an order detail is queried, will not be persisted or updated in the repository.

### Promotions

Promotions and offers are handled by Commerce with the corresponding discount price sent as part of the order.

### Returns and Exchanges

By default, the Oracle Retail Order Management integration does not support returns and exchanges.

### Configurable Product Support

By default, the Oracle Retail Order Management integration does not support configurable products.

### Payment Methods

Oracle Retail Order Management uses payment methods that are identified with unique integers. For integration, Commerce Payment Methods are mapped to Oracle Retail Order Management numeric payment methods.

You can obtain a list of the payment methods available by calling the following endpoint:

```
GET /ccadmin/v1/merchant/paymentGateways
```

This returns a list of available payment methods. The response also includes the ID of the merchant, whether the method has been enabled, and the repository ID of the payment method. The repository ID identifies the payment method code to use. The following example displays a partial response that identifies a CyberSource payment method:

```
{
  "paymentGateways": [
    {
      "sopCredentials": {
        "storefront": {
          "sopURL": "http://10.101.101.101:8080/ccstore/v1/PM/
cybersourceSOP",
          "expirationDate": "2019-01-28T11:54:30.207Z",
          "profileId": "Admin",
          "applicationName": "storefront",
          "hasSecretKey": true,
          "hasAccessKey": true,
          "repositoryId": "SOP-A"
        }
      }
    }
  ],
}
```

In an environment with multiple sites, you need to pass the `siteId` value in header to get payment gateway details for the site. To fetch payment gateways specific to a site, you need to pass `x-ccsite` header. For example, to fetch payment gateways for `siteUS` site, the header should include `x-ccsite: siteUS`.

You can find information about individual endpoints in the REST API documentation that is available through the Oracle Help Center.

Note that this documentation reflects the most recent version of Commerce. If you are currently using an earlier version of Commerce, the API documentation on the Oracle Help Center may include endpoints that are not available on your version.

Once you have obtained the repository ID, use it to map the payment method as follows:

Cloud Commerce	OROMS
CS-A	SOP-A

Remove

Add

As orders are created in Commerce and passed to Oracle Retail Order Management, the payment mappings are passed to Oracle Retail Order Management for fulfillment and settlement with the required payment gateways.

**Note:** To ensure PCI compliance, no credit card or gift card numbers are sent to Oracle Retail Order Management. The transaction reference and the authorization ID are sent with the order.

### Payment Gateways

Commerce has preconfigured integrations with Chase and CyberSource payment gateways. By default, the CyberSource payment gateway is selected for the integration. The Oracle Retail Order Management System requires a credit card to be provided when using the Chase gateway, however, Commerce does not store credit cards and cannot provide them. As such, Chase credit cards are not supported with this integration. Oracle Retail Order Management supports CyberSource and PayPal gateways by default.

**Note:** When using the CyberSource payment gateway, you must provide the AuthNumber for reauthorization and settlement. Because Commerce does not store the AuthNumber, the field is set to NA by default. You must update this field with the AuthNumber values for the CyberSource gateway.

To configure integrations with other order processing systems, use the Generic Payment Framework, as described in the Oracle CX Commerce documentation. Generic payment is not supported by default; however, Oracle Retail Order Management APIs can support sending payment details separately for an order. Whenever a generic payment is used in Commerce, payment details are not sent and the order is put into an error state. Once you update the payment details for the order, the Oracle Retail Order Management System changes the order to an open state.

### External Pricing

Note that this integration does not support external pricing. If you are using a combination of external prices and sale prices from Commerce, this integration will average the prices of items of the same SKU. For example, a customer qualifies for a promotion in Commerce that enables him to purchase 5 hats for the sales price of \$5 each, instead of the list price of \$6. If the customer wants to buy 10 hats, in Commerce, 5 of the hats would be sold at \$5 and 5 of the hats would be sold at \$6 for a total of \$55. Oracle Retail Order Management does not differentiate between the price lists. Instead, it averages the final price between all of the hats, therefore all 10 hats are priced at \$5.50 each for a total of \$55. This may impact return processes, as in the above example, the return price of a hat would be \$5.50 and not \$5 or \$6.

### Payment Types

Payment Types are set up and configured using the Payment Processing setting and the Payment Types tab, which is described in the Oracle CX Commerce

documentation. For information on Oracle Retail Order Management payment types, refer to the Oracle Retail Order Management documentation.

### Shipping Methods

Oracle Retail Order Management uses shipping methods that are identified by unique integers. You can configure multiple shipping methods. For integration, Commerce shipping methods are mapped to Oracle Retail Order Management numeric shipping methods.

The shipping methods are sent to Oracle Retail Order Management during fulfillment. Commerce and Oracle Retail Order Management shipping methods should be synchronized. These mappings ensure that the orders created on Commerce refer to and use similar shipping methods chosen by the customer for shipping.

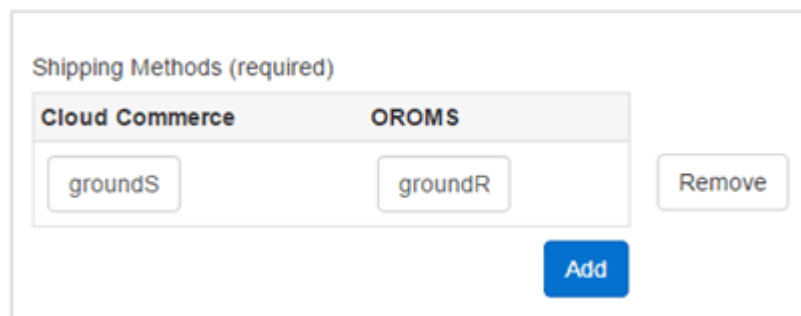
You can obtain a list of the shipping methods available by calling the following endpoint:

```
GET /ccadmin/v1/merchant/shippingMethods
```

This returns a list of available shipping methods. The response also includes information on each shipping method, and the repository ID of the shipping method. The repository ID identifies the shipping method code to use. The following example displays a partial response that identifies a ground shipping method:

```
"enabled": true,  
"displaySequence": 0,  
"eligibleForProductWithSurcharges": false,  
"ranges": [  
  {  
    "amount": 4.75,  
    "high": 14.99,  
    "low": 0,  
    "repositoryId": "groundRange1"  
  }  
]
```

Once you have obtained the repository ID, use it to map the shipping method:



Shipping methods are set up and configured using the Shipping Methods setting, which is described in the Oracle CX Commerce documentation. For information for integrating with external shipping systems, refer to the Oracle CX Commerce documentation. For information on Oracle Retail Order Management shipping methods, refer to the Oracle Retail Order Management documentation.

### External Shipping Methods

Commerce supports external shipping methods. However, these shipping methods may not be available in the list of shipping methods displayed in the administration

interface when configuring your Oracle Retail Order Management integration. To use the shipping methods for your external system you must add them to the Shipping Methods mapping table.

Once all of the integration parameters have been configured and saved, the integration process can occur.