ordergroove

&

Installation Guide

Version 19.2.0

Table of Contents

Summary

The purpose of this document is to help system integration partners and developers implement the Ordergroove solution as it pertains to the Salesforce Commerce Cloud platform.

Ordergroove provides frictionless shopping experiences and a recurring, predictable and profitable revenue stream for Salesforce customers through subscriptions, reorder, memberships, clubs and boxes. The cartridge allows for omnichannel subscriptions, secure payment, and access to Ordergroove's best-in-class recurring revenue platform, the Relationship Commerce Cloud.

The cartridge is fully compatible with the Storefront Reference Architecture (SFRA) and also has support for Site Genesis (Pipelines & SGJC).

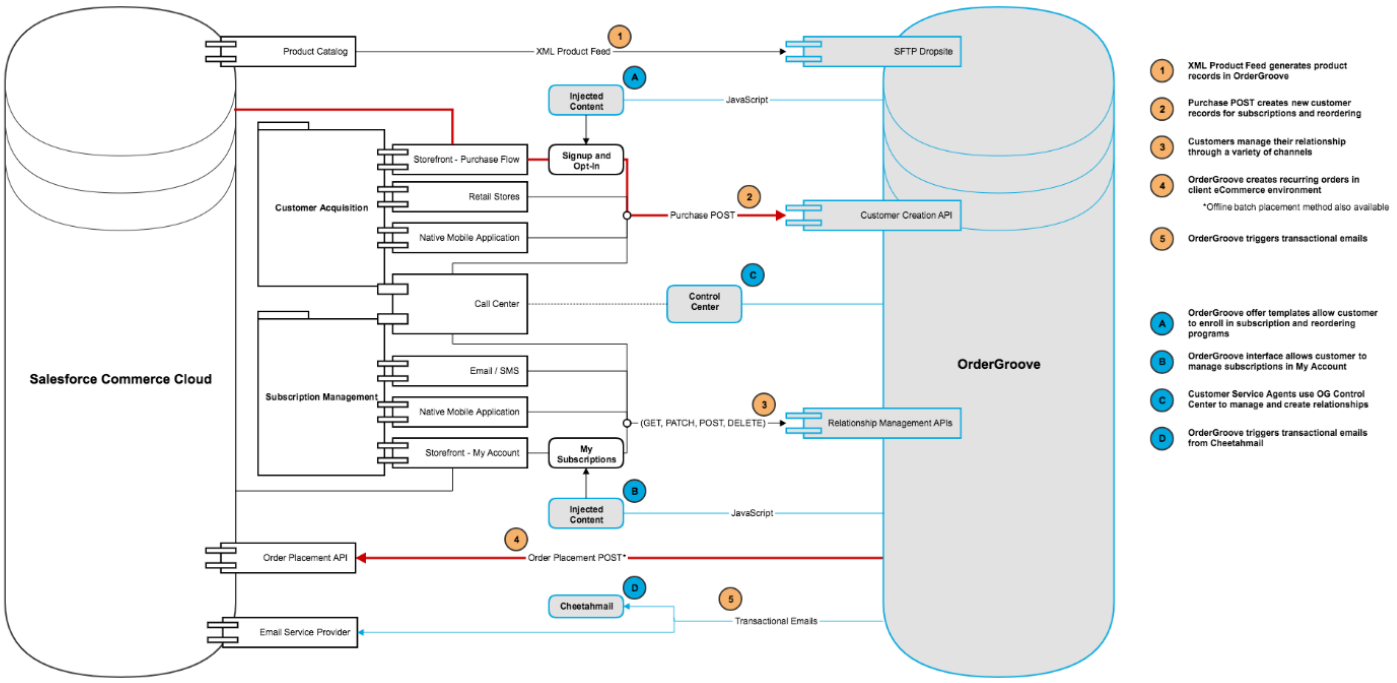
Component Overview

Functional Overview

The cartridge enables data flow between SFCC and Ordergroove through various API calls and feeds:

1. The product feed enables SFCC to send Ordergroove an updated list of products in the product catalog, along with prices and availability
2. Subscription creation is enabled through an HTTPS POST (referred to as Purchase Post), which informs Ordergroove when a checkout is completed; additional data is sent when there is a subscription product that is checked out by the customer
3. Customer information is kept in sync between the two systems
4. Order placement is enabled by creating a secure endpoint to which Ordergroove posts recurring orders for a customer on their behalf; this functionality hooks into core SFCC functionality, including, but not limited to, tax calculation, payment authorization, transactional emails, and order export for downstream fulfillment

A full list of features and supported use cases is provided below, along with an overall data flow diagram.



Use Cases

Subscription Creation

- Creation attempt happens only after successful payment authorization.
- Site preferences for subscription configuration.
- Email notifications to IT system support for specific error codes.
- Automatic retry logic and purging for failed attempts.

Recurring Orders

- XML Requests and Responses.
- Final price as calculated after promotions and incentives for line items in order.
- Orders flow as a normal storefront web channel.

Product Feed

- Site preferences for product feed configuration (price book ID & image view type).
- A scheduled job that can upload to the Ordergroove SFTP server as frequently as every 30 mins.

Customer Updates

- Synchronizes customer updates from account profile (phone number or email address) with Ordergroove.

Frontend Tagging

- Ordergroove “offers” (subscription enrollment widgets) use div and script tags that are utilized by JavaScript function calls.
- A base My Subscriptions Interface (MSI).

Signature Generation

- A hook extension which can be used to integrate with all Ordergroove APIs.

Limitations & Constraints

Depending on client-specific needs, there may be additional customizations necessary to fine-tune the Ordergroove solution to meet expectations. The storefront may require tweaks to element classes or promotional considerations, which are not available in this cartridge.

This cartridge does not take multiple or split payments into consideration. In addition, the cartridge does not handle multiple or split shipments to different shipping addresses. The first payment and default shipment will be used with this cartridge.

Compatibility

Certified and available since Commerce Cloud Store 19.10 (API Compatibility Mode 19.10) and SFRA 4.4.0. Also offers support of legacy pipeline implementations.

Privacy & Payment

Customer profile information being accessed is dependent on the selection of site preferences. Credit card account numbers can be AES encrypted and may be stored in Commerce Cloud for retry purposes only. All payment related information is AES encrypted in transit and at rest in Ordergroove’s database. All customer PII is AES encrypted at rest in Ordergroove’s database. In the case where Ordergroove must tokenize payment information, Ordergroove uses Paymetric as their tokenization vendor; Ordergroove is also PCI Level 1 Compliant.

Implementation Guide

Setup

The cartridge can be used with either Pipelines, Controllers, or Storefront Reference Architecture. One cartridge (int_ordergroove) is used for test and production environments containing all architectures. Only the merchant hash key, SFTP credentials, and service endpoint will differ based on environment. Generally, the cartridge is considered fully functional for Pipelines, SGJC, and SFRA. Be sure the following data is accessible:

Metadata	File Name
Custom & System Object Type Definitions	ordergroove-sys-custom-obj-defs.xml
Services	services.xml
Job Schedules	jobs.xml

The following table is a list of ISML templates in SFRA which will be overlaid with this cartridge:

File Path Name
account/dashboardProfileCards.isml
cart/productCard/cartProductCard.isml
common/scripts.isml
checkout/checkoutLogin.isml
checkout/confirmation/confirmation.isml
checkout/productCard/productCard.isml
product/productDetails.isml
product/quickView.isml

Configuration

Use a utility to create a zipped file for the **ordergroove-data-import** folder and navigate Business Manger into Adiministration >> Site Development >> Site Import & Export. Upload and import the zipped file, then skip to step 3. Otherwise, import the data manually by starting with step 1.

1. Import System & Custom Object Definitions
    a. Login to Business Manager
    b. Navigate: Administration >> Site Development >> Import & Export >> Upload
    c. Browse for **ordergroove-sys-custom-obj-defs.xml** and upload
    d. Click the back button and click the import metadata button
    e. Select **ordergroove-sys-custom-obj-defs.xml** and next button
    f. After the XML schema completes validation, choose import
2. Import Services & Job Schedules
    a. Navigate business manager: Administration >> Operations >> Import & Export >> Upload

b. Browse for **services.xml** and upload

c. Browse for **jobs.xml** and upload

d. Click the back button and click the import button under the services section

e. Select **services.xml** and next button

f. After the XML schema completes validation, choose import

g. Click the import button under the job schedules section

h. Select **jobs.xml** and next button

i. After the XML schema completes validation, choose import

3. Assign the cartridge to a site

   a. Navigate business manager: Administration >> Sites >> Manage Sites

   b. Select a site which will utilize the Ordergroove cartridge

   c. Choose the settings tab and add **int_ordergroove** to the beginning of the colon separated list if it is SFRA or to end the of the list if it is pipelines/controllers

   d. Repeat steps 3a through 3c for additional sites

   e. It is not necessary to add the int_ordergroove cartridge to the business manager site since the jobs are site-specific.

4. Update Site Preferences

   a. Navigate business manager with a site selected: Merchant Tools >> Site Preferences >> Custom Preferences >> Ordergroove

   b. Update every preference value as directed by the Ordergroove client solution partner

   c. Be sure to update the preferences based on instance type when promoting the integration to staging

5. Update URL Rules

   a. Navigate business manager with a site selected: Merchant Tools >> Site URLs >> URL Rules

   b. Select the pipeline URLs tab

   c. Create an alias **ordergroove** that resolves to **OrderGroove-OrderPlacement**

   d. Create an alias **auth** that resolves to **OrderGroove-Auth**

   e. Create an alias **authiframe** that resolves to **OrderGroove-AuthIframe**

   f. Create an alias **subscriptions** that resolves to **Account-MSI** (SFRA) or **OrderGroove-MSI** (SG)

   g. Provide the full URL based on step 5c to your Ordergroove client solution partner for all sites and environments where Ordergroove will be used (e.g. https://www.yoursite.com/ordergroove)

6. Update Services

   a. Navigate business manager: Administration >> Operations >> Services >> Service Credentials

   b. Choose **OrderGrooveCreateSubscription**

c. Update the URL based on environment:

Test	https://staging.sc. ordergroove.com/subscription/create
Production	https://sc. ordergroove.com/subscription/create

d. User and password should be empty

e. Choose **OrderGrooveCustomerUpdate**

f. Update the URL based on environment:

Test	https://staging.restapi. ordergroove.com/customers
Production	https://restapi. ordergroove.com/customers

g. User and password should be empty

h. Choose **OrderGrooveUploadFeed**

i. Update the URL based on environment:

Test	sftp://staging.feeds. ordergroove.com
Production	sftp://feeds. ordergroove.com

j. **User and password should be populated** and provided by your Ordergroove client solution partner (this will be different for **test and production)**

7. Update Job Schedules

a. Navigate business manager: Administration >> Operations >> Job Schedules

b. Choose **OrderGroovePurchasePostRetry**

c. Select the schedule and history tab to ensure the job is **enabled** and **recurring every 10 minutes** for a non-sandbox environment.

d. Select the resources tab and assign the job a system **resource lock** of **customobject**

e. Select the step configurator tab and choose the applicable sites as **scope**

f. Select the notification tab and enable/update the recipient notification **email address**

g. Choose **OrderGrooveProductFeed**

h. Select the schedule and history tab to ensure the job is **enabled** and **recurring every 30 minutes** for a non-sandbox environment.

i. Select the resources tab and assign the job a system **resource lock** of **feed**

j. Select the step configurator tab and choose the applicable sites as **scope**

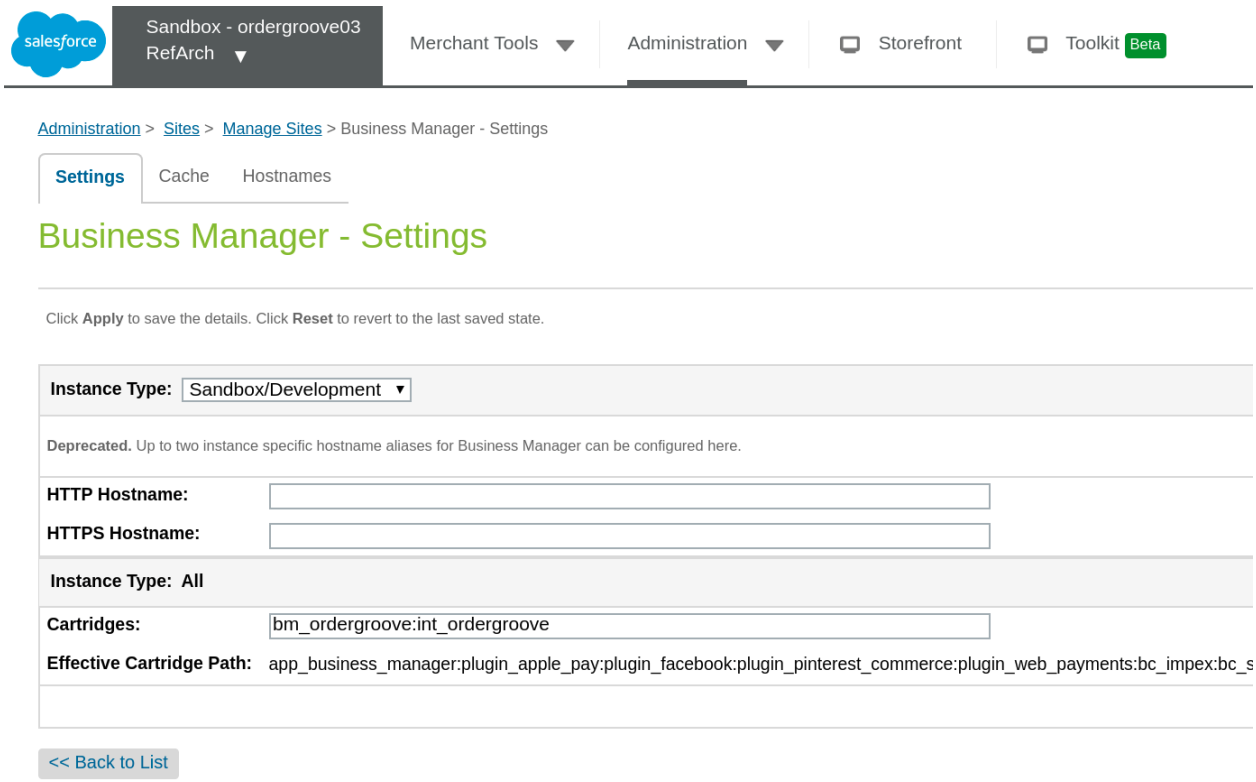
k. Select the notification tab and enable/update the recipient notification **email address**

Dashboard Configuration

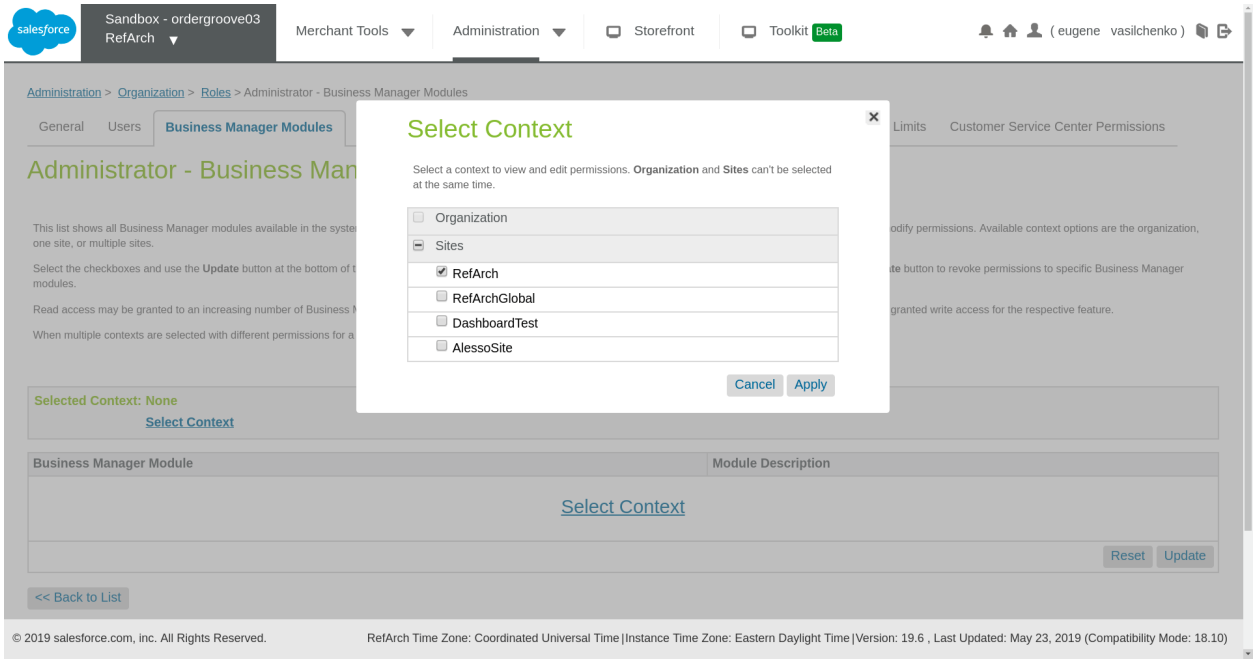
The Salesforce Ordergroove cartridge comes with a separate Business Manager plugin which allows you to login to and control the Ordergroove program directly from the Salesforce Business Manager interface. There are a couple of steps which are necessary to enable this functionality. Please take a look below for instructions.

Dashboard Setup

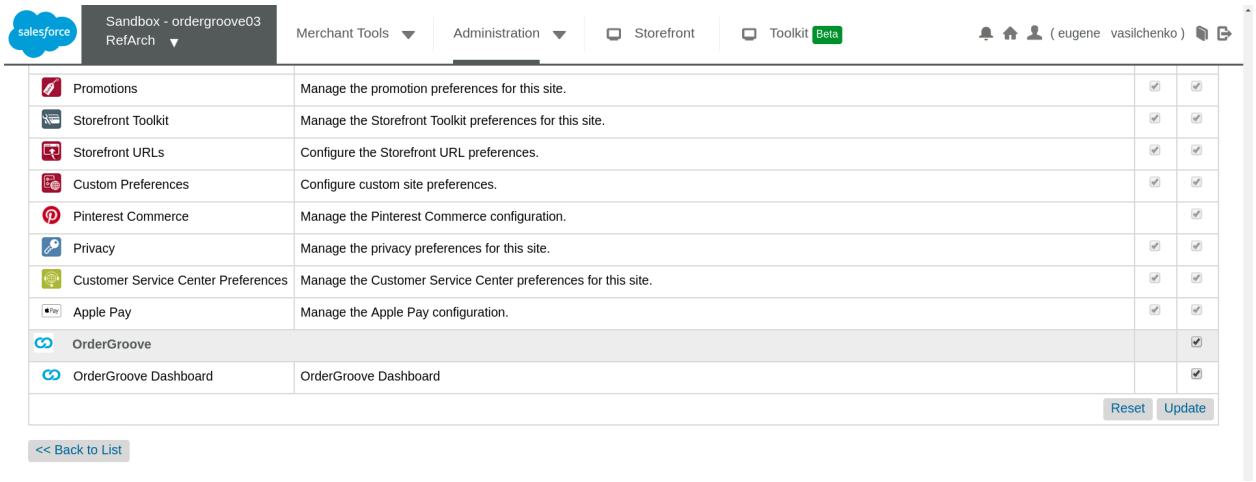
1. Navigate to the Business Manager settings screen and append the **bm_ordergroove** cartridge to the cartridge path:



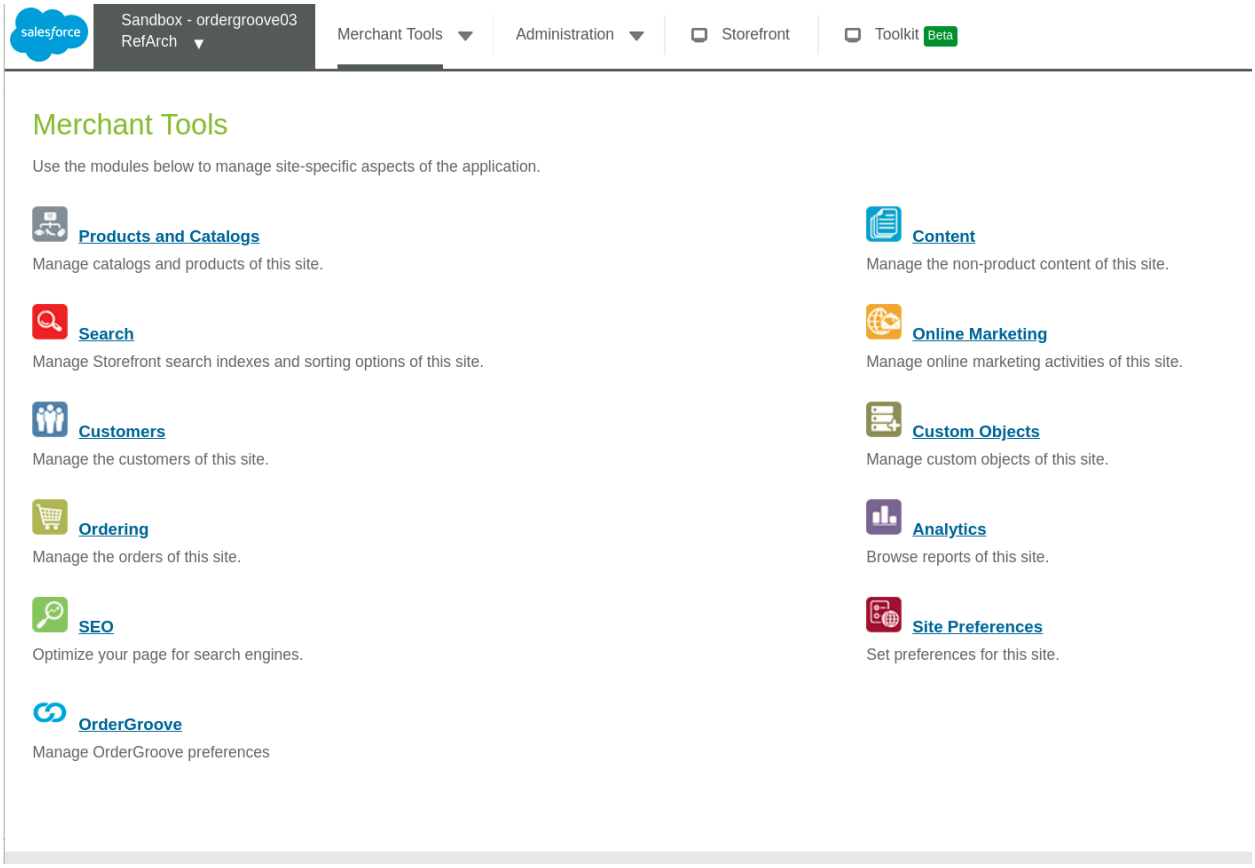
2. Once the **bm_ordergroove** cartridge has been added to Business Manager settings and your changes have been applied, you will need to configure permissions for the cartridge to enable Business Manager functionality. In order to do this, navigate to the Roles & Permissions area, choose a role and site for which you would like to enable this functionality.



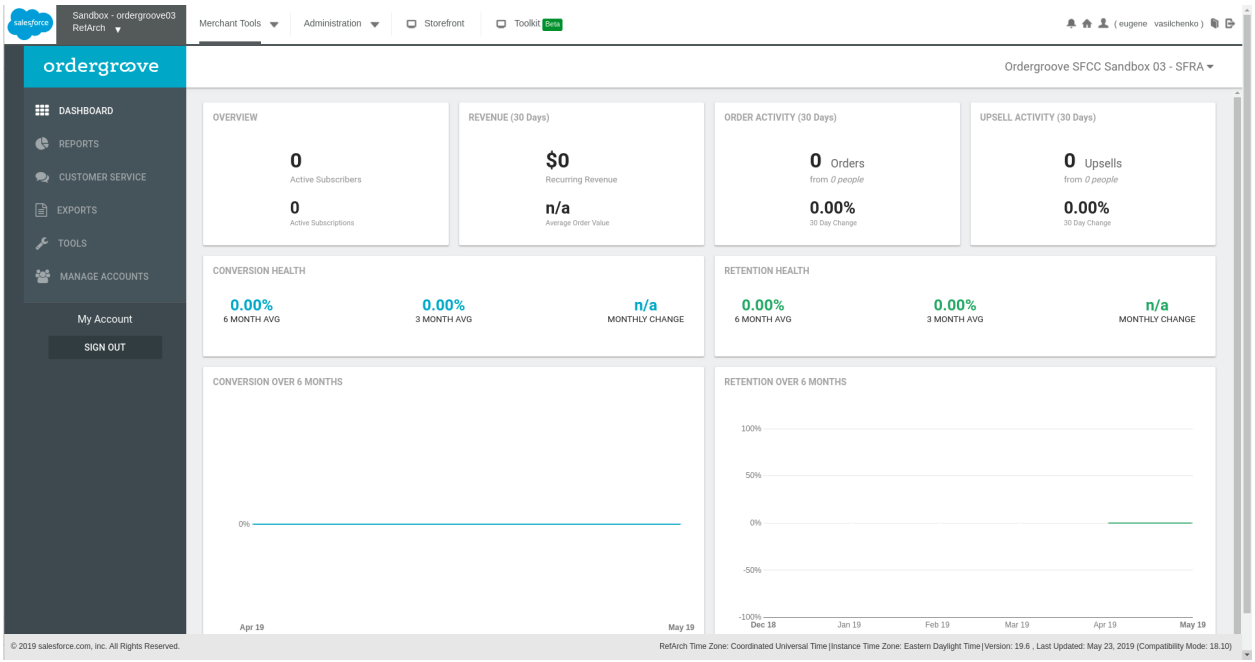
3. Once a site and role has been selected, just scroll down to the bottom of the page and enable the Ordergroove Dashboard:



Enabling the cartridge will add the Ordergroove specific navigation links in the Merchant Tools dropdown and section:



Once you navigate to Ordergroove dashboard, you can login using credentials provided to you by your Ordergroove contact and start managing your program:



Custom Code

Regardless of a site being SFRA, controller, or pipeline-based, there are two hooks and a preferred payment which need to be implemented for recurring orders in **OrderGroove.js**:

- Make the external tax call (if applicable)

```
// Pass basket and make external tax service call to override DW tax table.
if(HookMgr.hasHook('app.basket.calculate.taxify')) {
    //HookMgr.callHook('app.basket.calculate.taxify', 'Taxify', basket);
}
```

Replace the sample hook in **OrderGroove.js** with the vendor of your choice. If your site uses the platform's built-in taxation table, then no changes are necessary. If your site is pipeline-based, it might be necessary to segment existing taxation logic by using the **hooks.json** file. Alternatively, you can call the pipeline from the controller using the **Pipeline Class** if your site API is compatible.

- Make the external payment service provider (PSP) call

```
// Make external call to authorize credit card
if(HookMgr.hasHook('app.payment.processor.BASIC_CREDIT')) {
    authorizationResult = HookMgr.callHook('app.payment.processor.BASIC_CREDIT', 'Authorize', {
        Order: order,
        OrderNo: order.getOrderNo(),
        PaymentInstrument: opi
    });
```

Replace the BASIC_CREDIT hook in **OrderGroove.js** with the payment gateway of your choice. Again, if your site is pipeline-based, it might be necessary to segment existing payment authorization logic by using the **hooks.json** file. Alternatively, you can call the pipeline from the controller using the **Pipeline Class** if your site API is compatible.

Lastly, when using tokenization instead of sending the payment card number, it is assumed a **custom attribute named preferred** exists for the customer's wallet.

Make a call to the Ordergroove Purchase POST endpoint during checkout for subscription creation and analytics. The code will ensure payment is authorized and the order was placed successfully prior to making the purchase post call.

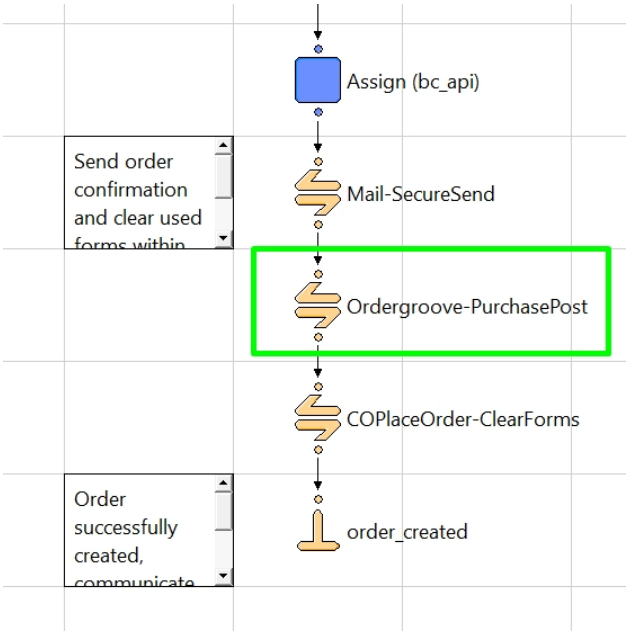
For SFRA, the controller **CheckoutServices.js** will overlay and append the **PlaceOrder** function so no action is necessary.

For controllers, open **COPlacerOrder.js** and add the following highlighted code to the appropriate section:

```
require('int_ordergroove/cartridge/scripts/purchasePost').orderNo(order.getOrderNo());
```

```
var orderPlacementStatus = Order.submit(order);
if (!orderPlacementStatus.error) {
    require('int_ordergroove/cartridge/scripts/purchasePost').orderNo(order.getOrderNo());
    clearForms();
}
return orderPlacementStatus;
```

For pipelines, open **COPlaceOrder.xml** and add the highlighted call node:



Integrate the customer update endpoint to keep Ordergroove synchronized with account profile changes.

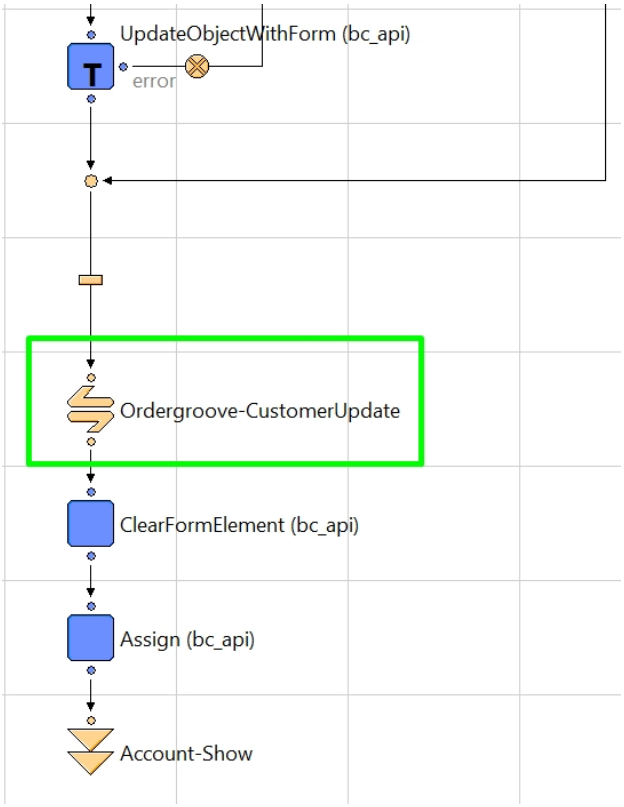
For SFRA, the controller **Account.js** will overlay and append the **SaveProfile** function so no action is necessary.

For controllers, open **Account.js** and add the following highlighted code to the appropriate section:

```
require('int_ordergroove/cartridge/scripts/customerUpdate').sync(customer.getProfile());
```

```
    if (isProfileUpdateValid && hasEditSucceeded) {
        require('int_ordergroove/cartridge/scripts/customerUpdate').sync(customer.getProfile());
        response.redirect(URLUtils.https('Account-Show'));
    } else {
        response.redirect(URLUtils.https('Account-EditProfile', 'invalid', 'true'));
    }
},
```

For pipelines, open **Account.xml** and add the highlighted call node:



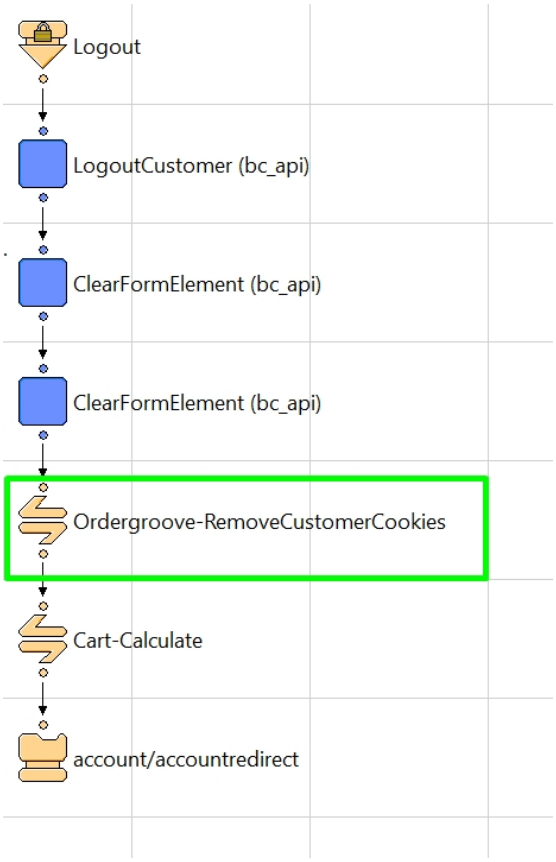
Add front-end page tagging and markup so that Ordergroove subscription offers will display throughout various areas on the site.

Pipelines

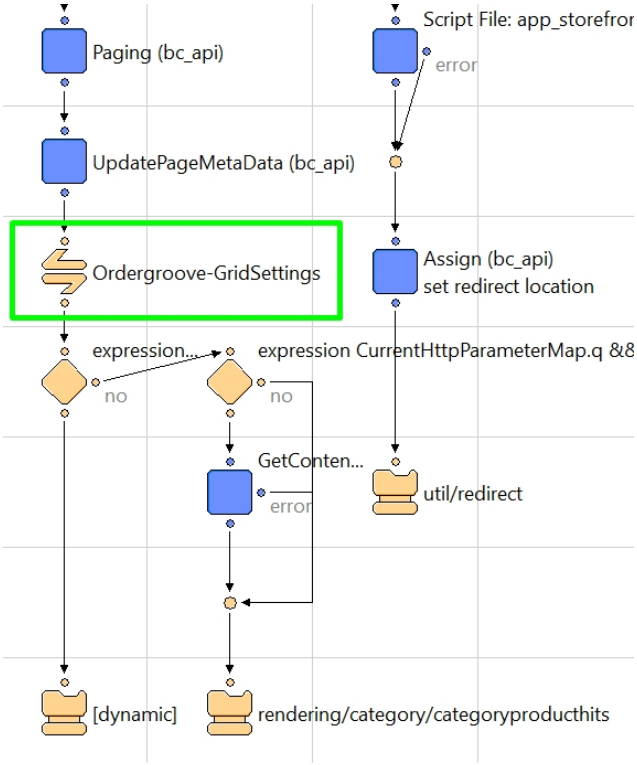
Since one cartridge is used for all architectures and controllers take precedence over pipelines, delete the following files from the int_ordergroove cartridge (**only if using pipelines**):

File Path Name
int_ordergroove/cartridge/controllers/Account.js
int_ordergroove/cartridge/controllers/Cart.js
int_ordergroove/cartridge/controllers/Login.js
int_ordergroove/cartridge/controllers/Order.js
int_ordergroove/cartridge/controllers/Product.js
int_ordergroove/cartridge/controllers/Search.js

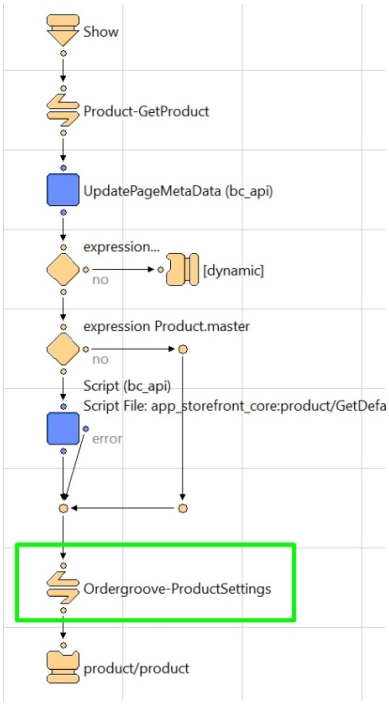
Modify **Login.xml** and add the highlighted call node:



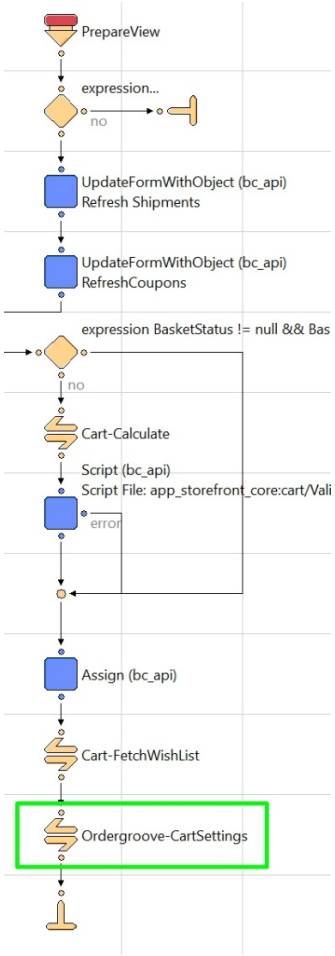
Modify **Search.xml** and add the highlighted call node:



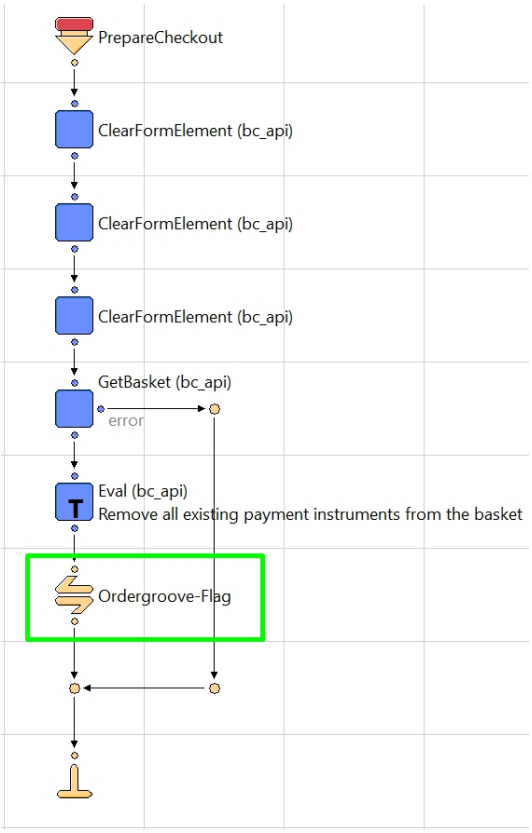
Modify **Product.xml** and add the highlighted call node:



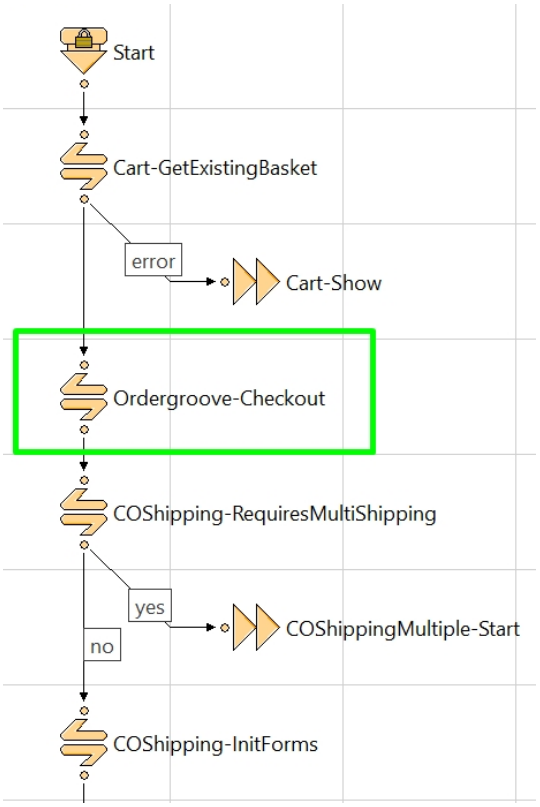
Modify **Cart.xml** and add the highlighted call node:



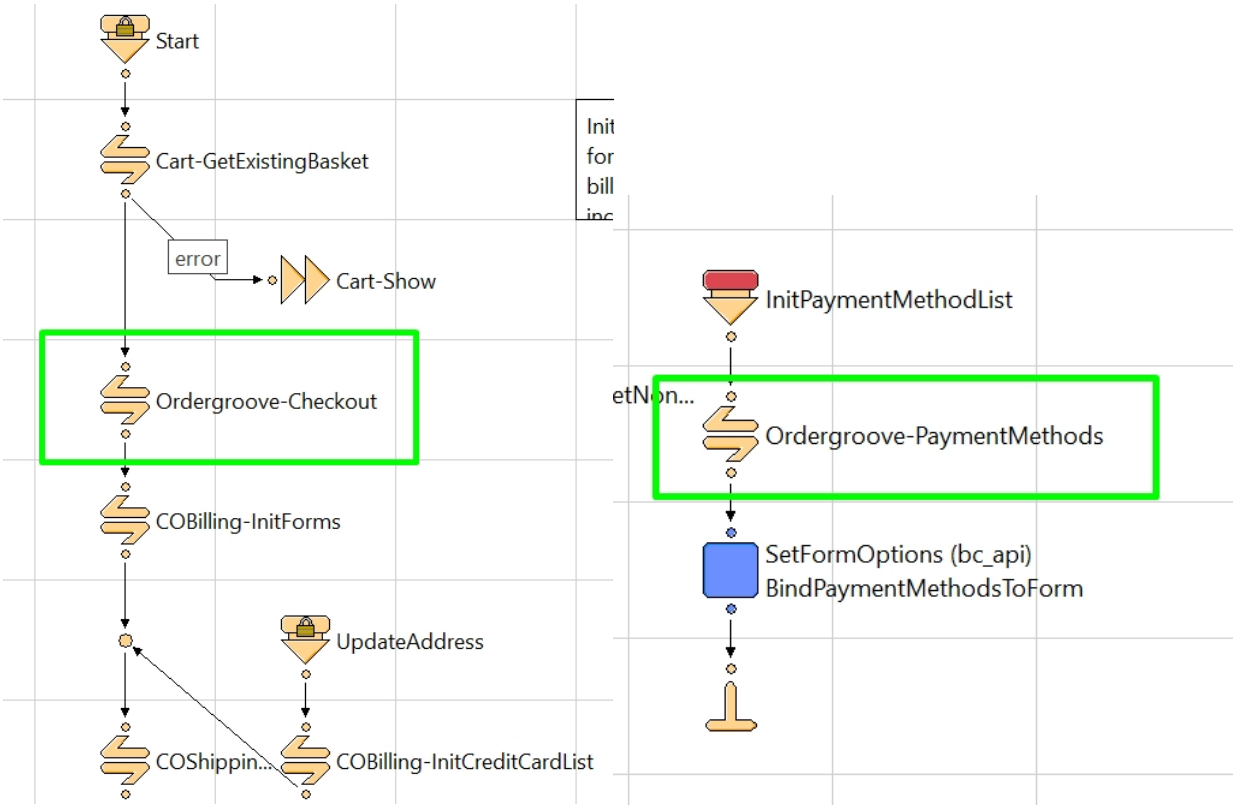
Modify **COCustomer.xml** and add the highlighted call node:



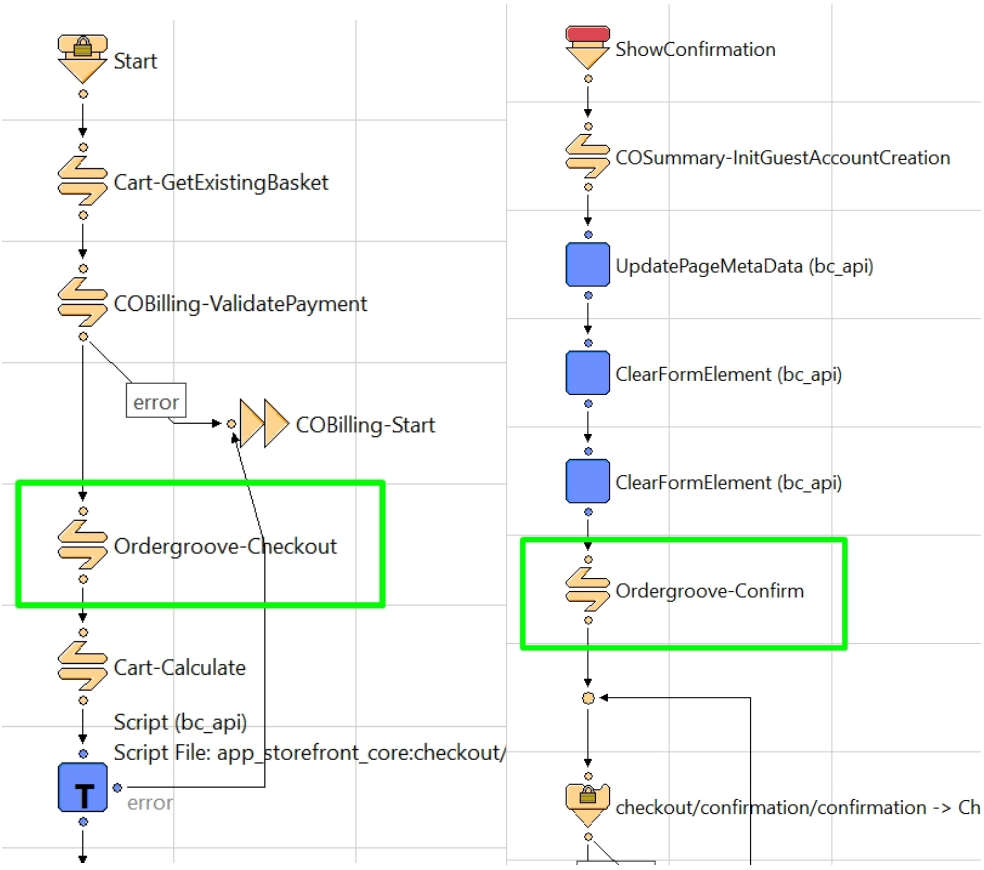
Modify **COShipping.xml** and add the highlighted call node:



Modify **COBilling.xml** and add the highlighted call nodes:



Modify **COSummary.xml** and add the highlighted call nodes:



Controllers (SGJC)

Modify **Login.js** and add the following line to the appropriate section:

```
require('int_ordergroove/cartridge/scripts/customerCookies').remove();
```

```
/**
 * Logs the customer out and clears the login and profile forms.
 * Calls the {@link module:controllers/Account~Show|Account controller Show function}.
 */
function Logout() {
    Customer.logout();
    require('int_ordergroove/cartridge/scripts/customerCookies').remove();

    app.getForm('login').clear();
    app.getForm('profile').clear();

    // TODO: Investigate whether this line should be removed
    //Cart.get().calculate();

    response.redirect(URLUtils.https('Account-Show'));
}
```

Modify **Search.js** and add the two lines to the appropriate sections:

```
var productSettings = require('int_ordergroove/cartridge/scripts/search').getSettings();
productSettings: productSettings,
```

```
function show() {

    var params = request.httpParameterMap;

    if (params.format.stringValue === 'ajax' || params.format.stringValue === 'page-element')
        // TODO refactor and merge showProductGrid() code into here
        showProductGrid();
        return;
    }

    var redirectUrl = SearchModel.getSearchRedirect(params.q.value);

    if (redirectUrl){
        app.getView({
            Location: redirectUrl.location,
            CacheTag: true
        }).render('util/redirect');
        return;
    }

    // Constructs the search based on the HTTP params and sets the categoryID.
    var Search = app.getModel('Search');
    var productSearchModel = Search.initializeProductSearchModel(params);
    var contentSearchModel = Search.initializeContentSearchModel(params);
    var productSettings = require('int_ordergroove/cartridge/scripts/search').getSettings();

    // execute the product search
    productSearchModel.search();
    contentSearchModel.search();

    if (productSearchModel.emptyQuery && contentSearchModel.emptyQuery) {
        response.redirect(URLUtils.abs('Home-Show'));
    } else if (productSearchModel.count > 0) {

        if ((productSearchModel.count > 1) || productSearchModel.refinedSearch || (contentSea
            var productPagingModel = new PagingModel(productSearchModel.productSearchHits, pr
            if (params.start.submitted) {
                productPagingModel.setStart(params.start.intValue);
            }

            if (params.sz.submitted && request.httpParameterMap.sz.intValue <= 60) {
                productPagingModel.setPageSize(params.sz.intValue);
            } else {
                productPagingModel.setPageSize(12);
            }

            if (productSearchModel.category) {
                meta.update(productSearchModel.category);
            }
            meta.updatePageMetaTags(productSearchModel);

            if (productSearchModel.categorySearch && !productSearchModel.refinedCategorySearc
                // Renders a dynamic template.
                app.getView({
                    productSettings: productSettings,
                    ProductSearchResult: productSearchModel,
                    ContentSearchResult: contentSearchModel,
                    ProductPagingModel: productPagingModel
                }).render(productSearchModel.category.template);
            } else {
```

Modify **Product.js** and add the two lines to the appropriate sections:

```
var productSettings = require('int_ordergroove/cartridge/scripts/product').getSettings(product);
productSettings: productSettings,
```

```
function show() {

    const Product = app.getModel('Product');
    let product = Product.get(params.pid.stringValue);
    const currentVariationModel = product.updateVariationSelection(params);
    product = product.isVariationGroup() ? product : getSelectedProduct(product);
    var productSettings = require('int_ordergroove/cartridge/scripts/product').getSettings(product);

    if (product.isVisible()) {
        meta.update(product);
        meta.updatePageMetaTags(product);
        app.getView('Product', {
            product: product,
            productSettings: productSettings,
            DefaultVariant: product.getVariationModel().getDefaultVariant(),
            CurrentOptionModel: product.updateOptionSelection(params),
            CurrentVariationModel: currentVariationModel
        }).render(product.getTemplate() || 'product/product');
    } else {
```

Modify **Cart.js** and add the two lines to the appropriate sections:

```
var productSettings = require('int_ordergroove/cartridge/scripts/cart).getSettings();
productSettings: productSettings,
```

```
function show() {
    var cartForm = app.getForm('cart');
    app.getForm('login').invalidate();
    var productSettings = require('int_ordergroove/cartridge/scripts/cart').getSettings();

    cartForm.get('shipments').invalidate();

    app.getView('Cart', {
        cart: app.getModel('Cart').get(),
        productSettings: productSettings,
        RegistrationStatus: false
    }).render('checkout/cart/cart');
```

Modify **COCustomer.js** and add the two parts to the appropriate sections:

```
var autoShip = false;
if (dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer') === null ||
dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer') === true) {
            autoShip = dw.system.HookMgr.callHook('ordergroove.customer', 'isAutoShip',
request.getHttpHeaders().get('cookie'));
        } else {
            var BasketMgr = require('dw/order/BasketMgr');
            var basket = BasketMgr.getCurrentOrNewBasket();
            autoShip = dw.system.HookMgr.callHook('ordergroove.customer', 'isNewAutoShip',
basket);
        }
autoShip: autoShip,
```

```
function start() {
    var oauthLoginForm = app.getForm('oauthlogin');
    app.getForm('singleshipping').clear();
    app.getForm('multishipping').clear();
    app.getForm('billing').clear();

    Transaction.wrap(function () {
        Cart.goc().removeAllPaymentInstruments();
    });

    // Direct to first checkout step if already authenticated.
    if (customer.authenticated) {
        response.redirect(URLUtils.https('COShipping-Start'));
        return;
    } else {
        var loginForm = app.getForm('login');
        loginForm.clear();
        oauthLoginForm.clear();

        // Prepopulate login form field with customer's login name.
        if (customer.registered) {
            loginForm.setValue('username', customer.profile.credentials.login);
        }

        var loginAsset = Content.get('myaccount-login');

        var pageMeta = require('~/cartridge/scripts/meta');
        pageMeta.update(loginAsset);
```

```
        var autoShip = false;
        if (dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer') === null || dw.system.Site.getCurrent
            autoShip = dw.system.HookMgr.callHook('ordergroove.customer', 'isAutoShip', request.getHttpHeaders().get('cookie'));
        } else {
            var BasketMgr = require('dw/order/BasketMgr');
            var basket = BasketMgr.getCurrentOrNewBasket();
            autoShip = dw.system.HookMgr.callHook('ordergroove.customer', 'isNewAutoShip', basket);
        }
```

```
        app.getView({
            autoShip: autoShip,
            ContinueURL: URLUtils.https('COCustomer-LoginForm').append('scope', 'checkout')
        }).render('checkout/checkoutlogin');
    }
```

Modify **COShipping.js** and add three lines to the appropriate section:

```
require('int_ordergroove/cartridge/scripts/autoShipCustomer').validate();
var productSettings = require('int_ordergroove/cartridge/scripts/checkout').getSettings();
productSettings: productSettings,
```

```
function start() {
    var cart = app.getModel('Cart').get();
    var physicalShipments, pageMeta, homeDeliveries;

    if (!cart) {
        app.getController('Cart').Show();
        return;
    }

    require('int_ordergroove/cartridge/scripts/autoShipCustomer').validate();

    // Redirects to multishipping scenario if more than one physical shipment is contained in the ba
    physicalShipments = cart.getPhysicalShipments();
    if (Site.getCurrent().getCustomPreferenceValue('enableMultiShipping') && physicalShipments && pl
        app.getController('COShippingMultiple').Start();
        return;
    }

    // Initializes the singleshipping form and prepopulates it with the shipping address of the defa
    // shipment if the address exists, otherwise it preselects the default shipping method in the fo
    if (cart.getDefaultShipment().getShippingAddress()) {
        app.getForm('singleshipping.shippingAddress.addressFields').copyFrom(cart.getDefaultShipmen
        app.getForm('singleshipping.shippingAddress.addressFields.states').copyFrom(cart.getDefault
        app.getForm('singleshipping.shippingAddress').copyFrom(cart.getDefaultShipment());
    } else {
        if (customer.authenticated && customer.registered && customer.addressBook.preferredAddress)
            app.getForm('singleshipping.shippingAddress.addressFields').copyFrom(customer.addressBo
            app.getForm('singleshipping.shippingAddress.addressFields.states').copyFrom(customer.ad
        }
    }
    session.forms.singleshipping.shippingAddress.shippingMethodID.value = cart.getDefaultShipment()

    // Prepares shipments.
    homeDeliveries = prepareShipments();

    Transaction.wrap(function () {
        cart.calculate();
    });

    // Go to billing step, if we have no product line items, but only gift certificates in the bask
    if (cart.getProductLineItems().size() === 0) {
        app.getController('COBilling').Start();
    } else {
        var productSettings = require('int_ordergroove/cartridge/scripts/checkout').getSettings();
        pageMeta = require('~/cartridge/scripts/meta');
        pageMeta.update({
            pageTitle: Resource.msg('singleshipping.meta.pagetitle', 'checkout', 'SiteGenesis Check
        });
        app.getView({
            productSettings: productSettings,
            ContinueURL: URLUtils.https('COShipping-SingleShipping'),
            Basket: cart.object,
            HomeDeliveries: homeDeliveries
        }).render('checkout/shipping/singleshipping');
```

Modify **COBilling.js** and add the lines to the appropriate sections:

```
var productSettings = require('int_ordergroove/cartridge/scripts/checkout').getSettings();
productSettings: productSettings,
```

```
function returnToForm(cart, params) {
    var pageMeta = require('~/cartridge/scripts/meta');

    // if the payment method is set to gift certificate get the gift certificate code from the
    if (!empty(cart.getPaymentInstrument()) && cart.getPaymentInstrument().getPaymentMethod() =
        app.getForm('billing').copyFrom({
            giftCertCode: cart.getPaymentInstrument().getGiftCertificateCode()
        });
    }

    pageMeta.update({
        pageTitle: Resource.msg('billing.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')
    });
```

```
    var productSettings = require('int_ordergroove/cartridge/scripts/checkout').getSettings();

    if (params) {
        app.getView(require('~/cartridge/scripts/object').extend(params, {
            productSettings: productSettings,
            Basket: cart.object,
            ContinueURL: URLUtils.https('COBilling-Billing')
        })).render('checkout/billing/billing');
    } else {
        app.getView({
            productSettings: productSettings,
            Basket: cart.object,
            ContinueURL: URLUtils.https('COBilling-Billing')
        }).render('checkout/billing/billing');
```

```
applicablePaymentMethods =
require('int_ordergroove/cartridge/scripts/autoShipPaymentMethods')
.filter(applicablePaymentMethods);
```

```
function initCreditCardList(cart) {
    var paymentAmount = cart.getNonGiftCertificateAmount();
    var countryCode;
    var applicablePaymentMethods;
    var applicablePaymentCards;
    var applicableCreditCards;

    countryCode = Countries.getCurrent({
        CurrentRequest: {
            locale: request.locale
        }
    }).countryCode;

    applicablePaymentMethods = PaymentMgr.getApplicablePaymentMethods(customer, countryCode, paymentAmount.value);
    applicablePaymentMethods = require('int_ordergroove/cartridge/scripts/autoShipPaymentMethods').filter(applicablePaymentMethods);
    applicablePaymentCards = PaymentMgr.getPaymentMethod(PaymentInstrument.METHOD_CREDIT_CARD).getApplicablePaymentCards(customer, c

    app.getForm('billing').object.paymentMethods.creditCard.type.setOptions(applicablePaymentCards.iterator());

    applicableCreditCards = null;

    if (customer.authenticated) {
```

```
require('int_ordergroove/cartridge/scripts/autoShipCustomer').validate();
```

```
function publicStart() {
    var cart = app.getModel('Cart').get();
    if (cart) {
        require('int_ordergroove/cartridge/scripts/autoShipCustomer').validate();

        // Initializes all forms of the billing page including: - address form - ema
        initAddressForm(cart);
        initEmailAddress(cart);

        var creditCardList = initCreditCardList(cart);
        var applicablePaymentMethods = creditCardList.ApplicablePaymentMethods;

        var billingForm = app.getForm('billing').object;
        var paymentMethods = billingForm.paymentMethods;
        if (paymentMethods.valid) {
            paymentMethods.selectedPaymentMethodID.setOptions(applicablePaymentMetho
        } else {
            paymentMethods.clearFormElement();
        }

        app.getForm('billing.couponCode').clear();
        app.getForm('billing.giftCertCode').clear();

        start(cart, {ApplicableCreditCards: creditCardList.ApplicableCreditCards});
    } else {
        app.getController('Cart').Show();
    }
```

Modify **COSummary.js** and add the lines to the appropriate sections:

```
require('int_ordergroove/cartridge/scripts/autoShipCustomer').validate();
var productSettings = require('int_ordergroove/cartridge/scripts/checkout').getSettings();
productSettings: productSettings,
```

```
function start(context) {
    var cart = Cart.get();

    // Checks whether all payment methods are still applicable. Recalculates all existing non-gift
    // instrument totals according to redeemed gift certificates or additional discounts granted th
    // redemptions on this page.
    var COBilling = app.getController('COBilling');
    if (!COBilling.ValidatePayment(cart)) {
        COBilling.Start();
        return;
    } else {
        Transaction.wrap(function () {
            cart.calculate();
        });

        Transaction.wrap(function () {
            if (!cart.calculatePaymentTransactionTotal()) {
                COBilling.Start();
            }
        });

        var pageMeta = require('~/cartridge/scripts/meta');
        require('int_ordergroove/cartridge/scripts/autoShipCustomer').validate();
        var productSettings = require('int_ordergroove/cartridge/scripts/checkout').getSettings();
        var viewContext = require('app_storefront_core/cartridge/scripts/common/extend').immutable
            productSettings: productSettings,
            Basket: cart.object
        });
        pageMeta.update({pageTitle: Resource.msg('summary.meta.pagetitle', 'checkout', 'SiteGenesis
        app.getView(viewContext).render('checkout/summary/summary');
    }
}
```

```
require('int_ordergroove/cartridge/scripts/cartAutoShipCookie').remove();
var productSettings = require('int_ordergroove/cartridge/scripts/confirmation').getSettings();
isConfirmStage: true,
productSettings: productSettings,
```

```
function showConfirmation(order) {
    if (!customer.authenticated) {
        // Initializes the account creation form for guest checkouts by populating the first and la
        // used billing address.
        var customerForm = app.getForm('profile.customer');
        customerForm.setValue('firstname', order.billingAddress.firstName);
        customerForm.setValue('lastname', order.billingAddress.lastName);
        customerForm.setValue('email', order.customerEmail);
        customerForm.setValue('orderNo', order.orderNo);
    }

    app.getForm('profile.login.passwordconfirm').clear();
    app.getForm('profile.login.password').clear();

    var pageMeta = require('~/cartridge/scripts/meta');
    pageMeta.update({pageTitle: Resource.msg('confirmation.meta.pagetitle', 'checkout', 'SiteGenes
    require('int_ordergroove/cartridge/scripts/cartAutoShipCookie').remove();
    var productSettings = require('int_ordergroove/cartridge/scripts/confirmation').getSettings();
    app.getView({
        isConfirmStage: true,
        productSettings: productSettings,
        Order: order,
        ContinueURL: URLUtils.https('Account-RegistrationForm') // needed by registration form afte
    }).render('checkout/confirmation/confirmation');
}
```

Pipelines & Controllers (SG storefront core)

The function calls are an important part of displaying Ordergroove offers and so the JavaScript files need to compile into app.js to successfully display. Running gulp or grunt should build these files successfully.

Modify **index.js** and add the two lines to the appropriate sections:

```
var ordergroove =
require('../../../../../int_ordergroove/cartridge/client/default/js/pages/product/pdp');
ordergroove();
```

The required ordergroove module is relative to the site genesis storefront core. If your cartridge structure is different, then you may have to modify the path to compensate. The module being required is based on the site genesis HTML structure. If elements, classes, or IDs were modified, then additional customization will be necessary in **pdp.js**.

```
'use strict';

var dialog = require('../../dialog'),
    productStoreInventory = require('../../storeinventory/product'),
    tooltip = require('../../tooltip'),
    util = require('../../util'),
    addToCart = require('./addToCart'),
    availability = require('./availability'),
    image = require('./image'),
    productNav = require('./productNav'),
    productSet = require('./productSet'),
    recommendations = require('./recommendations'),
    variant = require('./variant');

var ordergroove = require('../../../../../int_ordergroove/cartridge/client/default/js/pages/product/pdp');

/**
 * @description Initialize product detail page with reviews, recommendation and product navigation.
 */
function initializeDom() {
    productNav();
    recommendations();
    tooltip.init();
}

/**
 * @description Initialize event handlers on product detail page
 */
function initializeEvents() {
    var $pdpMain = $('#pdpMain');

    addToCart();
    availability();
    variant();
    image();
    productSet();
    if (SitePreferences.STORE_PICKUP) {
        productStoreInventory.init();
    }
    ordergroove();

    // Add to Wishlist and Add to Gift Registry links behaviors
    $pdpMain.on('click', '[data-action="wishlist"], [data-action="gift-registry"]', function () {
        var data = util.getQueryStringParams($('.pdpForm').serialize());
        if (data.cartAction) {
```

Modify **variant.js** and add the two lines to the appropriate sections:

```
var ordergroove =
require('../../../../../int_ordergroove/cartridge/client/default/js/pages/product/pdp');
ordergroove.updateProduct();
```

The required ordergroove module is relative to the site genesis storefront core. If your cartridge structure is different, then you may have to modify the path to compensate. The module being required is based on the site genesis HTML structure. If elements, classes, or IDs were modified, then additional customization will be necessary in **pdp.js**.

```
'use strict';

var ajax = require('../../ajax'),
    image = require('./image'),
    progress = require('../../progress'),
    productStoreInventory = require('../../storeinventory/product'),
    tooltip = require('../../tooltip'),
    util = require('../../util');

var ordergroove = require('../../../../../int_ordergroove/cartridge/client/default/js/pages/product/pdp');

/**
 * @description update product content with new variant from href, load new content to #product-content pa
 * @param {String} href - url of the new product variant
 **/
var updateContent = function (href) {
    var $pdpForm = $('.pdpForm');
    var qty = $pdpForm.find('input[name="Quantity"]').first().val();
    var params = {
        Quantity: isNaN(qty) ? '1' : qty,
        format: 'ajax',
        productlistid: $pdpForm.find('input[name="productlistid"]').first().val()
    };

    progress.show($('#pdpMain'));

    ajax.load({
        url: util.appendParamsToUrl(href, params),
        target: $('#product-content'),
        callback: function () {
            if (SitePreferences.STORE_PICKUP) {
                productStoreInventory.init();
            }
            image.replaceImages();
            tooltip.init();
            ordergroove.updateProduct();
        }
    });
};
```

Modify **cart.js** and add the two lines to the appropriate sections:

```
var ordergroove = require('../../../../int_ordergroove/cartridge/client/default/js/pages/cart');
ordergroove();
```

The required ordergroove module is relative to the site genesis storefront core. If your cartridge structure is different, then you may have to modify the path to compensate. The module being required is based on the site genesis HTML structure. If elements, classes, or IDs were modified, then additional customization will be necessary in **cart.js**.

```
'use strict';

var account = require('./account'),
    bonusProductsView = require('../bonus-products-view'),
    quickview = require('../quickview'),
    cartStoreInventory = require('../storeinventory/cart');

var ordergroove = require('../../../../int_ordergroove/cartridge/client/default/js/pages/cart');

/**
 * @private
 * @function
 * @description Binds events to the cart page (edit item's details, bonus item's actions, coupon
 */
function initializeEvents() {
    $('#cart-table').on('click', '.item-edit-details a', function (e) {
        e.preventDefault();
        quickview.show({
            url: e.target.href,
            source: 'cart'
        });
    })
    .on('click', '.bonus-item-actions a, .item-details .bonusproducts a', function (e) {
        e.preventDefault();
        bonusProductsView.show(this.href);
    });

    // override enter key for coupon code entry
    $('form input[name$="_couponCode"]').on('keydown', function (e) {
        if (e.which === 13 && $(this).val().length === 0) { return false; }
    });

    //to prevent multiple submissions of the form when removing a product from the cart
    var removeItemEvent = false;
    $('button[name$="deleteProduct"]').on('click', function (e) {
        if (removeItemEvent) {
            e.preventDefault();
        } else {
            removeItemEvent = true;
        }
    });
}

exports.init = function () {
    initializeEvents();
    if (SitePreferences.STORE_PICKUP) {
        cartStoreInventory.init();
    }
    ordergroove();
    account.initCartLogin();
};
```

Modify **app.js** and add the line to the appropriate section:

```
require('../../../int_ordergroove/cartridge/client/default/js/optins')();
```

The required module is relative to the site genesis storefront core. If your cartridge structure is different, then you may have to modify the path to compensate.

```
// if jQuery has not been loaded, load from google cdn
if (!window.jQuery) {
    var s = document.createElement('script');
    s.setAttribute('src', 'https://ajax.googleapis.com/ajax/libs/jquery/1.
    s.setAttribute('type', 'text/javascript');
    document.getElementsByTagName('head')[0].appendChild(s);
}

require('./jquery-ext')();
require('./cookieprivacy')();
consentTracking.init();
require('./captcha')();
require('../../../int_ordergroove/cartridge/client/default/js/optins')();

function initializeEvents() {
    var controlKeys = ['8', '13', '46', '45', '36', '35', '38', '37', '40'

    $('body')
        .on('keydown', 'textarea[data-character-limit]', function (e) {
```

Modify **checkoutlogin.isml** and add the following condition containing guest checkout so that it will not display:

```
<isif condition="${pdict.autoShip !== true}">
...
</isif>
```

Modify **minilineitems.isml** and add the following markup where desired:

```
                <iscomment>Ordergroove Offer</iscomment>
                <isif
condition="${!empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable'))
&& dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable') == true}">
                    <isif
condition="${empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer'
)) || dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer') == true}">
                        <isif condition="${pdict.p_showreverse}">
                            <div class="og-offer" data-og-module="cart_flydown" data-og-
product="${productLineItem.productID}"></div>
                        <iselse/>
                            <div class="og-offer" data-og-module="or" data-og-
product="${productLineItem.productID}"></div>
                        </isif>
                    <iselse/>
                        <og-offer product="${productLineItem.productID}"></og-offer>
                    </isif>
                </isif>
```

```
    <div class="mini-cart-pricing">

        <span class="label">${Resource.msg('global.qty','locale',null)}:</span>
        <span class="value"><isprint value="${productLineItem.quantity}"/></span>

        <isif condition="${productLineItem.bonusProductLineItem}">
            <isset name="bonusProductPrice" value="${productLineItem.getAdjustedPrice()}" scope="page"/>
            <isinclude template="checkout/components/displaybonusproductprice" />
            <isprint value="${bonusProductPriceValue}" />
        <iselse/>
            <isset name="productTotal" value="${productLineItem.adjustedPrice}" scope="page"/>
            <isif condition="${productLineItem.optionProductLineItems.size() > 0}">
                <isloop items="${productLineItem.optionProductLineItems}" var="optionLI">
                    <isset name="productTotal" value="${productTotal.add(optionLI.adjustedPrice)}" scope="page",
                </isloop>
            </isif>
            <span class="mini-cart-price"><isprint value="${productTotal}"/></span>
        </isif>

    </div>
```

```
    <iscomment>Ordergroove Offer</iscomment>
    <isif condition="${!empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable')) && dw.s
        <isif condition="${empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer'))
            <isif condition="${pdict.p_showreverse}">
                <div class="og-offer" data-og-module="cart_flydown" data-og-product="${productLineItem.productI
            <iselse/>
                <div class="og-offer" data-og-module="or" data-og-product="${productLineItem.productID}"></div>
            </isif>
        <iselse/>
            <og-offer product="${productLineItem.productID}"></og-offer>
        </isif>
    </isif>
```

```
</div>

<isset name="itemCount" value="${itemCount+1}" scope="page"/>
```

Modify **confirmation.isml** and add the following markup where desired:

```
<iscomment>Ordergroove Offer</iscomment>
<isinclude template="checkout/confirmation/confirmationCopy" />
```

```
    <div class="confirmation <isif condition="${!pdict.CurrentCustomer.authent
        <div class="confirmation-message">

            <h1>${Resource.msg('confirmation.thankyou','checkout',null)}</h1>

            <iscontentasset aid="confirmation-message" />

            <iscomment>Ordergroove Offer</iscomment>
            <isinclude template="checkout/confirmation/confirmationCopy" />
        </div>

        <div class="order-confirmation-details">
            <isorderdetails order="${pdict.Order}"/>
        </div>

        <isinclude template="checkout/confirmation/confirmationregister"/>
```

Modify **footer_UI.isml** and add the following markup before app.js:

```
<isinclude template="components/ordergroove"/>
```

```
<script src="${URLUtils.staticURL('/lib/jquery/ui/jquery-ui.min.js')}" type="text/javascript"></script

<iscomment>third-party add-ons</iscomment>
<script src="${URLUtils.staticURL('/lib/jquery/jquery.jcarousel.min.js')}" type="text/javascript"></s
<script src="${URLUtils.staticURL('/lib/jquery/jquery.validate.min.js')}" type="text/javascript"></sc
<script src="${URLUtils.staticURL('/lib/jquery/jquery.zoom.min.js')}" ></script>
<script type="text/javascript"><isinclude template="resources/appresources"/></script>
<script type="text/javascript"><isinclude url="${URLUtils.url('Resources-LoadTransient')}"/></script>
<script>var consent = ${session.custom.consentTracking};</script>
<isinclude template="components/ordergroove"/>
<script src="${URLUtils.staticURL('/js/app.js')}"></script>
<isif condition="${!('pageContext' in this) || empty(pageContext)}">
    <isscript>pageContext = new Object();</isscript>
</isif>
<script>pageContext = <isprint value="${JSON.stringify(pageContext)}" encoding="off"/>;</script>
<script>
var meta = "${pdict.CurrentPageMetaData.description}";
var keywords = "${pdict.CurrentPageMetaData.keywords}";
</script>
```

Modify **displayliproduct.isml** and add the following markup where desired:

```
<iscomment>Ordergroove Offer</iscomment>
<isif
condition="${!empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable'))
&& dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable')==true}">
    <isif
condition="${empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer'
)) || dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer') == true}">
        <isif condition="${!empty(pdict.isConfirmStage) && pdict.isConfirmStage == true}">
          <div class="og-offer" data-og-module="conf" data-og-
product="${productLineItem.getProductID()}"></div>
        <iselse/>
            <div class="og-offer" data-og-module="sc" data-og-
product="${productLineItem.getProductID()}"></div>
        </isif>
    <iselse/>
        <og-offer product="${productLineItem.getProductID()}"></og-offer>
    </isif>
</isif>
```

```
<div class="sku">
    <span class="label">${Resource.msg('global.itemno','locale',null)} </span>
    <span class="value"><isprint value="${productLineItem.productID}"/></span>
</div>

<iscomment>variations</iscomment>
<isdisplayvariationvalues product="${productLineItem.product}"/>
```

```
<iscomment>Ordergroove Offer</iscomment>
<isif condition="${!empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable')) && dw.system.S
    <isif condition="${empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer')) || dw.
        <isif condition="${!empty(pdict.isConfirmStage) && pdict.isConfirmStage == true}">
           <div class="og-offer" data-og-module="conf" data-og-product="${productLineItem.getProductID()}"></div>
        <iselse/>
            <div class="og-offer" data-og-module="sc" data-og-product="${productLineItem.getProductID()}"></div>
        </isif>
    <iselse/>
        <og-offer product="${productLineItem.getProductID()}"></og-offer>
    </isif>
</isif>
```

```
<iscomment>product list info</iscomment>
<isif condition="${productLineItem.productListItem != null}">
    <span class="item-links">
```

Modify **productcontent.isml** and add the following markup where desired:

```
    <iscomment>Ordergroove Offer</iscomment>
    <isif
condition="${!empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable'))
&& dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable') == true}">
        <isif condition="${isQuickView}">
            <isif condition="${pdict.CurrentHttpParameterMap.source.stringValue ===
'quickview'}">
                <isif
condition="${empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer'
)) || dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer') == true}">
                    <div class="og-offer" data-og-module="qv" data-og-product="*"></div>
                <iselse/>
                    <og-offer product="${pdict.Product.getID()}"></og-offer>
                </isif>
            </isif>
        <iselse/>
            <isif
condition="${empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer'
)) || dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer') == true}">
             <div class="og-offer" data-og-module="pdp" data-og-product="*"></div>
            <iselse/>
                <og-offer product="${pdict.Product.getID()}"></og-offer>
            </isif>
        </isif>
    </isif>
```

```
<iscomment>
    variations
    ============================================================
</iscomment>

<isinclude template="product/components/variations"/>
```

```
<iscomment>Ordergroove Offer</iscomment>
<isif condition="${!empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveEnable')) && dw.syste
    <isif condition="${isQuickView}">
        <isif condition="${pdict.CurrentHttpParameterMap.source.stringValue === 'quickview'}">
            <isif condition="${empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffe
                <div class="og-offer" data-og-module="qv" data-og-product="*"></div>
            <iselse/>
                <og-offer product="${pdict.Product.getID()}"></og-offer>
            </isif>
        </isif>
    <iselse/>
        <isif condition="${empty(dw.system.Site.getCurrent().getCustomPreferenceValue('OrderGrooveLegacyOffer'))
           <div class="og-offer" data-og-module="pdp" data-og-product="*"></div>
        <iselse/>
            <og-offer product="${pdict.Product.getID()}"></og-offer>
        </isif>
    </isif>
</isif>
```

```
<iscomment>
    add to cart form
```

Modify content asset ID **account-nav-registered** and add a link for the subscription interface where desired:

```
<li><a title="Manage subscriptions" href="$httpsUrl(OrderGroove-MSI)$">My Subscriptions</a></li>
```

SFRA

The function calls are important for displaying Ordergroove offers so the JavaScript files need to compile into SFRA's main.js to successfully display. Make sure **package.json** contains the correct directory for the SFRA project by reviewing the '**paths.base**' property. Running npm should build these files successfully.

Modify **quickview.js** and add the following highlighted code to the **fillModalElement** function:

```
$('#quickViewModal').trigger('quickview:afterShow');
```

```
success: function (data) {
    var parsedHtml = parseHtml(data.renderedTemplate);

    $('.modal-body').empty();
    $('.modal-body').html(parsedHtml.body);
    $('.modal-footer').html(parsedHtml.footer);
    $('.full-pdp-link').text(data.quickViewFullDetailMsg);
    $('#quickViewModal .full-pdp-link').attr('href', data.productUrl);
    $('#quickViewModal .size-chart').attr('href', data.productUrl);
    $('#quickViewModal .modal-header .close .sr-only').text(data.closeButtonText);
    $('#quickViewModal .enter-message').text(data.enterDialogMessage);
    $('#quickViewModal').modal('show');
    $('#quickViewModal').trigger('quickview:afterShow');
    $.spinner().stop();
},
error: function () {
    $.spinner().stop();
}
```

External Interfaces

The outbound Ordergroove API requests use REST services to communicates via HTTPS POST. The inbound recurring order endpoint uses a guard that requires the request from Ordergroove to be delivered using HTTPS. The product feed communicates to Ordergroove servers via SFTP.

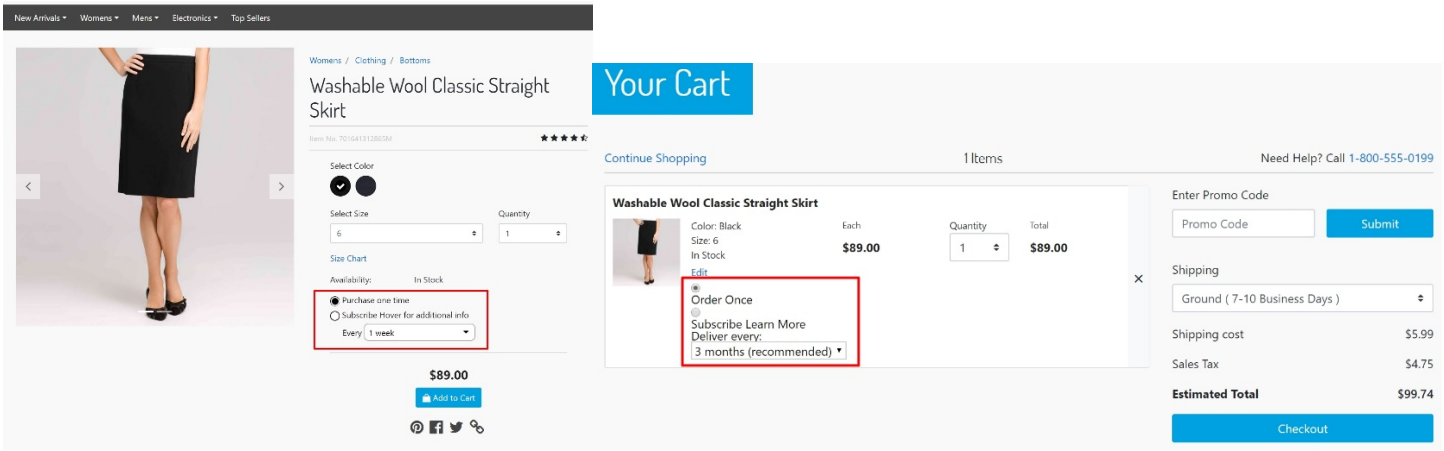
Ordergroove provides other APIs beyond the scope of this cartridge to enhance the user experience and meet the most sophisticated requirements. RPC endpoints may be provided in cases where the implementation of an action could be simplified by incorporating the business logic into a single endpoint. Please visit https://developers.ordergroove.com/ for more information.

Firewall Requirements

Most of the outbound services provided in the cartridge occur on port 80. However, the product feed places a file on the Ordergroove SFTP server using port 22 and may require a firewall change on the Commerce Cloud and Ordergroove side.

Testing

Subscription offerings are merchant specific and will be available as soon as the product feed is received and ingested by Ordergroove. Once configurations have been made by OG, the cartridge can be tested after completing the integration guide.



Operations & Maintenance

Data Storage

A custom object will be used to store data touch points in situations where failed attempts to create a subscription were made. These subscriptions will automatically be retried every 10 minutes within 24 hours. Each subscription retry will have retention set for 1 day from the time the order was placed. In the event the retry job is processed successfully, it will be purged before the 24-hour expiration. All custom objects will capture the order number as a key data point. However, only some implementations will store the Ordergroove session ID (based on a storefront cookie) or the AES encrypted credit card account number. The additional data will depend on client-specific site preferences being enabled.

Clients which do not have their own tokenizing payment vendor may elect to use Ordergroove's tokenizing solution via AES encryption and the Paymetric secure data repository (SDR).

Availability

Under highly stressed and overload conditions, Ordergroove servers may not respond immediately. Normally, these servers will recover within 10 minutes.

An email notification can be delivered to addresses in the site preference section for an initially failed subscription on certain response codes. Redundant email notifications will not be sent for retry attempts since it will be handled and reprocessed automatically. Please refer to the Ordergroove service level agreement (SLA) for more information.

If any production problems should occur, a master kill switch for the Ordergroove implementation is in place for all integration points. When disabled, the switch will also cause the job to cease from reprocessing failed subscriptions. It is important to note, any custom objects left in queue will be purged after a 24-hour expiration period. In addition, Ordergroove will not be able to place recurring orders via the Commerce Cloud endpoint when the Ordergroove integration has been disabled.

Support

Please contact customersupport@ordergroove.com for any defects or improvements.

User Guide

Roles & Responsibilities

The client will be responsible for any customizations with products (e.g. product options, bundles, variation groups, etc.) along with any SiteGenesis page tagging or additional feeds.

The client should go through an audit process with Ordergroove to ensure the validity of page tagging especially in cases where HTML elements, classes, and IDs have deviated from SiteGenesis.

Ordergroove will guide the client with a tailored experience by providing a client-specific integration guide. Ordergroove will also handle marketing related emails and styling for subscription offers or impulse upsells.

Business Manager

Many configuration options are available in the Ordergroove site preference section and contain detailed descriptions for each preference.

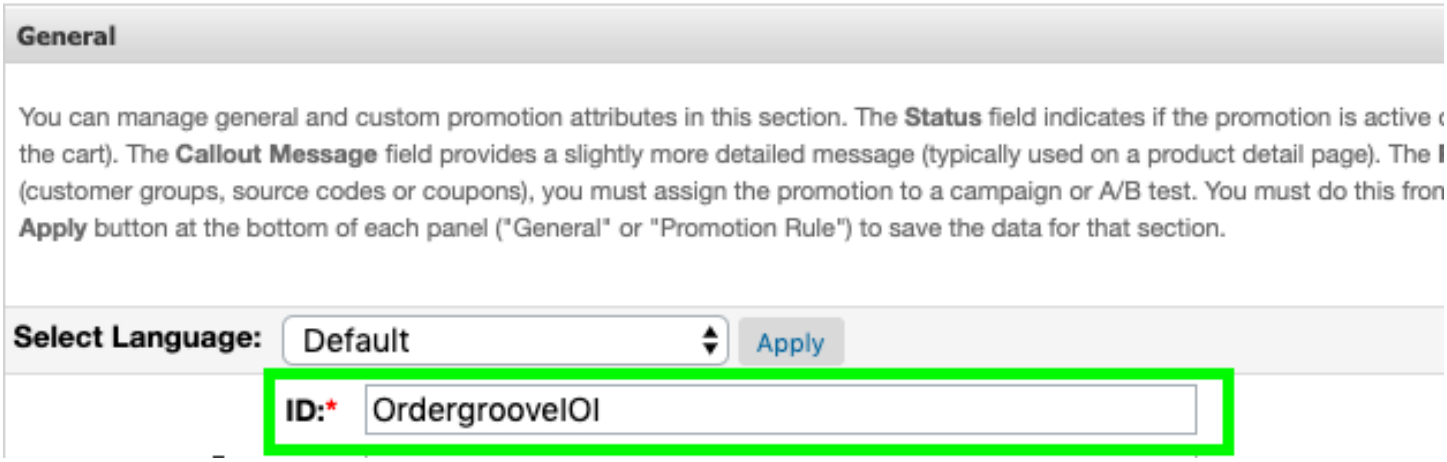
Storefront Functionality

New functionality will be evident on the product detail page (PDP), quick view on search results and product listing pages (PLP), and the cart page. The account section also contains a link to the 'my subscription interface' (MSI) on each user's dashboard.

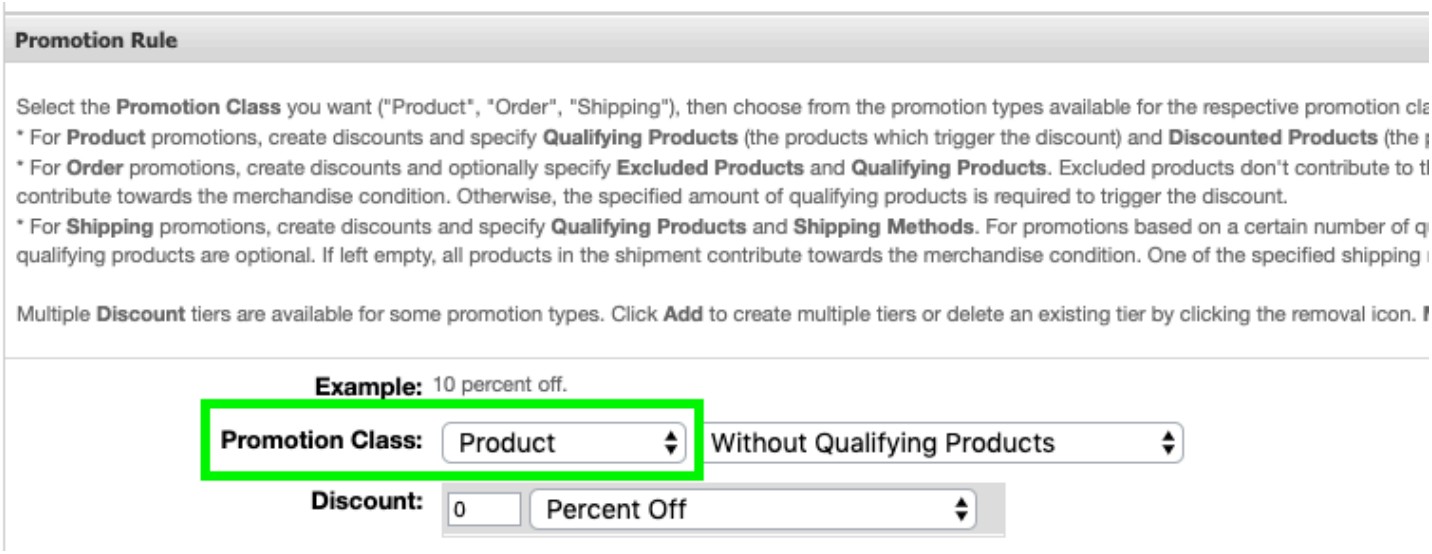
IOI Promotion

The Initial-Order-Incentive Promotion, or IOI Promotion, allows the client to offer a discount to users for the first order in a new subscription. The IOI Promotion works exactly like other promotions in Salesforce, with a couple exceptions.

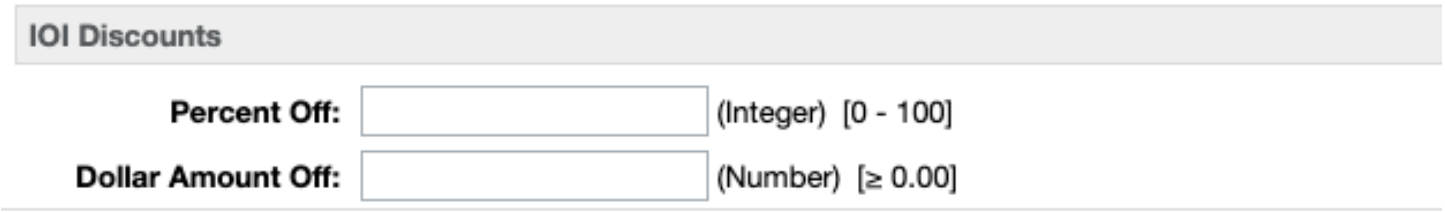
The ID of the IOI Promotion must be "OrdergrooveIOI":



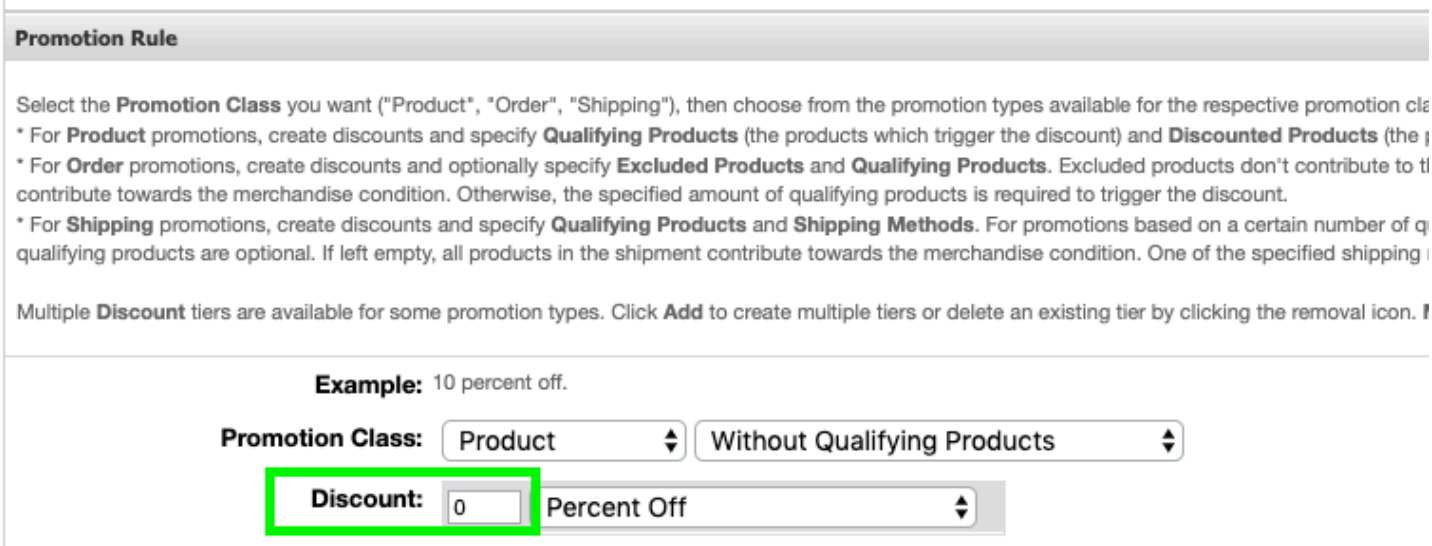
The Promotion Class of the IOI Promotion must be “Product”:



The discount associated with the IOI Promotion must be managed via the IOI Discounts section:



Since the discount of the IOI Promotion is managed via the IOI Discounts section, the value of the Discount field used for other promotions must be 0:



Known Issues

When at least one subscription is selected in the cart, only a registered user may proceed through checkout and the payment method must be a credit card.

Split or multiple shipping address and split credit card payment are not available for use in this cartridge.

Release History

Version	Date	Changes
19.2.0	December 09, 2019	Added new offers & SG tagging support
19.1.0	August 02, 2019	Added new functionality
18.1.0	July 10, 2018	Supporting SFRA
17.1.0	November 13, 2017	New release
15.1.0	November 16, 2015	Initial release