**Subscribe Pro**

***Version 20.1.0***



**Table of Contents**

[1. Summary 1-3](#_3znysh7)

[2. Component Overview 2-4](#_2et92p0)

[2.1 Functional Overview 2-4](#_tyjcwt)

[2.2 Use Cases 2-4](#_3dy6vkm)

[2.3 Limitations, Constraints 2-4](#_4d34og8)

[2.4 Compatibility 2-4](#_2s8eyo1)

[2.5 Privacy, Payment 2-4](#_17dp8vu)

[3. Implementation Guide 3-5](#_26in1rg)

[3.1 Setup 3-5](#_lnxbz9)

[3.2 Configuration 3-5](#_35nkun2)

[3.3 Custom Code 3-5](#_1ksv4uv)

[3.4 External Interfaces 3-5](#_44sinio)

[3.5 Testing 3-5](#_2jxsxqh)

[4. Operations, Maintenance 4-6](#_z337ya)

[4.1 Data Storage 4-6](#_3j2qqm3)

[4.2 Availability 4-6](#_1y810tw)

[4.3 Support 4-6](#_4i7ojhp)

[5. User Guide 5-7](#_1ci93xb)

[5.1 Roles, Responsibilities 5-7](#_3whwml4)

[5.2 Business Manager 5-7](#_2bn6wsx)

[5.3 Storefront Functionality 5-7](#_3as4poj)

[6. Known Issues 6-8](#_1pxezwc)

[7. Release History 7-9](#_2p2csry)

# Summary

This document provides technical instructions for installing the Commerce Cloud Integration LINK integration that lets you integrate Subscribe Pro product subscriptions with a Commerce Cloud storefront.

This cartridge integrates Salesforce Commerce Cloud with the Subscribe Pro Subscription Commerce Platform.

This integration requires deployment of a new cartridge and modification of Storefront code.

A subscription to the Subscribe Pro solution is required to use this cartridge.

Sandbox (testing) environments are available.

Provided “int\_subscribe\_pro” cartridge is implemented using controllers architecture.

This integration is based on the Site Genesis demo store provided by Commerce Cloud.

# Component Overview

## Functional Overview

Subscribe Pro enables brands and retailers to add subscription commerce, auto-ship, continuity, clubs, boxes and recurring billing features to their existing eCommerce site, fully integrated with their existing eCommerce technology. Subscribe Pro supports supports recurring payments via credit card and eCheck, with pre-built integrations for all common payment gateways. Contact us today to find out how our solution can help you implement a product subscription program.

## Use Cases

**Scenario**: A user visits a site that has subscription products available to purchase and would like to purchase those Products one time in order to try the product out before signing up for a recurring subscription.

**Result**: User will be presented with various options to subscribe to the subscription product with one of the options being a one time purchase. The user should be able to add this product to cart and checkout as if the product is a normal product in the catalog. Messaging should be present throughout checkout to inform the user that this is a one time purchase.

**Scenario**: A user visits a site that has subscription products available to purchase and would like to purchase a subscription product on a recurring interval automatically.

**Result**: User will be presented with various options to subscribe to the subscription product with one of the options being a subscription. The user should be able to add this product to cart and checkout as if the product is a normal product in the catalog. Messaging should be present throughout checkout to inform the user that this is a subscription purchase and display the interval they have chosen for repeat orders.

**Scenario:** A user has added some subscription products to their Basket and would like to adjust some of the values.

**Result:** User will be presented with options to change their subscription settings on the Cart Page. The user will also be able to click the “edit” link, on the subscription product, and change the subscription options in the quick view dialog.

**Scenario:** User has purchased a subscription product. The next time the ProcessSubproSubscriptions is run the necessary details about the customer and the products purchased will be transmitted to Subscribe Pro and the order will be marked as having been processed.

## Limitations, Constraints

* Merchants must be enrolled at Subscribe Pro
* Subscription products must be configured at Subscribe Pro and Sales Force Commerce Cloud
* A Credit Card will be required for subscription products
* Customers must be logged into the storefront in order to purchase subscription products
* Customer must save their Credit Card to their account in order to purchase subscription products

## Compatibility

Compatible OCAPI version: 19.8

Compatible Site Genesis version: 104.1.3

## Privacy, Payment

Customer profile data and payment information will be transferred and stored at Subscribe Pro.

This information includes:

* Name
* Email
* Address Details
* Saved Payment Instrument ID

# Implementation Guide

## Setup

Subscribe Pro LINK integration uses the following cartridges:

* *int\_subscribe\_pro* – this cartridge contains all controllers, templates and scripts to create and display Subscribe Pro subscriptions. Also LINK-subscribe-pro contains the Business Manager XML imports such as the System Object Types customizations, services configurations and configuration of job for processing orders with pending subscription request.
* *app\_storefront\_controllers*,
* *app\_storefront\_core* – standard storefront cartridges with Subscribe Pro integrated. They can be used as an example for the integration and testing the cartridge.

Complete the following steps to install the cartridge.

### Initial Install

See the Commerce Cloud documentation for adding a cartridge to your storefront: <https://documentation.demandware.com/DOC1/index.jsp?topic=%2Fcom.demandware.dochelp%2FSiteDevelopment%2FAddinganexistingcartridgetoyourstorefront.html&cp=0_4_0_0_3>

**Note.** You must also register the cartridge with the Business Manager site in order to run the Subscribe Pro jobs. Navigate to *Administration > Sites > Manage Sites > Business Manager > Settings tab*.

#### Update Site's cartridge path

1. Navigate to *Administration > Sites > Manage Sites > [SiteName] > Settings tab*.
2. Add “:int\_subscribe\_pro” to the effective cartridge path.
3. Click “Apply”.

#### Install Services

1. Navigate to *Administration > Operations > Import & Export > Services*
   1. Import the profile and credentials found in the following XML
      1. *LINK-subscribe-pro/Metadata/services/services-shared.xml*
   2. Import the service definitions found in the following XML
      1. *LINK-subscribe-pro/Metadata/services/services.xml*

#### Import Meta Data

1. Navigate to *Administration > Site Development > Import & Export > Meta data*.
2. Import the System Object custom attributes found in the following XML
   1. *LINK-subscribe-pro/Metadata/meta/system-objecttype-extensions.xml*

After import, following system definition attributes should appear in the system:

#### System object definitions

|  |  |  |  |
| --- | --- | --- | --- |
| ***CustomerAddress*** | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproAddressID | Subscribe Pro Address ID | String | This attribute will contain the address id that is returned from Subscription Pro upon creating this address in the Subscribe Pro system. |
| ***CustomerPaymentInstrument*** | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproPaymentProfileID | Subscribe Pro Payment Profile ID | String | This attribute will contain the payment profile id that is returned from Subscription Pro upon creating payment profile record in the Subscribe Pro system. |
| subproCCPrefix | Credit Card Prefix | String | This attribute will contain the first 6 digits of the customers Credit Card, to be sent to Subscribe Pro. |
| ***Order / Basket*** *Basket duplicates the order attributes so that they will automatically be passed from one to another.* | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproContainsSubscriptions | Contains Subscriptions | Boolean | This attribute will be used to quickly assess if an order has subscriptions contained within it |
| subproSubscriptionsToBeProcessed | Subscriptions to be processed | Boolean | This attribute will note whether or not this order has subscriptions that need to be processed. This value will be set to true if any products in the basket have subscriptions and will be set to false after those subscriptions have been processed |
| ***OrderAddress*** | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproAddressID | Subscribe Pro Address ID | String | This attribute will contain the address id that is returned from Subscription Pro upon creating this address in the Subscribe Pro system. |
| customerAddressID | SFCC Customer Address ID | String | This attribute will contain the SFCC customer address id. This information may be useful when a customer is updating subscription options after an order has been placed. |
| ***Product*** | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproSubscriptionEnabled | Subscription Enabled | Boolean | This will decide whether or not the PDP should attempt to show subscription information |
| ***ProductLineItem*** | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproSubscriptionOptionMode | Subscription Option Mode | String | This attribute will contain a value for whether or not if this product is subscription or a one time purchase. Possible values are:  onetime\_purchase  subscription |
| subproSubscriptionSelectedOptionMode | Subscription Selected Option Mode | String | This attribute will contain subscription mode which was chosen on this ProductLineItem |
| subproSubscriptionInterval | Subscription Refill Interval | String | This attribute will contain the user selected time interval to refresh a subscription product |
| subproSubscriptionAvailableIntervals | Subscription Available Intervals | String | This attribute will contain a string with names of available subscription refill intervals, separated by comma |
| subproSubscriptionCreated | Subscription Created | Boolean |  |
| subproSubscriptionDateCreated | Subscription Date Created | Date |  |
| subproSubscriptionID | Subscription ID | String | This will be populated when the order processing occurs and the actual subscription is created. |
| subproSubscriptionDiscount | Subscription Discount | Double | This attribute will contain subscription discount offered on this product, as either a percentage or a fixed price, depending on the value of the subproSubscriptionIsDiscountPercentage property |
| subproSubscriptionIsDiscountPercentage | Subscription Is Discount Percentage | Boolean | This attribute will contain true if the discount on the product specified as a percentage or false if it is a fixed price amount. |
| ***Profile*** | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproCustomerID | Subscribe Pro Customer ID | String | This attribute will contain the customer id that is returned from Subscription Pro upon creating this customers records in the Subscribe Pro system. |
| **Site preferences**  *Please configure the following site preferences:* | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproCheckoutLoginMsg | Subpro Checkout Login Message | String | This attribute will contain a message that informs customer that he/she needs to be logged in as a registered customer to proceed checkout with SubPro Subscription products |
| subproCreditCardRequirementMsg | Subpro Credit Card Requirement Message | String | This attribute will contain an error message noting that a credit card is required for subscriptions |
| subproOrderProcessingErrorMail | Subscribe Pro Order Processing Error Mail | String | An email address that order subscriptions processing failures will be sent to |
| subproWidgetScriptUrl | Subscribe Pro Widget Script Url | String | URL to Subscribe Pro Widget Script |
| subproApiBaseUrl | Subscribe Pro API Base Url | String | Subscribe Pro API Base URL |
| subproAPICredSuffix | Suffix to append to the Subscribe Pro Web Service Credentials | String | Suffix to append to the Subscribe Pro Web Service Credentials |
| **Payment Card** | | | |
| **ID** | **Name** | **Type** | **Description** |
| subproCardType | SubPro Card Type | String | Credit Card Type that matches the Credit Card Types at Subscribe Pro |

#### Import ProcessSubproSubscriptions job

1. Navigate to *Administration > Operations > Import & Export > Job Schedules*.
2. Import the System Object custom attributes found in the following XML
   1. *LINK-subscribe-pro/Metadata/jobs/process-orders-job.xml*

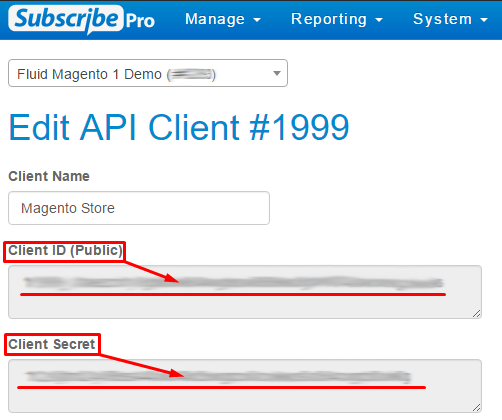
## 

## 

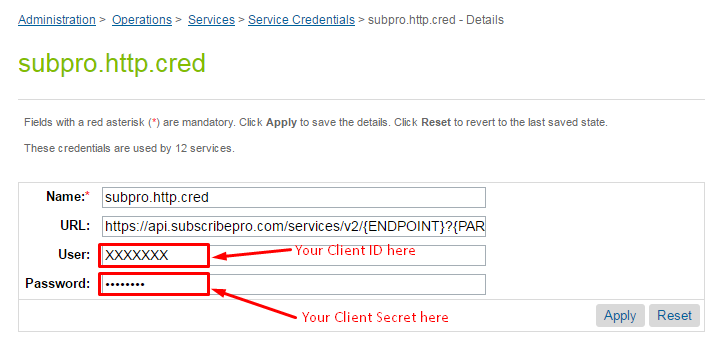
## Configuration

### Update Web Service Credentials

1. Get your Client ID and Client Secret in Subscribe Pro BM:  
   <https://platform.subscribepro.com/system/client>



1. Navigate to *Administration > Operations > Services > Service Credentials > subpro.http.cred - Details*
2. Use Client ID as User and Client Secret as Password:



1. Navigate to *Administration > Operations > Services > Service Credentials > subpro.http.cred.oauth - Details*
2. Use Client ID as User and Client Secret as Password (the same as above).

### 

### 

### Setup Schedule Job to process orders with pending subscriptions

The job will process all orders that have any subscriptions that need to be processed.

Configure job parameters:

1. Navigate to *Administration > Operations > Job Schedules > ProcessSubproSubscriptions* job.
2. On tab *Schedule and History* set run interval (default is every 15 minutes).
3. Open *Step Configurator* tab, choose *ProcessSubproSubscriptions-Process* workflow.
4. Set (***in hours***) custom parameter *ordersProcessInterval* to define what the start time for the search placed orders should be.
5. Click “Assign” button.

### Configure Subscription Products

To configure products which will have a subscription you’ll need to make changes to SFCC Business Manager as well as to Subscribe Pro Business Manager.

There are different type of products that can be configured:

* Standard Product
* Variation Product
* Variation Masters
* Product Bundles
* Product Sets
* Option Product

Below described action you need to perform to configure each type of product.

#### Standard Product

Standard Product is a product that is sold and displayed alone and does not have variations, such as different sizes or colors. If you include a standard product in a product set or product bundle, it is considered a set product or bundled product, and not a standard product.

##### Configuring Product on the Commerce Cloud Platform

The PDP page will take the Commerce Cloud Product ID and make an API request to the Subscribe Pro Products API Endpoint using the Commerce Cloud Product ID as the "sku" parameter.

The following image shows the Product ID that should be configured as a SKU in the Subscribe Pro Platform.

A screenshot of a cell phone

Description automatically generated

This API call will only be attempted if the Commerce Cloud Product has the: **Subscription Enabled** custom attribute set to: **true**.

The following image shows this setting in the following Business Manager Module:

*Merchant Tools > Products and Catalogs > Products*

A screenshot of a social media post

Description automatically generated

##### Configuring Product on the Subscribe Pro Platform

In addition to configuring the product on Commerce Cloud, the product will also need to be configured on the Subscribe Pro Platform.

<https://platform.subscribepro.com/login>

Subscription products can be managed on the platform at: *Manage > Subscription Products*

Within this module a new subscription product should be created that has a SKU that matches the Commerce Cloud Product ID.

The following is an example of a configured product that will match the product provided above for Commerce Cloud:

A screenshot of a cell phone

Description automatically generated

#### Variation Product

Variation Product is a product that is a specific variation. For example, if a Vegas brand t-shirt is represented by a variation master, then a product variation is a size 10, blue Vegas brand t-shirt.

Variation products associated with the same variation master share most of the attributes defined for the product master, but have their own product IDs and images.

Variation Products should be configured the same way as a “Standard Product”. See instructions above.

#### Variation Masters

Variation Masters is a product that represents all the variations for a particular product. For example, a specific style of t-shirt.

A variation master cannot be purchased directly, only product variations with specific variation information can be purchased. For example, if a master has color and size variation attributes, you must specify the color and size of a specific variant to purchase it.

If you have more than three variation attributes, you might want to consider using options.

To enable subscriptions for variation master products:

1. A Subscribe Pro Subscription Product will need to be created for each of the Variants contained within the Master Variant.
2. The master variant will need to have the "Subscriptions Enabled" custom attribute set to "yes".
   1. Any attributes set on the Master Product will flow down to all of the variant products, unless the variant product has overridden that particular attribute.

**Example:**

***Master Product (in Commerce Cloud)***

|  |  |
| --- | --- |
| **ID** | **Name** |
| 25752235 | Checked Silk Tie |

***Variant Product(s) (in Commerce Cloud)***

|  |  |  |
| --- | --- | --- |
| **Parent ID** | **ID** | **Options** |
| 25752235 | 682875090845 | Color: Cobalt |
| 25752235 | 682875719029 | Color: Navy |
| 25752235 | 682875540326 | Color: Yellow |

***Subscribe Pro - Subscription Products (in Subscribe Pro Platform)***

|  |  |  |
| --- | --- | --- |
| **ID** | **SKU** | **Name** |
| 587218 | 682875090845 | Checked Silk Tie - Cobalt |
| 587219 | 682875719029 | Checked Silk Tie - Navy |
| 587220 | 682875540326 | Checked Silk Tie - Yellow |

#### Product Bundles

Commerce Cloud Digital product bundles:

* Are a separate product with its own SKU and price
* Are unaffected by the price of the individual products
* Contain specified products in specified numbers

In respect to subscription products, product bundles can be treated the same as product variants. The end user will select subscription options for the entire bundle and the single product ID for the bundle is what will be configured and transmitted to Subscribe Pro when creating subscriptions.

See the instructions how to configure a Standard Product above.

Sales Force Commerce Cloud will automatically add all of the bundled products (children) to the Basket when the product bundle is added, whether it be from the storefront or from OCAPI.

The following is an example of an OCAPI request and response using product bundle: sony-ps3-bundle:

|  |
| --- |
| **Request** |
| {  "customer\_info": {  "customer\_no": "00000001",  "email": "<EMAIL>"  },  "billing\_address": {  "first\_name": "<FIRST>",  "last\_name": "<LAST>",  "address1": "<STREET>",  "city": "<CITY>",  "state\_code": "<STATE>",  "country\_code": "<COUNTRY>"  },  "product\_items": [{  "product\_id": "sony-ps3-bundle",  "quantity": 1  }],  "shipments": [{  "gift": false,  "shipping\_method": {  "id": "001"  },  "shipping\_address": {  "first\_name": "<FIRST>",  "last\_name": "<LAST>",  "address1": "<STREET>",  "city": "<CITY>",  "state\_code": "<STATE>",  "country\_code": "<COUNTRY>"  } }]  } |

|  |
| --- |
| **Response** |
| {  "\_v":"17.3",  "\_type":"basket",  "\_resource\_state":"22e95720919f1eaa894bda606664b5a61a9575e0a942a1452bbfdb7062da0f69",  "\_flash":[  {  "\_type":"flash",  "type":"PaymentMethodRequired",  "message":"No payment method ID was specified. Please provide a valid payment method ID.",  "path":"$.payment\_instruments[0].payment\_method\_id"  }  ],  "adjusted\_merchandize\_total\_tax":22.45,  "adjusted\_shipping\_total\_tax":0.5,  "agent\_basket":true,  "basket\_id":"16f3006604e77fff7105caf63a",  "billing\_address":{  "\_type":"order\_address",  "address1":"<STREET>",  "city":"<CITY>",  "country\_code":"<STATE>",  "first\_name":"<FIRST>",  "full\_name":"<FULL>",  "id":"b600c40aec47165cacc48c7ad2",  "last\_name":"<LAST>",  "state\_code":"<STATE>"  },  "channel\_type":"callcenter",  "creation\_date":"2017-04-19T15:26:23.314Z",  "currency":"USD",  "customer\_info":{  "\_type":"customer\_info",  "customer\_id":"abKT2hbWsyyV8alhk6VJg9Ow0E",  "customer\_no":"00000001",  "email":"<EMAIL>"  },  "last\_modified":"2017-04-19T15:26:23.360Z",  "merchandize\_total\_tax":22.45,  "notes":{  "\_type":"simple\_link",  "link":"https://subscribepro01-tech-prtnr-na02-dw.demandware.net/s/SiteGenesis/dw/shop/v17\_3/baskets/16f3006604e77fff7105caf63a/notes"  },  "order\_total":481.94,  "product\_items":[  {  "\_type":"product\_item",  "adjusted\_tax":22.45,  "base\_price":449,  "bonus\_product\_line\_item":false,  "bundled\_product\_items":[  {  "\_type":"product\_item",  "adjusted\_tax":null,  "base\_price":null,  "bonus\_product\_line\_item":false,  "item\_id":"94921b91f537474e71376abaf2",  "item\_text":"Sony Playstation 3 Game Console",  "price":null,  "price\_after\_item\_discount":null,  "price\_after\_order\_discount":0,  "product\_id":"sony-ps3-console",  "product\_name":"Sony Playstation 3 Game Console",  "quantity":1,  "shipment\_id":"me",  "tax":null,  "tax\_basis":null,  "tax\_class\_id":"standard",  "tax\_rate":0.05  },  {  "\_type":"product\_item",  "adjusted\_tax":null,  "base\_price":null,  "bonus\_product\_line\_item":false,  "item\_id":"6356afd70d4a120da36a9d3baf",  "item\_text":"Nascar 09 (for Sony PS3)",  "price":null,  "price\_after\_item\_discount":null,  "price\_after\_order\_discount":0,  "product\_id":"easports-nascar-09-ps3",  "product\_name":"Nascar 09 (for Sony PS3)",  "quantity":1,  "shipment\_id":"me",  "tax":null,  "tax\_basis":null,  "tax\_class\_id":"standard",  "tax\_rate":0.05  },  {  "\_type":"product\_item",  "adjusted\_tax":null,  "base\_price":null,  "bonus\_product\_line\_item":false,  "item\_id":"8b606524f0c7a7325cc4e4e23a",  "item\_text":"Monopoly Here and Now: The World Edition (for Sony PS3)",  "price":null,  "price\_after\_item\_discount":null,  "price\_after\_order\_discount":0,  "product\_id":"easports-monopoly-ps3",  "product\_name":"Monopoly Here and Now: The World Edition (for Sony PS3)",  "quantity":1,  "shipment\_id":"me",  "tax":null,  "tax\_basis":null,  "tax\_class\_id":"standard",  "tax\_rate":0.05  },  {  "\_type":"product\_item",  "adjusted\_tax":null,  "base\_price":null,  "bonus\_product\_line\_item":false,  "item\_id":"0798adadda95b0a5c4d282980f",  "item\_text":"Eternal Sonata (for Sony PS3)",  "price":null,  "price\_after\_item\_discount":null,  "price\_after\_order\_discount":0,  "product\_id":"namco-eternal-sonata-ps3",  "product\_name":"Eternal Sonata (for Sony PS3)",  "quantity":1,  "shipment\_id":"me",  "tax":null,  "tax\_basis":null,  "tax\_class\_id":"standard",  "tax\_rate":0.05  },  {  "\_type":"product\_item",  "adjusted\_tax":null,  "base\_price":null,  "bonus\_product\_line\_item":false,  "item\_id":"b0bfeff3a8214ea603c570d73b",  "item\_text":"Warhawk (for Sony PS3)",  "price":null,  "price\_after\_item\_discount":null,  "price\_after\_order\_discount":0,  "product\_id":"sony-warhawk-ps3",  "product\_name":"Warhawk (for Sony PS3)",  "quantity":1,  "shipment\_id":"me",  "tax":null,  "tax\_basis":null,  "tax\_class\_id":"standard",  "tax\_rate":0.05  }  ],  "item\_id":"8ddceb64e1d3856008a753122a",  "item\_text":"Playstation 3 Bundle",  "option\_items":[  {  "\_type":"option\_item",  "adjusted\_tax":0,  "base\_price":0,  "bonus\_product\_line\_item":false,  "item\_id":"392333b488e166df0b46927548",  "item\_text":"Extended Warranty: None",  "option\_id":"consoleWarranty",  "option\_value\_id":"000",  "price":0,  "price\_after\_item\_discount":0,  "price\_after\_order\_discount":0,  "product\_id":"000",  "product\_name":"Extended Warranty: None",  "quantity":1,  "shipment\_id":"me",  "tax":0,  "tax\_basis":0,  "tax\_class\_id":"standard",  "tax\_rate":0.05  }  ],  "price":449,  "price\_after\_item\_discount":449,  "price\_after\_order\_discount":449,  "product\_id":"sony-ps3-bundle",  "product\_name":"Playstation 3 Bundle",  "quantity":1,  "shipment\_id":"me",  "tax":22.45,  "tax\_basis":449,  "tax\_class\_id":"standard",  "tax\_rate":0.05  }  ],  "product\_sub\_total":449,  "product\_total":449,  "shipments":[  {  "\_type":"shipment",  "adjusted\_merchandize\_total\_tax":22.45,  "adjusted\_shipping\_total\_tax":0.5,  "gift":false,  "merchandize\_total\_tax":22.45,  "product\_sub\_total":449,  "product\_total":449,  "shipment\_id":"me",  "shipment\_total":481.94,  "shipping\_address":{  "\_type":"order\_address",  "address1":"<STREET>",  "city":"<CITY>",  "country\_code":"<COUNTRY>",  "first\_name":"<FIRST>",  "full\_name":"<FULL>",  "id":"29410763e35c40915b515a0cdd",  "last\_name":"<LAST>",  "state\_code":"<STATE>"  },  "shipping\_method":{  "\_type":"shipping\_method",  "description":"Order received within 7-10 business days",  "id":"001",  "name":"Ground",  "price":9.99  },  "shipping\_status":"not\_shipped",  "shipping\_total":9.99,  "shipping\_total\_tax":0.5,  "tax\_total":22.95  }  ],  "shipping\_items":[  {  "\_type":"shipping\_item",  "adjusted\_tax":0.5,  "base\_price":9.99,  "item\_id":"1ad26e6d3776f228e7d56228d7",  "item\_text":"Shipping",  "price":9.99,  "price\_after\_item\_discount":9.99,  "shipment\_id":"me",  "tax":0.5,  "tax\_basis":9.99,  "tax\_class\_id":"standard",  "tax\_rate":0.05  }  ],  "shipping\_total":9.99,  "shipping\_total\_tax":0.5,  "taxation":"net",  "tax\_total":22.95  } |

#### Product Sets

Product Set is a product that includes several products that are displayed together and can be purchased together or separately. For example, an accessories kit that includes a hairbrush, comb, and mirror, each of which can also be bought separately. Another common example is an outfit, with a jacket, shirt, and pants, which can be purchased together or each item can be purchased separately.

Unlike Product Bundles, Products Set Products are added to the cart individually. The end consumer has the option to add each product individually or to add all of the products at once.

When the consumer adds all of the products at once, the form fields for each individual product are preserved, for example quantity and subscription options.

Products within a product set can be configured the same as a standard product and the user will have the ability to configure the subscription options and interval for each individual product. See the instructions for configuring a Standard Product above.

#### Option Product

Commerce Cloud Digital, product options enable you to sell configurable products that have optional accessories, upgrades, or additional services. Options are always purchased with a product and cannot be purchased separately. They have their own price and display name, but do not have their own thumbnail images. They cannot be searched by the customer, but are usually visible on the product detail page.

Products that are configured with Product Options should be configured the same as a normal subscription product in Commerce Cloud and Subscribe Pro. The only difference with these products is that the Order Processing Script, that creates the subscription at Subscribe Pro, will also include any product options, selected for the product line item, in the platform\_specific\_fields attribute.

|  |
| --- |
| **Subscribe Pro Subscription Request** |
| {  "subscription":{  "customer\_id":"348328",  "payment\_profile\_id":"410028",  "requires\_shipping":true,  "shipping\_address\_id":455367,  "product\_sku":"canon-powershot-e1",  "qty":1,  "use\_fixed\_price":false,  "interval":"Month",  "next\_order\_date":"2017-04-19T21:33:56.954Z",  "first\_order\_already\_created":true,  "send\_customer\_notification\_email":true,  "platform\_specific\_fields":{  "sfcc":{  "product\_options":[  {  "id":"digitalCameraWarranty",  "value":"002"  }  ]  }  }  }  } |

These product options will need to be added to the Basket when recurring orders are placed through OCAPI. The following is an example of a Basket object, with a product option, that could be sent back to Commerce Cloud via OCAPI:

|  |
| --- |
| **OCAPI Basket Object** |
| {  "customer\_info": {  "customer\_no": "00000001",  "email": "<EMAIL>"  },  "billing\_address": {  "first\_name": "<FIRST>",  "last\_name": "<LAST>",  "address1": "<STREET>",  "city": "<CITY>",  "state\_code": "<STATE>",  "country\_code": "<COUNTRY>"  },  "product\_items": [{  "product\_id": "samsung-hl72a650",  "quantity": 1,  "option\_items": [  {  "option\_id": "tvWarranty",  "option\_value\_id": "002"  }  ]  }],  "shipments": [{  "gift": false,  "shipping\_method": {  "id": "001"  },  "shipping\_address": {  "first\_name": "<FIRST>",  "last\_name": "<LAST>",  "address1": "<STREET>",  "city": "<CITY>",  "state\_code": "<STATE>",  "country\_code": "<COUNTRY>"  }  }]  } |

### Configure OCAPI client

Subscribe Pro will make use of the "Shop" Open Commerce API (OCAPI) in order to place recurring orders for subscription products. This will be accomplished server to server using a Business Manager login.

In Business Manager, under Administration > Site Development > Open Commerce API Settings, ensure that the resources and methods are configured to support the example configuration in the **Documentation/okapi-example-configuration.json** file.

### Configure Payments

Subscribe Pro will make use of OCAPI to place orders on behalf of the customer. This is one of the reason's that a customer must be logged in and have a stored payment instrument in order to place an order with subscriptions.

#### Credit Cards

A Payment token and the Payment Instrument ID will be transmitted to Subscribe Pro as part of the Subscription creation process.

Credit Card tokenization will need to be implemented by a tokenization provider when this cartridge is implemented, on an actual site. This will be accomplished by making use of the: setCreditCardToken method in the: PaymentInstrument object.

See SFCC documentation:

<https://documentation.demandware.com/DOC1/topic/com.demandware.dochelp/DWAPI/scriptapi/html/api/class_dw_order_PaymentInstrument.html>

In order to provide an example of this, on the base site, some code has been added the COBilling controller to generate a random token for credit cards:

|  |
| --- |
| /\*\*  \* This needs to be replaced with a valid PSP Tokenizer before going to production  \*/  let random = new dw.crypto.SecureRandom();  let bytes = random.nextBytes(32);  let tokenString = newCreditCard.getUUID() + new Date().getTime() + bytes.toString();  let messageDigest = new dw.crypto.MessageDigest(dw.crypto.MessageDigest.DIGEST\_SHA\_512);  let hash = dw.crypto.Encoding.toBase64(messageDigest.digestBytes(new dw.util.Bytes(tokenString)));    newCreditCard.setCreditCardToken(hash); |

#### Subscribe Pro - Payment Profile

Payment profile should be used to pay for subscriptions.

The use of this profile could be divided into two steps:

* creating subscription
* recurring subscription charging

##### Creating subscription

As part of the order processing script, we send the relevant Payment Data to the Subscribe Pro using the */vault/paymentprofile/external-vault end-point*.

The following is the relevant snippet of code in the Order Processing Logic:

***/int\_subscribe\_pro/cartridge/scripts/subpro/jobs/ProcessOrderSubscriptions.js***

|  |
| --- |
| /\*\*  \* Apple Pay otherwise assume Credit Card  \*/  if (paymentInstrument.getPaymentMethod() === "DW\_APPLE\_PAY") {  let transactionID = paymentInstrument.getPaymentTransaction().getTransactionID();  let response = SubscribeProLib.getPaymentProfile(null, transactionID);      /\*\*  \* If there was a problem creating the payment profile, error out  \*/  if (response.error) {  logError(response, 'getPaymentProfile');  continue;  } else {  paymentProfileID = response.result.payment\_profiles.pop().id;  }  } else {  paymentProfileID = ('subproPaymentProfileID' in customerPaymentInstrument.custom) ? customerPaymentInstrument.custom.subproPaymentProfileID : false;    /\*\*  \* If Payment Profile already exists,  \* Call service to verify that it still exists at Subscribe Pro  \*/  if (paymentProfileID) {    let response = SubscribeProLib.getPaymentProfile(paymentProfileID);    /\*\*  \* Payment Profile not found, create new Payment Profile record  \* Otherwise create the payment profile  \*/  if (response.error && response.result.code === 404) {  paymentProfileID = createSubproPaymentProfile(customerProfile, customerPaymentInstrument, order.billingAddress);  /\*\*  \* Some other error occurred, error out  \*/  } else if (response.error) {  logError(response, 'getPaymentProfile');  continue;  }  } else {  paymentProfileID = createSubproPaymentProfile(customerProfile, customerPaymentInstrument, order.billingAddress);  }  } |

##### Recurring subscription charging

To charge a regular fee we should create OCAPI order using payment profile information saved earlier.

Then using this data SubPro side will create order for paying for subscription via OCAPI.

1. Request POST /dw/shop/v17\_3/baskets to create basket for order.
2. Use POST /dw/shop/v17\_3/baskets/{basket\_id}/payment\_instruments to add payment instruments to basket. This requires fields payment\_instrument\_id and amount.
3. Request POST /dw/shop/v17\_3/orders with basket\_id attribute to create order.

## Custom Code

**Templates**

* addressinclude.isml
* cart.isml
* checkoutlogin.isml
* displayliproduct.isml
* minilineitems.isml
* multishippingshipments.isml
* orderdetails.isml
* orderdetailsemail.isml
* paymentinstrumentlist.isml
* paymentmethods.isml
* productcontent.isml
* producttopcontentPS.isml
* productsetproduct.isml
* pt\_productdetails\_UI.isml
* pt\_productsearchresult.isml
* summary.isml
* creditcardjson.isml

**Controllers**

* Account.js
* COBilling.js
* COCustomer.js
* COSummary.js
* Address.js
* PaymentInstruments.js

**JS**

* js/subscriptionOptions.js

**Scripts**

* CartModel.js
* cart.js
* variant.js
* GetProductDetailUrl.ds
* billing.js

**Forms**

* creditcard.xml

**Resource**

* Resource.ds
* account.properties

The code changes below detail the customizations that must be made to the SiteGenesis core and controllers cartridges to implement Subscribe Pro.

### subscriptionOptions.js Script

The cartridge/js/subscriptionOptions.js file in the int\_subscribe\_pro cartridge should be copied into the app\_storefront\_core/cartridge/js directory.

### Resource

***Modify account.properties***

Locate properties file */app\_storefront\_core/cartridge/templates/resource/account.properties*.

Add the following line:

|  |
| --- |
| account.subpro.mysubscriptions=My subscriptions |

### PDP

Changes need to be done to display product subscription options (is they are available) and Subscribe Pro “Add to Order” widget.

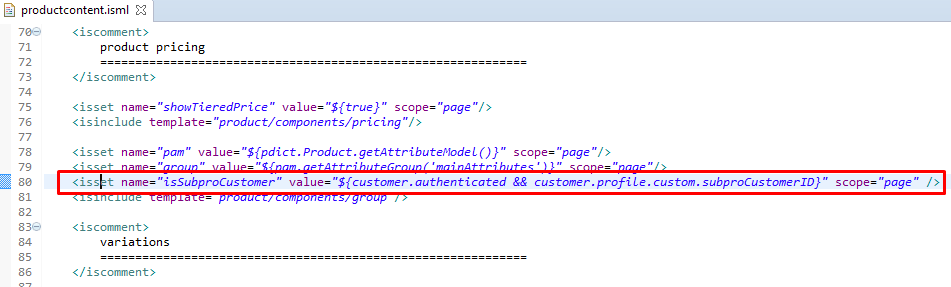
#### Modify productcontent.isml

Find template */app\_storefront\_core/cartridge/templates/default/product/productcontent.isml*.

1. Set the variable specifying if current customer is a Subscribe Pro customer:

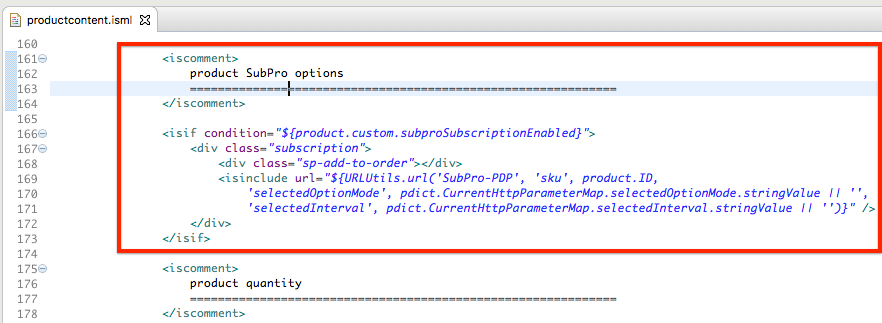
|  |
| --- |
| <isset name="isSubproCustomer" value="${customer.authenticated && customer.profile.custom.subproCustomerID}" scope="page" /> |

Here is the expected result:



1. Scroll down to add to cart form, find *“product quantity”* section and paste code below before it to render product subscription options:

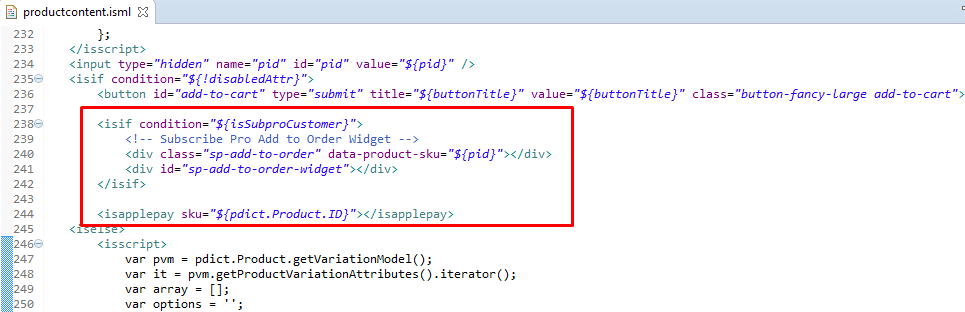
|  |
| --- |
| <iscomment>  product SubPro options  =============================================================  </iscomment>  <isif condition="${product.custom.subproSubscriptionEnabled}">  <div class="subscription">  <div class="sp-add-to-order"></div>  <isinclude url="${URLUtils.url('SubPro-PDP', 'sku', product.ID,  'selectedOptionMode', pdict.CurrentHttpParameterMap.selectedOptionMode.stringValue || '',  'selectedInterval', pdict.CurrentHttpParameterMap.selectedInterval.stringValue || '')}" />  </div>  </isif> |

Here is the expected result:  


1. Scroll down to “add-to-cart” button and insert code below after it to render Subscribe Pro “Add to Order” widget:

|  |
| --- |
| <isif condition="${isSubproCustomer}">  <!-- Subscribe Pro Add to Order Widget -->  <div class="sp-add-to-order" data-product-sku="${pid}"></div>  <div id="sp-add-to-order-widget"></div>  </isif> |

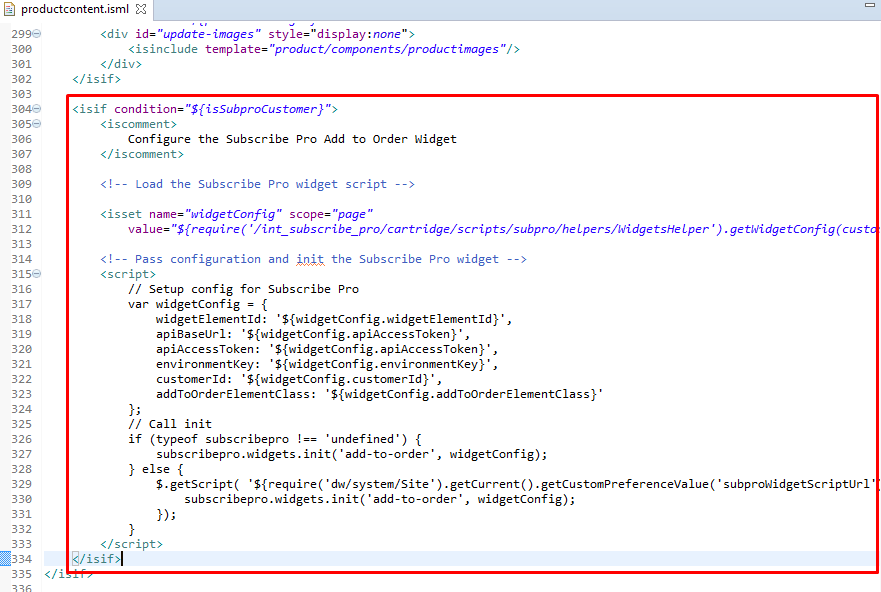
Here is the expected result:



1. Add code to configure Subscribe Pro “Add to Order” widget:

|  |
| --- |
| <isif condition="${isSubproCustomer}">  <iscomment>  Configure the Subscribe Pro Add to Order Widget  </iscomment>  <!-- Load the Subscribe Pro widget script -->  <isset name="widgetConfig" scope="page"  value="${require('/int\_subscribe\_pro/cartridge/scripts/subpro/helpers/WidgetsHelper').getWidgetConfig(customer.profile.custom.subproCustomerID, 'client\_credentials', 'widget', 'sp-add-to-order-widget', 'sp-add-to-order')}"/>  <!-- Pass configuration and init the Subscribe Pro widget -->  <script>  // Setup config for Subscribe Pro  var widgetConfig = {  widgetElementId: '${widgetConfig.widgetElementId}',  apiBaseUrl: '${widgetConfig.apiAccessToken}',  apiAccessToken: '${widgetConfig.apiAccessToken}',  environmentKey: '${widgetConfig.environmentKey}',  customerId: '${widgetConfig.customerId}',  addToOrderElementClass: '${widgetConfig.addToOrderElementClass}'  };  // Call init  if (typeof subscribepro !== 'undefined') {  subscribepro.widgets.init('add-to-order', widgetConfig);  } else {  $.getScript( '${require('dw/system/Site').getCurrent().getCustomPreferenceValue('subproWidgetScriptUrl')}', function( data, textStatus, jqxhr ) {  subscribepro.widgets.init('add-to-order', widgetConfig);  });  }  </script>  </isif> |

Here is the expected result:



#### Modify productsetproduct.isml

Locate template */app\_storefront\_core/cartridge/templates/default/product/components/productsetproduct.isml*. Find input

<input type="hidden" name="pid" value="${pdict.Product.ID}" />

Insert code below after it to render product subscription options.

|  |
| --- |
| <iscomment>  product SubPro options  =============================================================  </iscomment>  <isif condition="${pdict.Product.custom.subproSubscriptionEnabled}">  <div class="subscription">  <div class="sp-add-to-order"></div>  <isinclude url="${URLUtils.url('SubPro-PDP', 'sku', pdict.Product.ID)}" />  </div>  </isif> |

Here is the expected result:



#### Modify producttopcontentPS.isml

Locate */app\_storefront\_core/cartridge/templates/default/product/producttopcontentPS.isml* template. Insert code below as shown on the screenshot.

|  |
| --- |
| <iscomment>  product SubPro options  =============================================================  </iscomment>  <isif condition="${pdict.Product.custom.subproSubscriptionEnabled}">  <h1>SUBPRO</h1>  <div class="subscription">  <div class="sp-add-to-order"></div>  <isinclude url="${URLUtils.url('SubPro-PDP', 'sku', pdict.Product.ID)}" />  </div>  </isif> |

Here is the expected result:

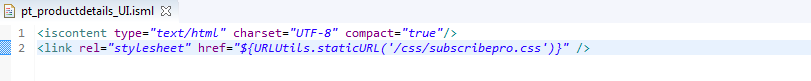


#### Modify pt\_productdetails\_UI.isml

Locate */app\_storefront\_core/cartridge/templates/default/product/pt\_productdetails\_UI.isml* template. Add line below to load styles.

|  |
| --- |
| <link rel="stylesheet" href="${URLUtils.staticURL('/css/subscribepro.css')}" /> |

Here is the expected result:



#### Modify variant.js script

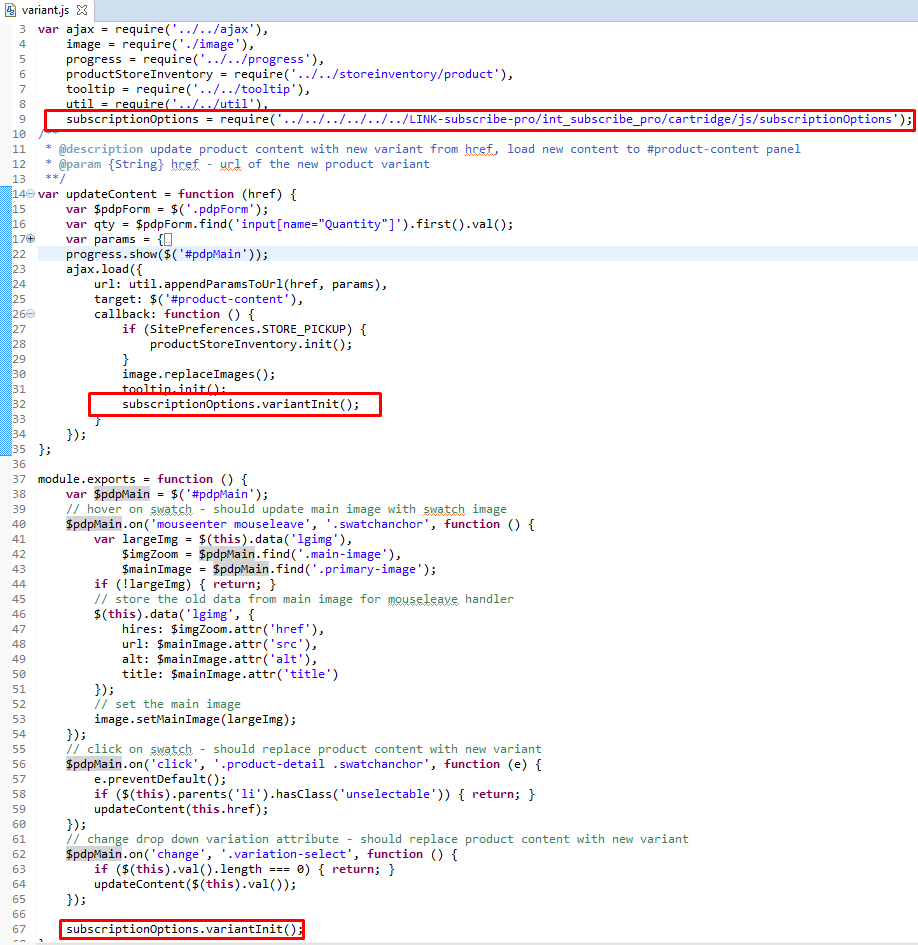
Locate */app\_storefront\_core/cartridge/js/pages/product/variant.js* script. Add logic to handle displaying of subscription options. Include script from the int\_subscribe\_pro cartridge *(Note: path in the require() should point to the file you copied into the app\_storefront\_core js directory, see above at the start of the code changes section*):

|  |
| --- |
| var subscriptionOptions = require('../subscriptionOptions'); |

and call function variantInit() in two locations as pictured below:

|  |
| --- |
| subscriptionOptions.variantInit(); |

Here is the expected result:

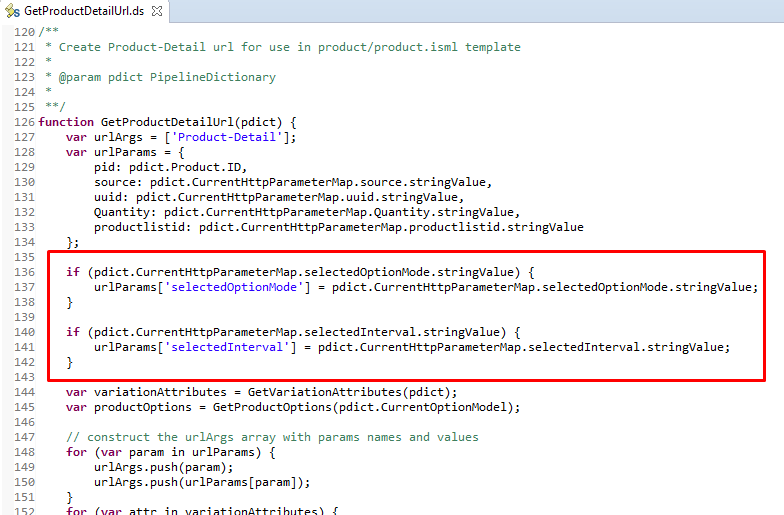


#### Modify GetProductDetailUrl.ds

Locate */app\_storefront\_core/cartridge/scripts/product/GetProductDetailUrl.ds* script. Find GetProductDetailUrl function and add code below to append selected subscription option mode and interval to URL.

|  |
| --- |
| if (pdict.CurrentHttpParameterMap.selectedOptionMode.stringValue) {  urlParams['selectedOptionMode'] = pdict.CurrentHttpParameterMap.selectedOptionMode.stringValue;  }    if (pdict.CurrentHttpParameterMap.selectedInterval.stringValue) {  urlParams['selectedInterval'] = pdict.CurrentHttpParameterMap.selectedInterval.stringValue;  } |

Here is the expected result:



#### 

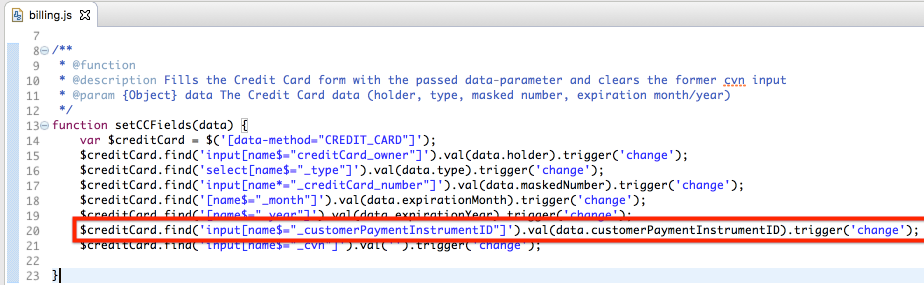
#### 

#### Modify billing.js

Locate */app\_storefront\_core/cartridge/js/pages/checkout/billing.js* script. Find setCCFields function and add code below to update the customer payment instrument input field with the selected UUID.

|  |
| --- |
| $creditCard.find('input[name$="\_customerPaymentInstrumentID"]').val(data.customerPaymentInstrumentID).trigger('change'); |

Here is the expected result:



### 

### 

### PLP

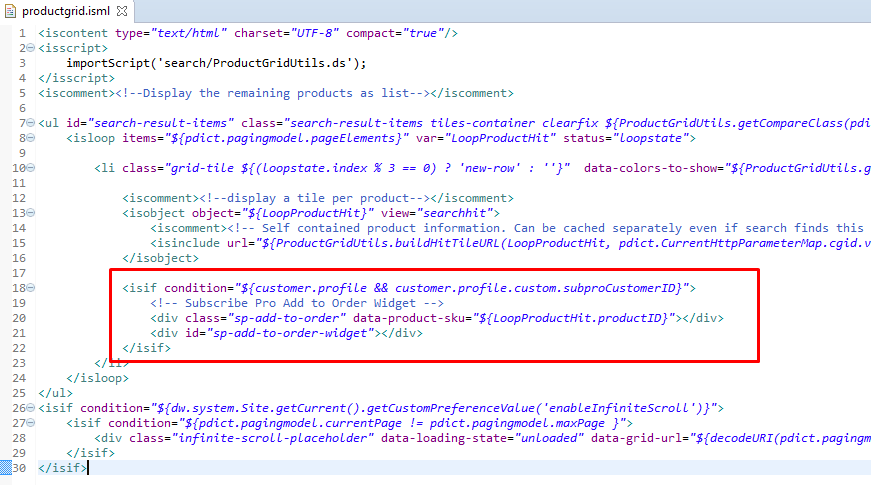
Changes need to be done to display Subscribe Pro “Add to Order” widget.

#### Modify productgrid.isml

Find template */app\_storefront\_core/cartridge/templates/default/search/productgrid.isml*. Insert code below to add tag which the Subscribe Pro “Add to Order” widget will be rendered to.

|  |
| --- |
| <isif condition="${customer.profile && customer.profile.custom.subproCustomerID}">  <!-- Subscribe Pro Add to Order Widget -->  <div class="sp-add-to-order" data-product-sku="${LoopProductHit.productID}"></div>  <div id="sp-add-to-order-widget"></div>  </isif> |

Here is the expected result:



#### Modify pt\_productsearchresult.isml

Find template */app\_storefront\_core/cartridge/templates/default/search/pt\_productsearchresult.isml*. Add code below at the end of body to render the Subscribe Pro “Add to Order” widget.

|  |
| --- |
| <isif condition="${customer.authenticated && customer.profile.custom.subproCustomerID}">  <iscomment>  Configure the Subscribe Pro Add to Order Widget  </iscomment>  <!-- Load the Subscribe Pro widget script -->  <script src="${require('dw/system/Site').getCurrent().getCustomPreferenceValue('subproWidgetScriptUrl')}"></script>  <isset name="widgetConfig" scope="page"  value="${require('/int\_subscribe\_pro/cartridge/scripts/subpro/helpers/WidgetsHelper').getWidgetConfig(customer.profile.custom.subproCustomerID, 'client\_credentials', 'widget', 'sp-add-to-order-widget', 'sp-add-to-order')}"/>  <!-- Pass configuration and init the Subscribe Pro widget -->  <script>  // Setup config for Subscribe Pro  var widgetConfig = <isprint value="${JSON.stringify(widgetConfig)}" encoding="off" />;  subscribepro.widgets.init('add-to-order', widgetConfig);  </script>  </isif> |

### Basket (cart)

Changes need to be done to display subscription options for products in cart.

#### Modify cart.isml

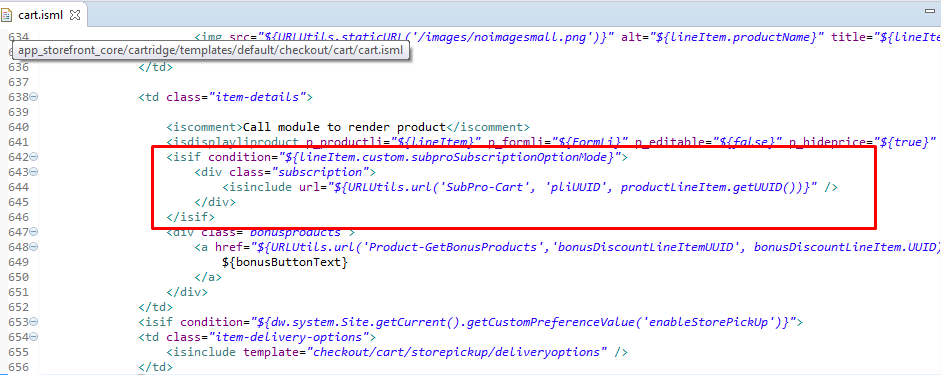
Find template */app\_storefront\_core/cartridge/templates/default/checkout/cart/cart.isml*. Add code below to include template with subscription options if product line item has SubPro subscriptions.

|  |
| --- |
| <isdisplayliproduct p\_productli="${lineItem}" p\_formli="${FormLi}" p\_editable="${true}" p\_hideprice="${true}" p\_hidepromo="${false}" />  <isif condition="${lineItem.custom.subproSubscriptionOptionMode}">  <div class="subscription">  <isinclude url="${URLUtils.url('SubPro-Cart', 'pliUUID', productLineItem.getUUID())}" />  </div>  </isif> |

Insert it after each <isdisplayliproduct/> tag (there should be two such places).

Here is the expected result:



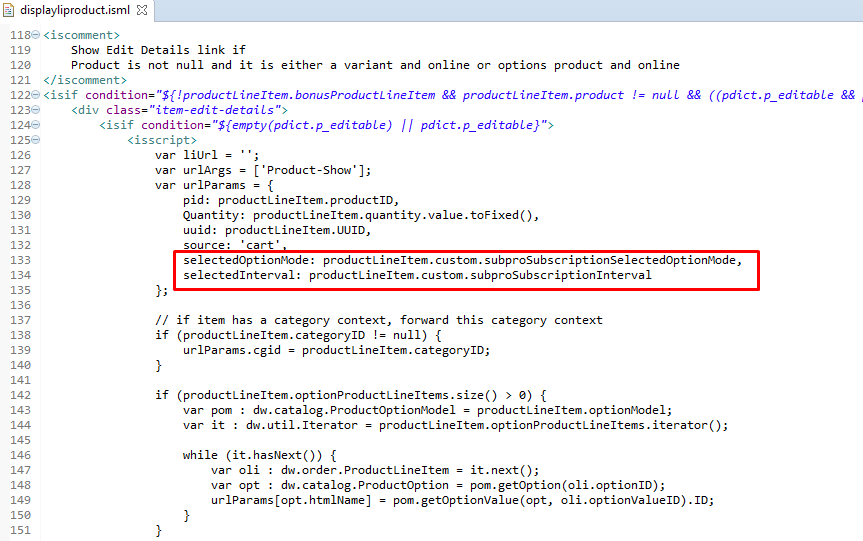


#### Modify displayliproduct.isml

Locate */app\_storefront\_core/cartridge/templates/default/product/components/displayliproduct.isml* template. Find <isscript> where URL is built. Add following lines to urlParams object to save subscription selected option mode and interval.

|  |
| --- |
| selectedOptionMode: productLineItem.custom.subproSubscriptionSelectedOptionMode,  selectedInterval: productLineItem.custom.subproSubscriptionInterval |

Here is the expected result:

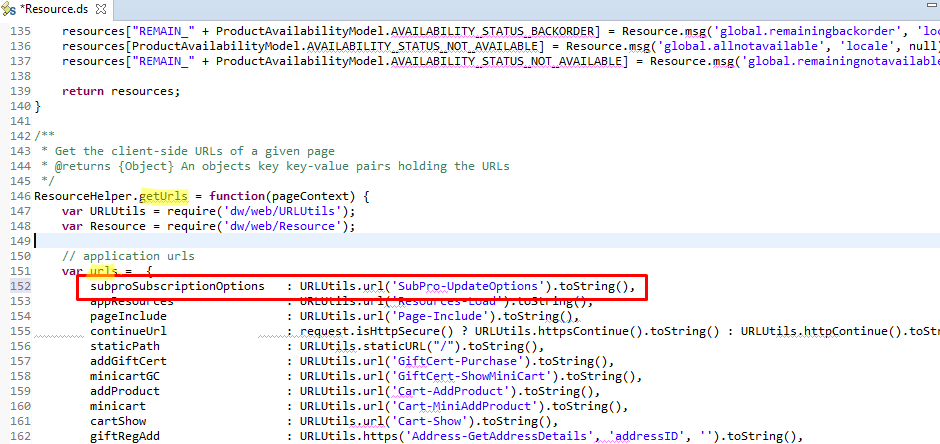


#### Modify Resource.ds

Find */app\_storefront\_core/cartridge/scripts/util/Resource.ds*. Find method getUrls(), variable urls and insert code below to add url to SubPro-UpdateOptions controller node which will update Subscription Options on Product Line Item in cart.

|  |
| --- |
| subproSubscriptionOptions : URLUtils.url('SubPro-UpdateOptions').toString() |

Here is the expected result:



#### Modify cart.js

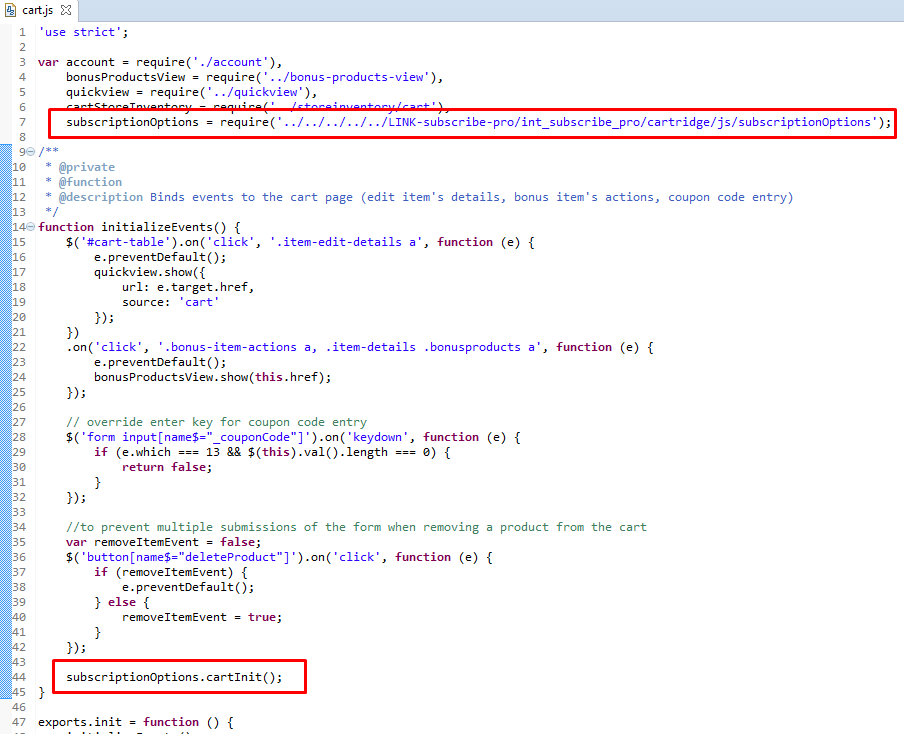
Find script */app\_storefront\_core/cartridge/js/pages/cart.js*. Add logic to handle displaying and modifying of the subscription options. Include js/subscriptionOptions.js file from the int\_subscribe\_pro cartridge *(Note: path in the require() should point to the file you copied into the app\_storefront\_core js directory, see above at the start of the code changes section):*

|  |
| --- |
| var subscriptionOptions = require('../subscriptionOptions'); |

and at the end of initializeEvents() function call function cartInit():

|  |
| --- |
| subscriptionOptions.cartInit(); |

Here is the expected result:



#### Modify CartModel.js

Find script model */app\_storefront\_controllers/cartridge/scripts/models/CartModel.js*.

Add logic to get product subscription options from template and then save them to Product Line Item custom attributes. Data that will be stored in Product Line Item:

* subproSubscriptionAvailableOptionMode
* subproSubscriptionOptionMode
* subproSubscriptionInterval
* subproSubscriptionAvailableIntervals
* subproSubscriptionDiscount
* subproSubscriptionIsDiscountPercentage
* subproContainsSubscriptions
* subproSubscriptionsToBeProcessed

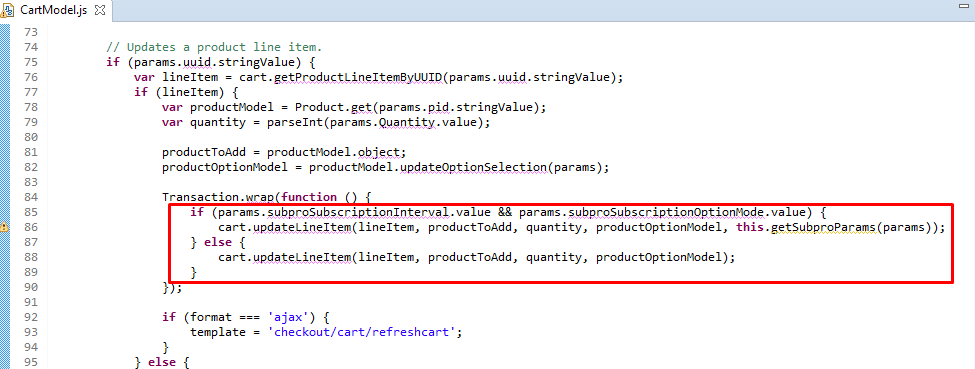
1. Find method *addProductToCart*. Find line:

cart.updateLineItem(lineItem, productToAdd, quantity, productOptionModel);

and replace with the code below:

|  |
| --- |
| if (params.subproSubscriptionInterval.value && params.subproSubscriptionOptionMode.value) {  cart.updateLineItem(lineItem, productToAdd, quantity, productOptionModel, cart.getSubproParams(params));  } else {  cart.updateLineItem(lineItem, productToAdd, quantity, productOptionModel);  } |

Here is the expected result:



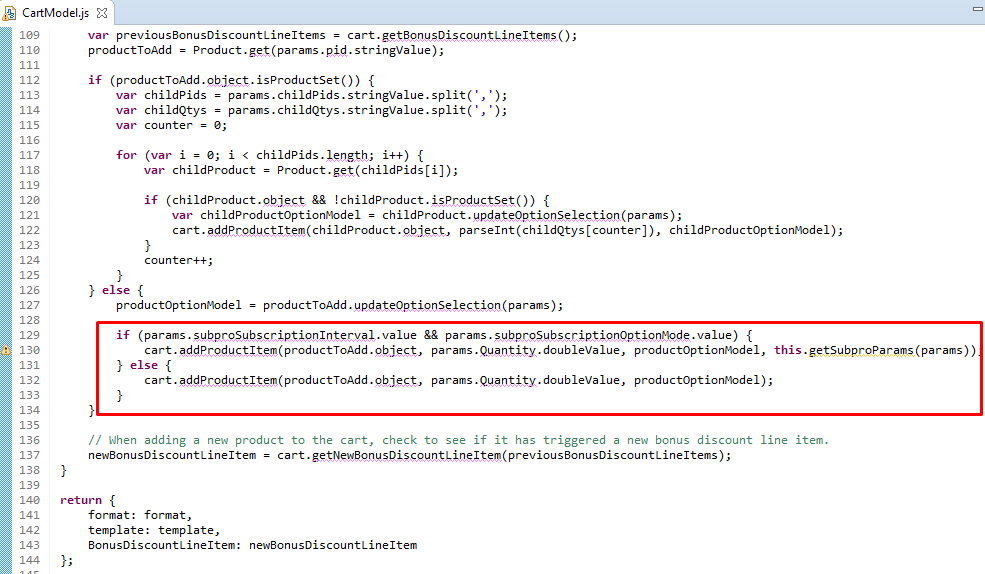
1. Find method *addProductToCart*. Find line:

cart.addProductItem(productToAdd.object, params.Quantity.doubleValue, productOptionModel);

and replace with the code below:

|  |
| --- |
| if (params.subproSubscriptionInterval.value && params.subproSubscriptionOptionMode.value) {  cart.addProductItem(productToAdd.object, params.Quantity.doubleValue, productOptionModel, cart.getSubproParams(params));  } else {  cart.addProductItem(productToAdd.object, params.Quantity.doubleValue, productOptionModel);  } |

Here is the expected result:



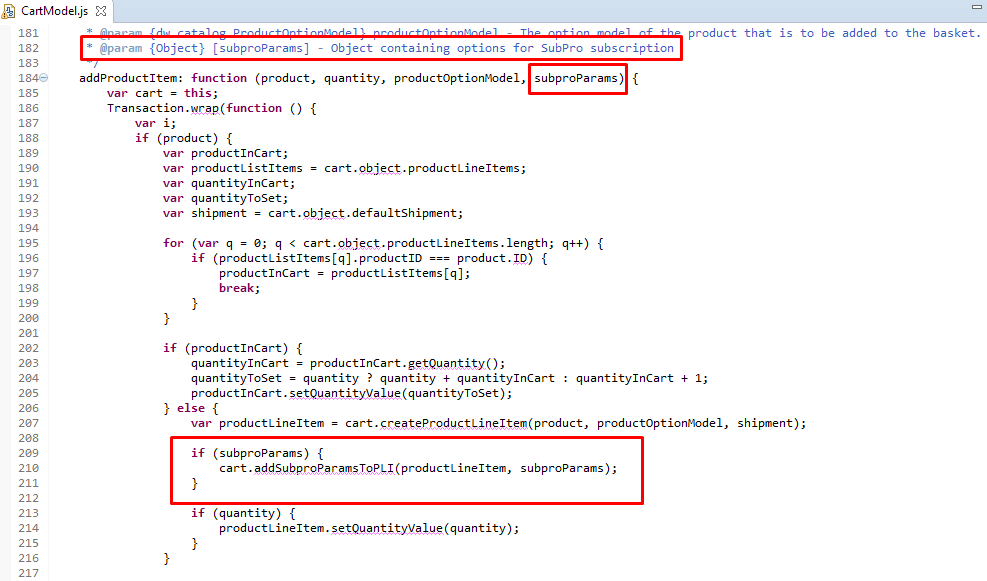
1. Find method *addProductItem*. Add *subproParams* to function parameters.  
   Add subproParams parameter to method (see image below).  
   Find line:

var productLineItem = cart.createProductLineItem(product, productOptionModel, shipment);

Add code below after it to update Product Line Item with Subscribe Pro parameters:

|  |
| --- |
| if (subproParams) {  cart.addSubproParamsToPLI(productLineItem, subproParams);  } |

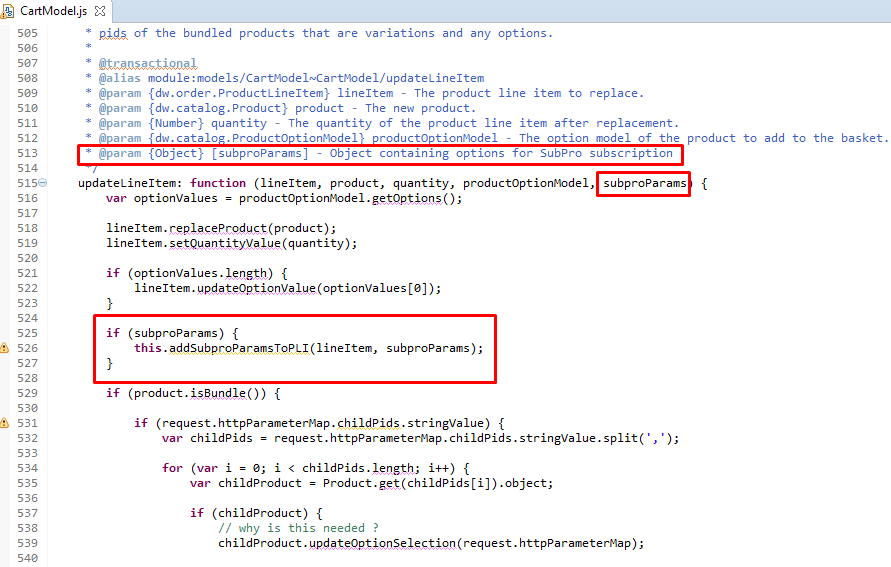
Here is the expected result:



1. Find method *updateLineItem*. Add *subproParams* to function parameters.  
   Add subproParams parameter to method (see image below).  
   Add code below after it to update Product Line Item with Subscribe Pro parameters:

|  |
| --- |
| if (subproParams) {  this.addSubproParamsToPLI(lineItem, subproParams);  } |

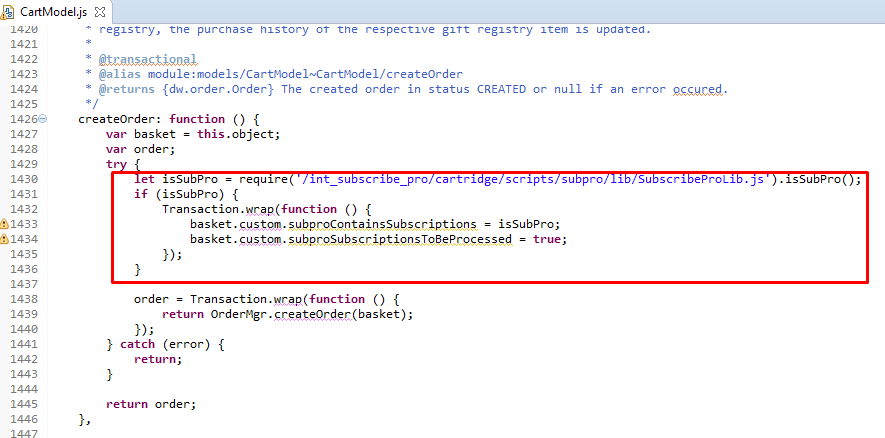
Here is the expected result:



1. Find method *createOrder*.  
   Add code below to update basket with Subscribe Pro properties.

|  |
| --- |
| let isSubPro = require('/int\_subscribe\_pro/cartridge/scripts/subpro/lib/SubscribeProLib.js').isSubPro();  if (isSubPro) {  Transaction.wrap(function () {  basket.custom.subproContainsSubscriptions = isSubPro;  basket.custom.subproSubscriptionsToBeProcessed = true;  });  } |

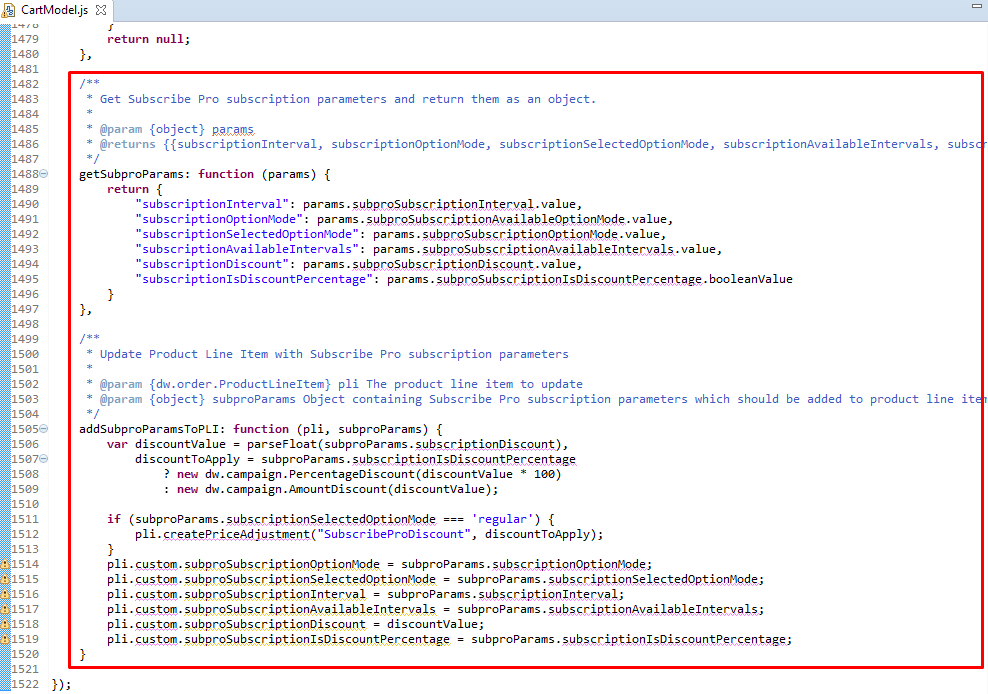
Here is the expected result:



1. Finally, add methods to CartModel object:

|  |
| --- |
| /\*\*  \* Get Subscribe Pro subscription parameters and return them as an object.  \*  \* @param {object} params  \* @returns {{subscriptionInterval, subscriptionOptionMode, subscriptionSelectedOptionMode, subscriptionAvailableIntervals, subscriptionDiscount, subscriptionIsDiscountPercentage: boolean}}  \*/  getSubproParams: function (params) {  return {  "subscriptionInterval": params.subproSubscriptionInterval.value,  "subscriptionOptionMode": params.subproSubscriptionAvailableOptionMode.value,  "subscriptionSelectedOptionMode": params.subproSubscriptionOptionMode.value,  "subscriptionAvailableIntervals": params.subproSubscriptionAvailableIntervals.value,  "subscriptionDiscount": params.subproSubscriptionDiscount.value,  "subscriptionIsDiscountPercentage": params.subproSubscriptionIsDiscountPercentage.booleanValue  }  },    /\*\*  \* Update Product Line Item with Subscribe Pro subscription parameters  \*  \* @param {dw.order.ProductLineItem} pli The product line item to update  \* @param {object} subproParams Object containing Subscribe Pro subscription parameters which should be added to product line item  \*/  addSubproParamsToPLI: function (pli, subproParams) {  var discountValue = parseFloat(subproParams.subscriptionDiscount),  discountToApply = subproParams.subscriptionIsDiscountPercentage  ? new dw.campaign.PercentageDiscount(discountValue \* 100)  : new dw.campaign.AmountDiscount(discountValue);    if (subproParams.subscriptionSelectedOptionMode === 'regular') {  pli.createPriceAdjustment("SubscribeProDiscount", discountToApply);  }  pli.custom.subproSubscriptionOptionMode = subproParams.subscriptionOptionMode;  pli.custom.subproSubscriptionSelectedOptionMode = subproParams.subscriptionSelectedOptionMode;  pli.custom.subproSubscriptionInterval = subproParams.subscriptionInterval;  pli.custom.subproSubscriptionAvailableIntervals = subproParams.subscriptionAvailableIntervals;  pli.custom.subproSubscriptionDiscount = discountValue;  pli.custom.subproSubscriptionIsDiscountPercentage = subproParams.subscriptionIsDiscountPercentage;  } |

Here is the expected result:



### Checkout

Changes need to be done to checkout process with subscription products.

#### Modify paymentmethods.isml

Locate template

*/app\_storefront\_core/cartridge/templates/default/checkout/billing/paymentmethods.isml*. Add code below to force show form with credit card payment methods.

1. Set isSubPro variable which will indicate if cart has items with subscription:

|  |
| --- |
| <isscript>  var SubscribeProLib = require('/int\_subscribe\_pro/cartridge/scripts/subpro/lib/SubscribeProLib');  </isscript>    <isset name="isSubPro" value="${SubscribeProLib.isSubPro()}" scope="page" /> |

1. Change following condition to:

|  |
| --- |
| <isif condition="${pdict.OrderTotal > 0 || isSubPro}"> |

1. Add code below to render message, informing customer that credit card is required.

|  |
| --- |
| <isif condition="${pdict.OrderTotal <= 0 && isSubPro}"></fieldset>  <div class="subpro-message">${require('dw/system/Site').getCurrent().getCustomPreferenceValue('subproCreditCardRequirementMsg')}</div>  </isif> |

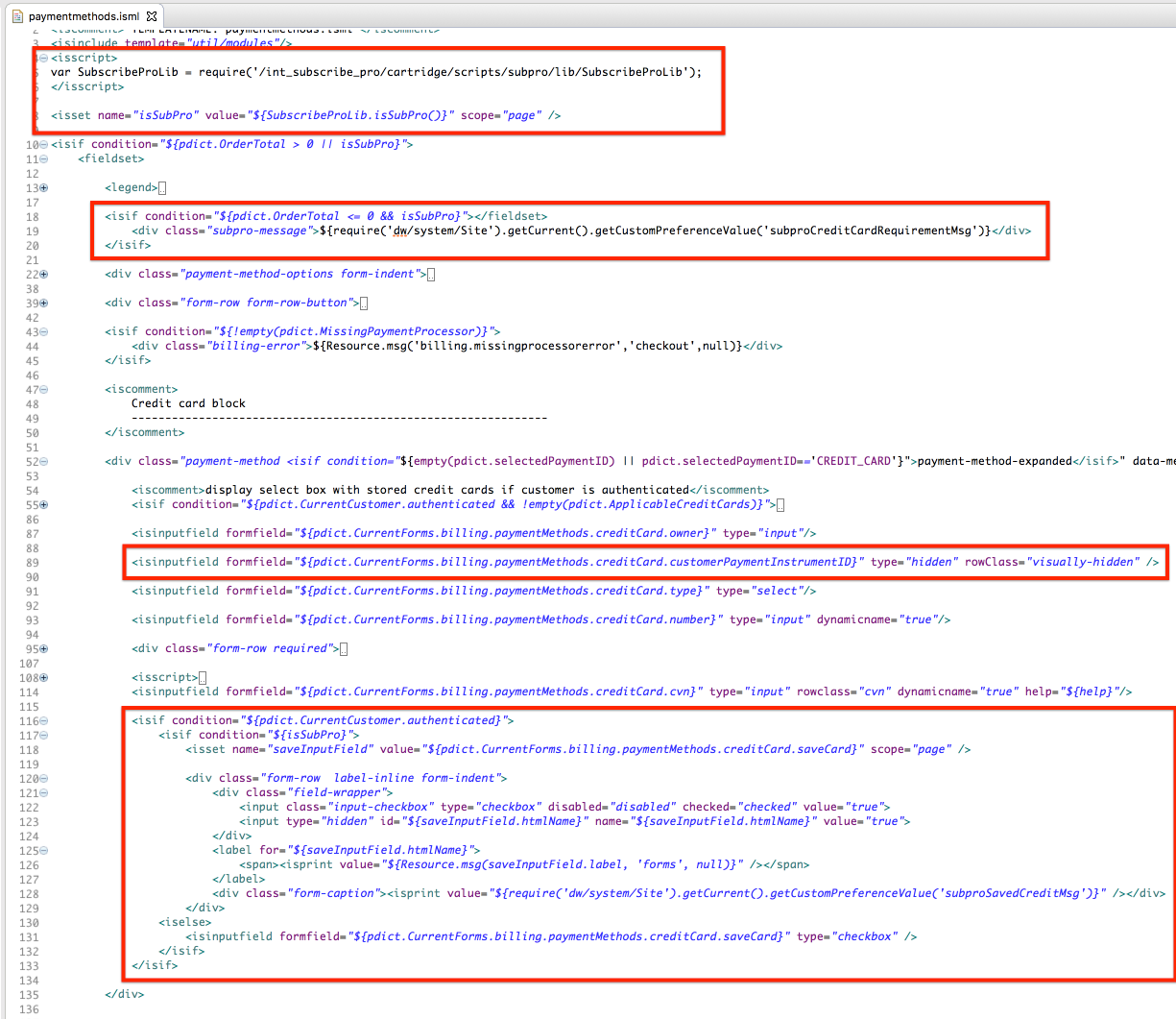
1. Add to force show form with credit card payment methods.

|  |
| --- |
| <isif condition="${pdict.CurrentCustomer.authenticated}">  <isif condition="${isSubPro}">  <isset name="saveInputField" value="${pdict.CurrentForms.billing.paymentMethods.creditCard.saveCard}" scope="page" />  <div class="form-row label-inline form-indent">  <div class="field-wrapper">  <input class="input-checkbox" type="checkbox" disabled="disabled" checked="checked" value="true">  <input type="hidden" id="${saveInputField.htmlName}" name="${saveInputField.htmlName}" value="true">  </div>  <label for="${saveInputField.htmlName}">  <span><isprint value="${Resource.msg(saveInputField.label, 'forms', null)}" /></span>  </label>  <div class="form-caption"><isprint value="${require('dw/system/Site').getCurrent().getCustomPreferenceValue('subproSavedCreditMsg')}" /></div>  </div>  <iselse>  <isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.saveCard}" type="checkbox" />  </isif>  </isif> |

1. Add hidden form element to track the selected Customer Payment Instrument.

|  |
| --- |
| <isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.customerPaymentInstrumentID}" type="hidden" rowClass="visually-hidden" /> |

Here is the expected result:



#### Modify checkoutlogin.isml

Locate template */app\_storefront\_core/cartridge/templates/default/checkout/checkoutlogin.isml*.

1. Add code below to display message to customer.

|  |
| --- |
| <isif condition="${pdict.isSubpro}">  <div class="subpro-message">${require('dw/system/Site').getCurrent().getCustomPreferenceValue('subproCheckoutLoginMsg')}</div>  </isif> |

1. Find condition

<isif condition="${!pdict.CurrentCustomer.registered}">

and replace it with code below to hide 'Checkout as Guest' option if cart has items with subscription.

|  |
| --- |
| <isif condition="${!pdict.CurrentCustomer.registered && !pdict.isSubpro}"> |

Here is the expected result:



#### 

#### Modify minilineitems.isml

Locate template */app\_storefront\_core/cartridge/templates/default/checkout/components/minilineitems.isml*. Add logic to show selected subscription options on mini cart and Order Summary Right Rail.

Add code block below after <div class="mini-cart-pricing"> tag.

|  |
| --- |
| <isif condition="${productLineItem.custom.subproSubscriptionOptionMode}">  <div class="mini-cart-subscription-options">  <isinclude url="${URLUtils.url('SubPro-OrderSummary', 'pliUUID', productLineItem.getUUID())}" />  </div>  </isif> |

Here is the expected result:



#### 

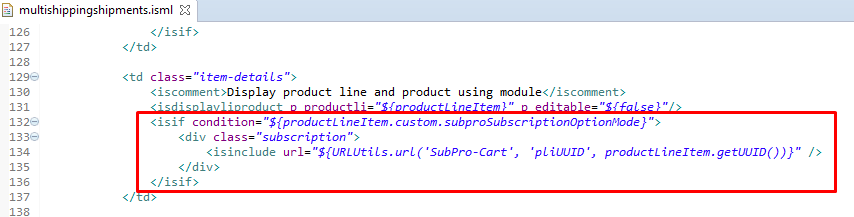
#### Modify multishippingshipments.isml

Locate template */app\_storefront\_core/cartridge/templates/default/checkout/shipping/multishipping/multishippingshipments.isml*.

Add code below after <isdisplayliproduct /> tag to show subscription options for multishipping.

|  |
| --- |
| <isif condition="${productLineItem.custom.subproSubscriptionOptionMode}">  <div class="subscription">  <isinclude url="${URLUtils.url('SubPro-Cart', 'pliUUID', productLineItem.getUUID())}" />  </div>  </isif> |

Here is the expected result:



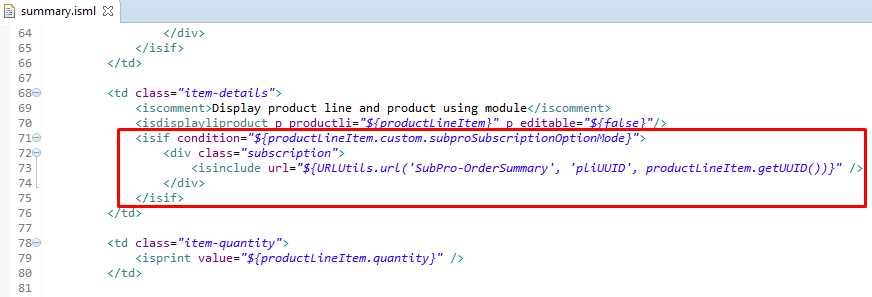
#### Modify summary.isml

Locate template */app\_storefront\_core/cartridge/templates/default/checkout/summary/summary.isml*.

Add code below after <isdisplayliproduct /> tag to show selected subscription options on Order Review page.

|  |
| --- |
| <isif condition="${productLineItem.custom.subproSubscriptionOptionMode}">  <div class="subscription">  <isinclude url="${URLUtils.url('SubPro-OrderSummary', 'pliUUID', productLineItem.getUUID())}" />  </div>  </isif> |

Here is the expected result:



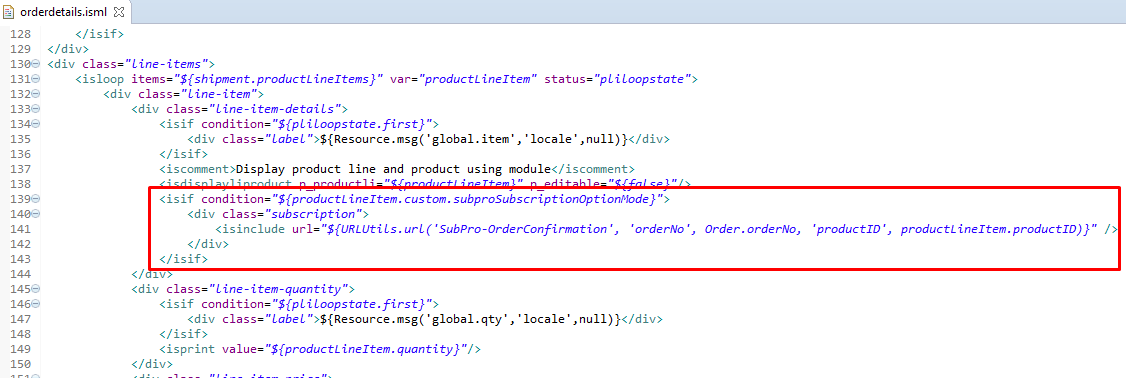
#### Modify orderdetails.isml

Locate template */app\_storefront\_core/cartridge/templates/default/components/order/orderdetails.isml*.

Add code below after <isdisplayliproduct /> tag to show elected subscription options on Order Confirmation page.

|  |
| --- |
| <isif condition="${productLineItem.custom.subproSubscriptionOptionMode}">  <div class="subscription">  <isinclude url="${URLUtils.url('SubPro-OrderConfirmation', 'orderNo', Order.orderNo, 'productID', productLineItem.productID)}" />  </div>  </isif> |

Here is the expected result:



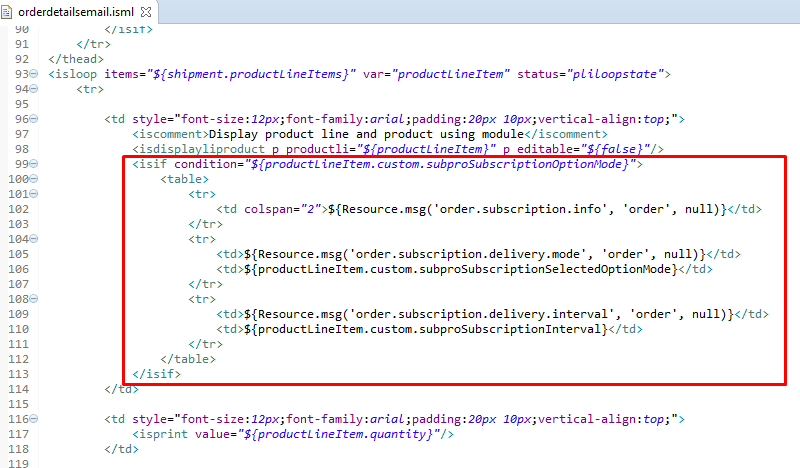
#### Modify orderdetailsemail.isml

Locate template */app\_storefront\_core/cartridge/templates/default/components/order/orderdetailsemail.isml*.

Add code below after <isdisplayliproduct /> tag to show selected subscription options in Order Confirmation Email.

|  |
| --- |
| <isif condition="${productLineItem.custom.subproSubscriptionOptionMode}">  <table>  <tr>  <td colspan="2">${Resource.msg('order.subscription.info', 'order', null)}</td>  </tr>  <tr>  <td>${Resource.msg('order.subscription.delivery.mode', 'order', null)}</td>  <td>${productLineItem.custom.subproSubscriptionSelectedOptionMode}</td>  </tr>  <tr>  <td>${Resource.msg('order.subscription.delivery.interval', 'order', null)}</td>  <td>${productLineItem.custom.subproSubscriptionInterval}</td>  </tr>  </table>  </isif> |

Here is the expected result:



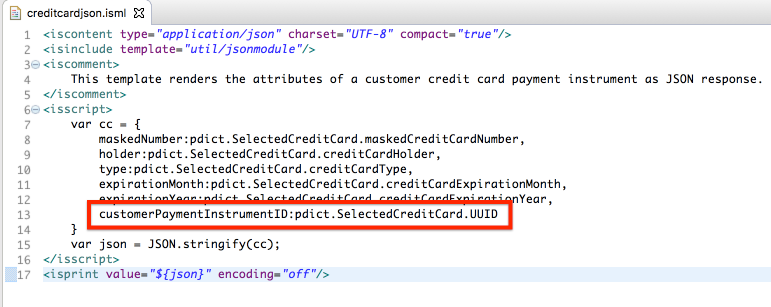
#### Modify creditcardjson.isml

Locate template */app\_storefront\_core/cartridge/templates/default/checkout/billing/creditcardjson.isml*

Add a new attribute to the returned JSON object that contains the saved payment instrument UUID

|  |
| --- |
| var cc = {  maskedNumber:pdict.SelectedCreditCard.maskedCreditCardNumber,  holder:pdict.SelectedCreditCard.creditCardHolder,  type:pdict.SelectedCreditCard.creditCardType,  expirationMonth:pdict.SelectedCreditCard.creditCardExpirationMonth,  expirationYear:pdict.SelectedCreditCard.creditCardExpirationYear,  customerPaymentInstrumentID:pdict.SelectedCreditCard.UUID  } |

Here is the expected result:

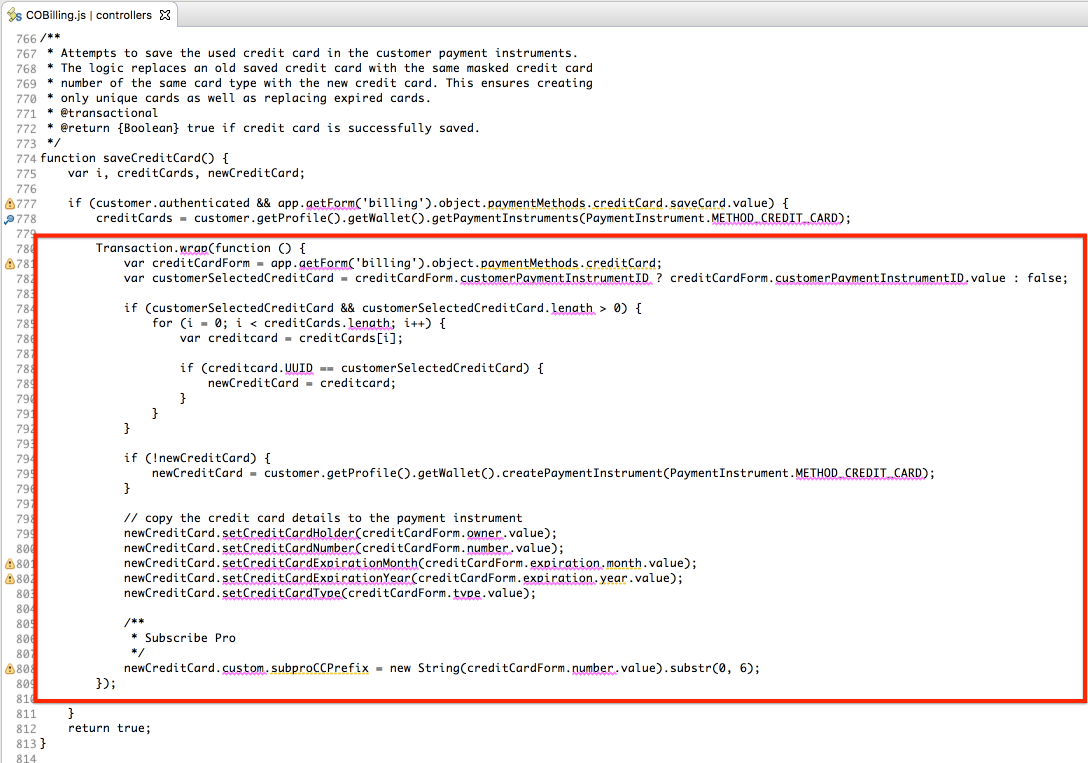


#### Modify COBilling.js

Locate */app\_storefront\_controllers/cartridge/controllers/COBilling.js* controller. Replace the code within the Transaction.wrap block to add save the credit cards first six digits and to update the saved customer payment instrument versus replacing it. This is important so that payment instruments UUID will persist for future orders placed via OCAPI.

|  |
| --- |
| /\*\* function saveCreditCard() \*/  Transaction.wrap(function () {  var creditCardForm = app.getForm('billing').object.paymentMethods.creditCard;  var customerSelectedCreditCard = creditCardForm.customerPaymentInstrumentID ? creditCardForm.customerPaymentInstrumentID.value : false;    if (customerSelectedCreditCard && customerSelectedCreditCard.length > 0) {  for (i = 0; i < creditCards.length; i++) {  var creditcard = creditCards[i];    if (creditcard.UUID == customerSelectedCreditCard) {  newCreditCard = creditcard;  }  }  }    if (!newCreditCard) {  newCreditCard = customer.getProfile().getWallet().createPaymentInstrument(PaymentInstrument.METHOD\_CREDIT\_CARD);  }    // copy the credit card details to the payment instrument  newCreditCard.setCreditCardHolder(creditCardForm.owner.value);  newCreditCard.setCreditCardNumber(creditCardForm.number.value);  newCreditCard.setCreditCardExpirationMonth(creditCardForm.expiration.month.value);  newCreditCard.setCreditCardExpirationYear(creditCardForm.expiration.year.value);  newCreditCard.setCreditCardType(creditCardForm.type.value);    /\*\*  \* Subscribe Pro  \*/  newCreditCard.custom.subproCCPrefix = new String(creditCardForm.number.value).substr(0, 6);  }); |

Here is the expected result:

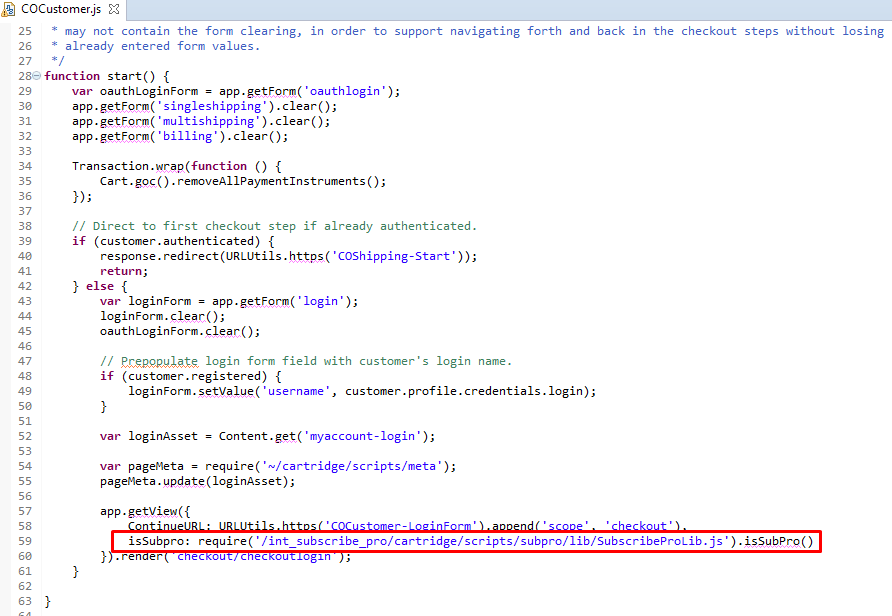


#### Modify COCustomer.js

Locate */app\_storefront\_controllers/cartridge/controllers/COCustomer.js* controller. Add line below to send parameter isSubpro, which specify if cart has products with SubPro subscription, to pdict.

|  |
| --- |
| isSubpro: require('/int\_subscribe\_pro/cartridge/scripts/subpro/lib/SubscribeProLib.js').isSubPro() |

Here is the expected result:



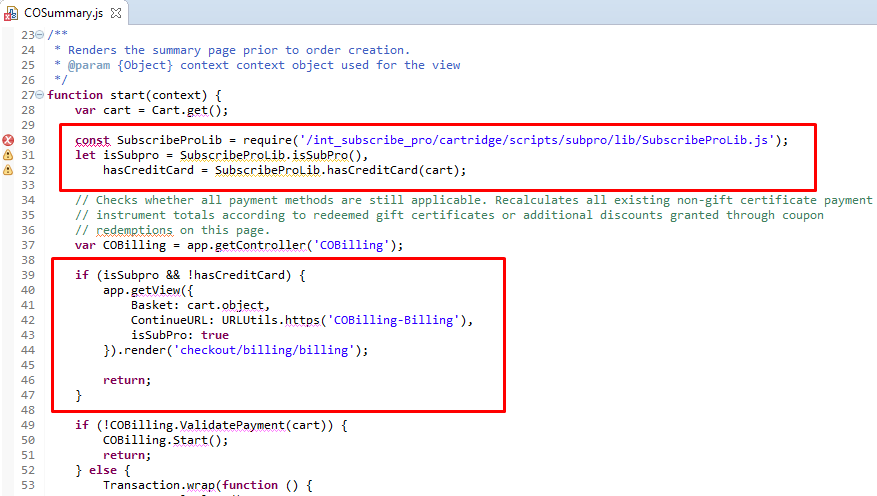
#### Modify COSummary.js

Locate */app\_storefront\_controllers/cartridge/controllers/COSummary.js* controller. Add logic to check if cart has any credit card as a payment method (it is required for subscription). If it doesn't we pass isSubPro: true parameter to checkout/billing/billing template, forcing it to show form with credit card payment methods.

Add logic from code block below to start() method

|  |
| --- |
| const SubscribeProLib = require('/int\_subscribe\_pro/cartridge/scripts/subpro/lib/SubscribeProLib.js');  let isSubpro = SubscribeProLib.isSubPro(),  hasCreditCard = SubscribeProLib.hasCreditCard(cart);    if (isSubpro && !hasCreditCard) {  app.getView({  Basket: cart.object,  ContinueURL: URLUtils.https('COBilling-Billing'),  isSubPro: true  }).render('checkout/billing/billing');    return;  } |

Here is the expected result:



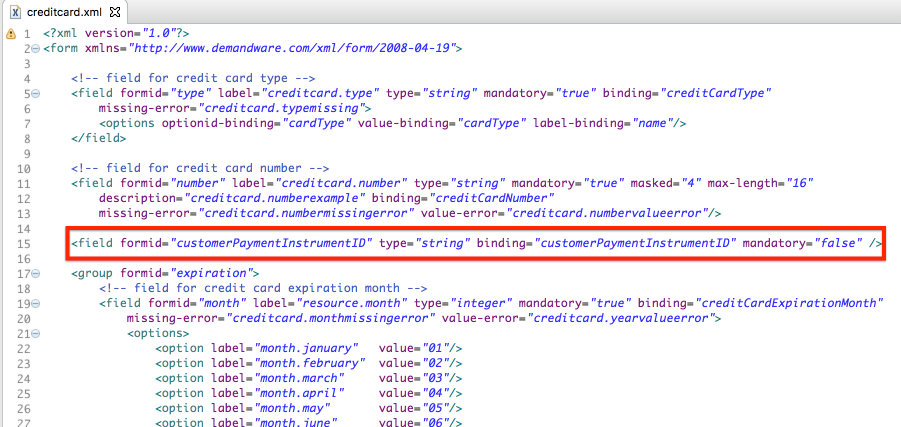
#### Modify creditcard.xml

Locate */app\_storefront\_core/cartridge/forms/default/creditcard.xml* form definition. Add a new form field for the customer’s saved payment instrument id.

Add the following form field:

|  |
| --- |
| <field formid="customerPaymentInstrumentID" type="string" binding="customerPaymentInstrumentID" mandatory="false" /> |

Here is the expected result:



### Account

Changes need to be done to My Account section for displaying customer Subscribe Pro subscriptions. To display such information Subscribe Pro Hosted Widgets are used. Changes need to be done to checkout process with subscription products.

***Modify controller Address.js***

Locate /app\_storefront\_controllers/cartridge/controllers/Address.js.

Replace the list() method with the following:

|  |
| --- |
| function list() {  var pageMeta = require('~/cartridge/scripts/meta');  var content = app.getModel('Content').get('myaccount-addresses');  if (content) {  pageMeta.update(content.object);  }  var viewParams = {};  if (session.custom.newAddress) {  viewParams.newAddress = {"address":session.custom.newAddress.sp};  viewParams.newAddressSfccId = session.custom.newAddress.sfcc.getID();  session.custom.newAddress = null;  }  if (session.custom.updatedOldAddress && session.custom.updatedNewAddress) {  viewParams.updatedAddress = {"prev\_address": session.custom.updatedOldAddress.sp, "address": session.custom.updatedNewAddress.sp};  session.custom.updatedNewAddress = null;  session.custom.updatedOldAddress = null;  }  if (session.custom.deletedAddress) {  viewParams.deletedAddress = {"address": session.custom.deletedAddress.sp};  session.custom.deletedAddress = null;  }  app.getView(viewParams).render('account/addressbook/addresslist');  } |

In the handleForm() method make the following changes.

After the addressForm variable declaration declare the addressHelper variable:

|  |
| --- |
| var addressHelper = require('int\_subscribe\_pro/cartridge/scripts/subpro/helpers/AddressHelper'); |

Inside the addressForm.handleAction() call, replace the create() method with the following:

|  |
| --- |
| create: function () {  var newAddress;  if (!session.forms.profile.address.valid || !(newAddress = Address.create(session.forms.profile.address))) {  response.redirect(URLUtils.https('Address-Add'));  success = false;  }  session.custom.newAddress = {  "sfcc": newAddress,  "sp": addressHelper.getSubproAddress(newAddress, session.customer.profile, false, true)  };  success = true;  }, |

Replace the edit() method with the following:

|  |
| --- |
| edit: function () {  if (!session.forms.profile.address.valid) {  success = false;  message = 'Form is invalid';  }  try {  // We can't use Address.get() because it doesn't return a CustomerAddress object, and we need to be able to access  // the 'custom' attributes on the address. Instead, we use the address book  let addressBook = customer.profile.addressBook;  var prevAddress = addressBook.getAddress(request.httpParameterMap.addressid.value);  session.custom.updatedOldAddress = {  "sfcc": prevAddress,  "sp": addressHelper.getSubproAddress(prevAddress, session.customer.profile, true, true)  };  var updatedAddress = Address.update(request.httpParameterMap.addressid.value, session.forms.profile.address);  session.custom.updatedNewAddress = {  "sfcc": updatedAddress,  "sp": addressHelper.getSubproAddress(updatedAddress, session.customer.profile, true, true)  };  success = true;  } catch (e) {  success = false;  message = e.message;  }  }, |

And replace the remove() method with the following:

|  |
| --- |
| remove: function () {  // We can't use Address.get() because it doesn't return a CustomerAddress object, and we need to be able to access  // the 'custom' attributes on the address. Instead, we use the address book  let addressBook = customer.profile.addressBook;  var deletedAddress = addressBook.getAddress(session.forms.profile.address.addressid.value);  if (Address.remove(session.forms.profile.address.addressid.value)) {  deletedAddress = null;  success = false;  }  else {  session.custom.deletedAddress = {  "sfcc": deletedAddress,  "sp": addressHelper.getSubproAddress(savedDeletedAddress, session.customer.profile, true, true)  };  }  } |

In the Delete function, add the following code between the CustomerStatusCodes and deleteAddressResult varilable declarations:

|  |
| --- |
| var addressHelper = require('int\_subscribe\_pro/cartridge/scripts/subpro/helpers/AddressHelper');  // We can't use Address.get() because it doesn't return a CustomerAddress object, and we need to be able to access  // the 'custom' attributes on the address. Instead, we use the address book  let addressBook = customer.profile.addressBook;  var deletedAddress = addressBook.getAddress(request.httpParameterMap.AddressID.value);  session.custom.deletedAddress = {  "sfcc": deletedAddress,  "sp": addressHelper.getSubproAddress(deletedAddress, session.customer.profile, true, true)  }; |

Lastly, add the follow new route:

|  |
| --- |
| function SetSPAddressID() {  var addressBook = customer.getProfile().getAddressBook();  var address = addressBook.getAddress(request.httpParameterMap.addressId.value);  var addressHelper = require('/int\_subscribe\_pro/cartridge/scripts/subpro/helpers/AddressHelper');  addressHelper.setSubproAddressID(address, request.httpParameterMap.spAddressId.value);  let r = require('~/cartridge/scripts/util/Response');  r.renderJSON({success: true});  }  /\*\* Sets Subscribe Pro Address ID on SFCC Address \*/  exports.SetSPAddressID = guard.ensure(['get', 'https', 'loggedIn'], SetSPAddressID); |

#### Modify controller PaymentInstruments.js

Locate controller /app\_storefront\_controllers/cartridge/controllers/PaymentInstruments.js

Replace the list() method with the following:

|  |
| --- |
| function list() {  var wallet = customer.getProfile().getWallet();  var paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  var pageMeta = require('~/cartridge/scripts/meta');  var paymentForm = app.getForm('paymentinstruments');  paymentForm.clear();  paymentForm.get('creditcards.storedcards').copyFrom(paymentInstruments);  pageMeta.update(dw.content.ContentMgr.getContent('myaccount-paymentsettings'));  var viewParams = {  PaymentInstruments: paymentInstruments  };  if (session.custom.newCard) {  viewParams.newCard = {"payment\_profile": session.custom.newCard.sp};  viewParams.newCardSfccId = session.custom.newCard.sfcc.getUUID();  session.custom.newCard = null;  }  if (session.custom.updatedCard) {  viewParams.updatedCard = {"payment\_profile": session.custom.updatedCard.sp};  session.custom.updatedCard = null;  }  if (session.custom.deletedCard) {  viewParams.deletedCard = {"payment\_profile": session.custom.deletedCard.sp};  session.custom.deletedCard = null;  }  app.getView(viewParams).render('account/payment/paymentinstrumentlist');  } |

Find the create() method. Before the if(isDuplicateCard) condition, add:

|  |
| --- |
| let subproPaymentProfileID = null; |

Then, inside the condition add the following before the wallet.removePaymentInstrument() call:

|  |
| --- |
| if (oldCard.custom.subproPaymentProfileID) {  subproPaymentProfileID = oldCard.custom.subproPaymentProfileID;  } |

Directly between the Transaction.commit() and paymentForm.clear() calls, add the following code:

|  |
| --- |
| // Save data to session for hosted wallet widget integration  var paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  var saveCard;  for (var i = 0; i < paymentInstruments.length; i++) {  var card = paymentInstruments[i];  if (card.creditCardNumber === ccNumber) {  saveCard = card;  break;  }  }  var paymentsHelper = require('int\_subscribe\_pro/cartridge/scripts/subpro/helpers/PaymentsHelper');  if (!isDuplicateCard) {  session.custom.newCard = {  "sp": paymentsHelper.getSubscriptionPaymentProfile(session.customer.profile, saveCard, {}, false),  "sfcc": saveCard  };  if (null === session.custom.newCard) {  Logger.info('New card could not be mapped to a payment profile.');  }  }  else {  Transaction.begin();  try {  saveCard.custom.subproPaymentProfileID = subproPaymentProfileID;  }  catch (err) {  Transaction.rollback();  return false;  }  Transaction.commit();  session.custom.updatedCard = {  "sp": paymentsHelper.getSubscriptionPaymentProfile(session.customer.profile, saveCard, {}, true),  "sfcc": saveCard  };  if (null === session.custom.updatedCard) {  Logger.info('New card could not be mapped to a payment profile.');  }  } |

In the Delete() method, right inside the remove() function, add the following:

|  |
| --- |
| var paymentsHelper = require('int\_subscribe\_pro/cartridge/scripts/subpro/helpers/PaymentsHelper');  session.custom.deletedCard = {  "sp": paymentsHelper.getSubscriptionPaymentProfile(session.customer.profile, action.object, {}, true),  "sfcc": action.object  }; |

Add the following route:

|  |
| --- |
| function SetSPPaymentProfileID() {  var wallet = customer.getProfile().getWallet();  var paymentInstruments = wallet.getPaymentInstruments('CREDIT\_CARD');  let paymentInstrumentId = request.httpParameterMap.paymentInstrumentId.value;  var paymentsHelper = require('/int\_subscribe\_pro/cartridge/scripts/subpro/helpers/PaymentsHelper');  let paymentInstrument = null;  for (let i in paymentInstruments) {  if (paymentInstrumentId == paymentInstruments[i].getUUID()) {  paymentInstrument = paymentInstruments[i];  }  }  let success = paymentInstrument != null;  if (success) {  paymentsHelper.setSubproPaymentProfileID(paymentInstrument, request.httpParameterMap.spPaymentProfileId.value);  }  let r = require('~/cartridge/scripts/util/Response');  r.renderJSON({success: true});  }  /\*\* Sets the Subscribe Pro Payment Profile ID on a SFCC Payment Instrument \*/  /\* @see module:controllers/PaymentInstruments~SetSPPaymentProfileID \*/  exports.SetSPPaymentProfileID = guard.ensure(['https', 'get', 'loggedIn'], SetSPPaymentProfileID); |

#### Modify addressinclude.isml

Locate template */app\_storefront\_core/cartridge/templates/default/account/addressbook/addressinclude.isml*.

Insert code below at the end of the file to facilitate configuring and loading the address book widget, “AddressBookAssist”.

|  |
| --- |
| <isset name="hasNewAddress" value="${ pdict.newAddress != null }" scope="page" />  <isset name="hasUpdatedAddress" value="${ pdict.updatedAddress != null }" scope="page" />  <isset name="hasDeletedAddress" value="${ pdict.deletedAddress != null }" scope="page" />  <isset name="loadWidget" value="${ hasNewAddress || hasUpdatedAddress || hasDeletedAddress }" scope="page" />  <isif condition="${customer.profile.custom.subproCustomerID && loadWidget}">  <!-- Subscribe Pro Address Book Assist Widget -->  <div id="sp-address-book-widget"></div>  <iscomment>  Configure the Subscribe Pro Address Book Assist Widget  </iscomment>  <!-- Load the Subscribe Pro widget script -->  <script src="${require('dw/system/Site').getCurrent().getCustomPreferenceValue('subproAddressWidgetScriptUrl')}"></script>  <isset name="widgetConfig" scope="page"  value="${require('/int\_subscribe\_pro/cartridge/scripts/subpro/helpers/WidgetsHelper').getWidgetConfig(customer.profile.custom.subproCustomerID, 'client\_credentials', 'widget', 'sp-address-book-widget')}"/>  <!-- Pass configuration and init the Subscribe Pro widget -->  <script>  // Setup config for Subscribe Pro  var widgetConfig = <isprint value="${JSON.stringify(widgetConfig)}" encoding="off" />;  AddressBookAssist.init(widgetConfig);  <isif condition="${ hasNewAddress }">  var newAddress = <isprint value="${JSON.stringify(pdict.newAddress)}" encoding="off" />;  var newAddressSfccId = '<isprint value="${pdict.newAddressSfccId}" encoding="off" />';  AddressBookAssist.onAddressCreated(newAddress).then(function (result) {  $.get("${URLUtils.url('Address-SetSPAddressID')}", {"addressId": newAddressSfccId, "spAddressId": result.id});  });  </isif>  <isif condition="${ hasUpdatedAddress }">  var updatedAddress = <isprint value="${JSON.stringify(pdict.updatedAddress)}" encoding="off" />;  AddressBookAssist.onAddressUpdated(updatedAddress);  </isif>  <isif condition="${ hasDeletedAddress }">  var deletedAddress = <isprint value="${JSON.stringify(pdict.deletedAddress)}" encoding="off" />;  AddressBookAssist.onAddressDeleted(deletedAddress);  </isif>  </script>  </isif> |

#### Modify paymentinstrumentlist.isml

Locate template */app\_storefront\_core/cartridge/templates/default/account/payment/paymentinstrumentlist.isml*.

1. Inside the isdecorate tag, set the variable to define if customer is Subscribe Pro customer

|  |
| --- |
| <isset name="isSubproCustomer" value="${customer.profile.custom.subproCustomerID}" scope="page" /> |

1. Insert code below just before the closing div tag for the <div class="paymentslist"> block to add Subscribe Pro “Address Wallet Assist” widget div.

|  |
| --- |
| <isset name="hasNewCard" value="${ pdict.newCard != null }" scope="page" />  <isset name="hasUpdatedCard" value="${ pdict.updatedCard != null }" scope="page" />  <isset name="hasDeletedCard" value="${ pdict.deletedCard != null }" scope="page" />  <isset name="loadWidget" value="${ hasNewCard || hasUpdatedCard || hasDeletedCard }" scope="page" />  <isif condition="${isSubproCustomer}">  <!-- Subscribe Pro Wallet Assist Widget -->  <div id="sp-wallet-assist-widget"></div>  </isif> |

1. Insert code below to load, configure and init the Subscribe Pro “Address Wallet Assist” widget

|  |
| --- |
| <isif condition="${isSubproCustomer && loadWidget}">  <iscomment>  Configure the Subscribe Pro Wallet Assist Widget  </iscomment>    <!-- Load the Subscribe Pro widget script -->  <script src="${require('dw/system/Site').getCurrent().getCustomPreferenceValue('subproWalletWidgetScriptUrl')}"></script>    <isset name="widgetConfig" scope="page"  value="${require('/int\_subscribe\_pro/cartridge/scripts/subpro/helpers/WidgetsHelper').getWidgetConfig(customer.profile.custom.subproCustomerID, 'client\_credentials', 'widget', 'sp-wallet-assist-widget')}"/>    <!-- Pass configuration and init the Subscribe Pro widget -->  <script>  // Setup config for Subscribe Pro  var widgetConfig = <isprint value="${JSON.stringify(widgetConfig)}" encoding="off" />;  WalletAssist.init(widgetConfig);  <isif condition="${ hasNewCard }">  var newCard = <isprint value="${JSON.stringify(pdict.newCard)}" encoding="off" />;  WalletAssist.onPaymentProfileCreated(newCard, function (result) {  var newCardSfccId = '<isprint value="${pdict.newCardSfccId}" encoding="off" />';  $.get("${URLUtils.url('PaymentInstruments-SetSPPaymentProfileID')}", {"paymentInstrumentId": newCardSfccId, "spPaymentProfileId": result.id});  });  </isif>  <isif condition="${ hasUpdatedCard }">  var updatedCard = <isprint value="${JSON.stringify(pdict.updatedCard)}" encoding="off" />;  WalletAssist.onPaymentProfileUpdated(updatedCard);  </isif>  <isif condition="${ hasDeletedCard }">  var deletedCard = <isprint value="${JSON.stringify(pdict.deletedCard)}" encoding="off" />;  WalletAssist.onPaymentProfileDeleted(deletedCard);  </isif>  </script>  </isif> |

#### My Account side navigation

To display My subscriptions in the My Account side navigation edit asset account-nav-registered

(*Merchant Tools > Content > Content Assets > account-nav-registered*).

Add the following:

|  |
| --- |
| <li><a title="Manage subscriptions" href="$httpsUrl(Subscriptions-List)$">My Subscriptions</a></li> |

#### Modify controller Account.js

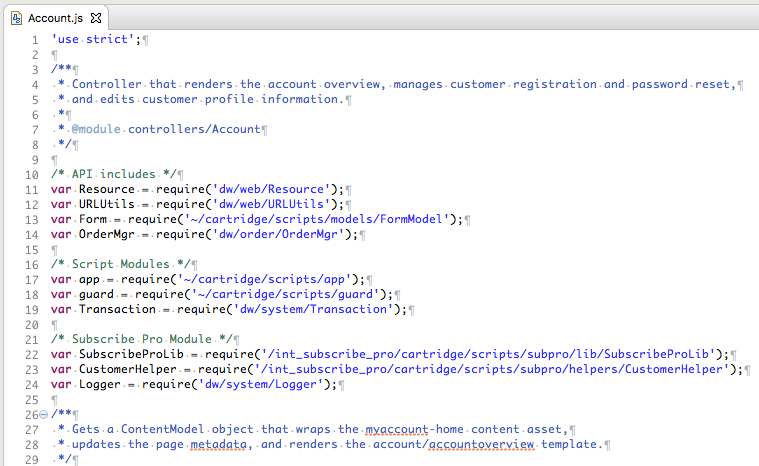
Locate */app\_storefront\_controllers/cartridge/controllers/Account.js* controller. Add the following includes to the top of the controller to be used later on in the controller:

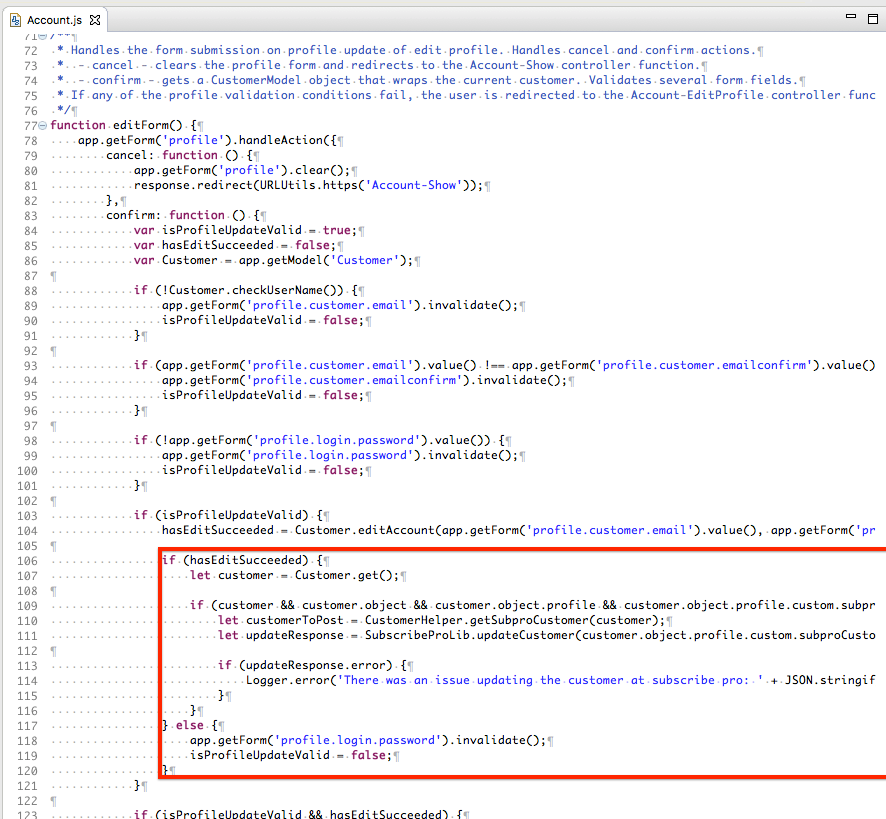
|  |
| --- |
| /\* Subscribe Pro Module \*/  var SubscribeProLib = require('/int\_subscribe\_pro/cartridge/scripts/subpro/lib/SubscribeProLib');  var CustomerHelper = require('/int\_subscribe\_pro/cartridge/scripts/subpro/helpers/CustomerHelper');  var Logger = require('dw/system/Logger'); |

Modify the *hasEditSucceeded* conditional in the *editForm* method to have some logic in the case of the conditional returning true, to update the customer profile at Subscribe Pro:

|  |
| --- |
| if (hasEditSucceeded) {  let customer = Customer.get();  if (customer && customer.object && customer.object.profile && customer.object.profile.custom.subproCustomerID) {  let customerToPost = CustomerHelper.getSubproCustomer(customer);  let updateResponse = SubscribeProLib.updateCustomer(customer.object.profile.custom.subproCustomerID, customerToPost);  if (updateResponse.error) {  Logger.error('There was an issue updating the customer at subscribe pro: ' + JSON.stringify(updateResponse));  }  }  } else {  /\* Existing false logic for hasEditSucceeded \*/ |

Here is the expected result:





## 

## 

## External Interfaces

Web service calls made to the Subscribe Pro RESTful API will make use of Sales Force Commerce Cloud Services:

<https://documentation.demandware.com/DOC1/topic/com.demandware.dochelp/WebServices/Webservices.html>

Application code should not call the web services directly but should instead call methods within the Subscribe Pro Javascript module:

*LINK-subscribe-pro/int\_subscribe\_pro/cartridge/scripts/subpro/lib/SubscribeProLib.js*

### Service Profile

There are separate profiles for GET vs POST request to enable / disable the circuit breaker.

|  |  |  |
| --- | --- | --- |
| **Name** | subpro.http.prof.get | subpro.http.prof.post |
| **Description** | Profile for GET requests | Profile for POST requests |
| **Connection Timeout (ms)** | 5000 | 5000 |
| **Enable Circuit Breaker** | yes | no |
| **Max Circuit Breaker Calls** | 3 | 0 |
| **Enable Rate Limit** | 1000 | 1000 |
| **Max Rate Limit Calls** |  |  |
| **Rate Limit Interval (ms)** |  |  |

### Service Credentials

The *"siteName"* portion of the Service Credential comes from the Site Preference: *subproAPICredSuffix*. This allows merchants to have different credentials for each site.

The default value of the Site Preference is siteName, so that it will automatically match the default credentials.

Please note that the user password will need to be updated after import.

|  |  |
| --- | --- |
| **Name** | subpro.http.cred.siteName |
| **URL** | https://api.subscribepro.com/services/v2/{ENDPOINT}?{PARAMS} |
| **User** | Subscribe Pro Client ID |
| **Password** | Subscribe Pro Client Secret |

|  |  |
| --- | --- |
| **Name** | subpro.http.cred.oauth.siteName |
| **URL** | https://api.subscribepro.com/oauth/v2/{ENDPOINT}?{PARAMS} |
| **User** | Subscribe Pro Client ID |
| **Password** | Subscribe Pro Client Secret |

### 

### 

### Service Endpoints

|  |  |  |
| --- | --- | --- |
| **ID** | **API endpoint** | **Description** |
| subpro.http.get.config | GET /services/v2/config | Retrieve configuration for the authenticated account |
| subpro.http.get.subscriptions | GET /services/v2/subscriptions | Retrieve a collection of subscriptions. |
| subpro.http.post.subscription | POST /services/v2/subscription | Create a new subscription |
| subpro.http.post.addresses | POST /services/v2/address | Create / Update a Subscribe Pro addresses |
| subpro.http.post.addressfindcreate | POST /services/v2/address/find-or-create | Find a matching address or create a new one |
| subpro.http.get.addresses | GET /services/v2/addresses | Retrieve a collection of addresses |
| subpro.http.get.products | GET /services/v2/products/{id} | Retrieve a single product by id |
| subpro.http.get.customers | GET /services/v2/customers/{id} | Retrieve a single customer by id |
| subpro.http.post.customer | POST /services/v2/customer | Create a new customer |
| subpro.http.post.customers | POST /services/v2/customers/{id} | Update a single customer, identified by customer id |
| subpro.http.get.token | GET /oauth/v2/token | Get an OAuth2 access token |
| subpro.http.get.paymentprofile | GET /services/v2/vault/paymentprofiles/{id} | Retrieve a single payment profile by id |
| subpro.http.post.paymentprofile.vault | POST /services/v2/vault/paymentprofile/external-vault | Create a new payment profile for an external vault |

For more details visit Subscribe Pro REST API Documentation:  
<https://feat-sfcc-cd75l4i-cdbrvib5kpzoa.us.platform.sh/docs/rest>

## Testing

### 

### Sandbox(es)

In order to test the application in a non-production environment, you will need to have a Subscribe Pro Sandbox.

Subscribe Pro provides "sandbox" accounts to all merchants.

* One Sandbox account is included with standard plans
* Additional sandbox accounts (or those which aren't associated with a paid plan) can be added but will incur a fee.

### Integration Testing

**Subscription Products**

The following are some test cases that can be completed to ensure subscription products are configured correctly.

1. User’s should see subscription options for products, on the product detail page, that have subscription’s enabled and a corresponding product has been configured on the Subscribe Pro Platform.
2. User’s should see an option for a one time purchase, if this option has been configured on Subscribe Pro.
3. User’s should see an option to purchase a product on a recurring basis, if this option has been configured on Subscribe Pro.
4. User’s should see the configured repeat Order options and be able to select the desired frequency of recurring orders.
5. User’s should be able to add a product, with subscription options, to the Basket and see the selected options present in the: Mini-Cart, Basket, Order Details (in the right rail of Checkout), the Order Confirmation Screen and in the Order Confirmation Email.
6. User’s should be able to change their subscription options while viewing the product in the Basket.
7. Subscription details should be transmitted to Subscribe Pro through a configured Job / Job Schedule.
8. Failed requests to create a subscription should be emailed to the configured user for further investigation.

**My Account**

The following are some test cases that can be completed to ensure that the My Account has been configured correctly.

1. User’s should be able to see a list of their Subscription orders in the “My Subscriptions” section of “My Account”.
2. User’s should see an option to update their Subscription Payment option upon changing their saved payment instrument in “My Account”.
3. User’s should see an option to update their Subscription Address option upon changing their saved address in “My Account”.
4. User’s should be able to update their profile information in “My Account” and those details are automatically transmitted to Subscribe Pro.

**Pixel / Widget Tracking**

The following are some test cases that can be completed to ensure that the Subscribe Pro Widget is included in all of the necessary locations.

1. The Javascript Widget should be executed and transmitting details to Subscribe Pro via Ajax on the following pages:
   1. My Account - Addresses
   2. My Account - Payment Instruments
   3. My Account - My Subscriptions
   4. Product Detail Page
   5. Product Search Results

# Operations, Maintenance

## Data Storage

Subscription selections and Subscribe Pro identifiers (customer, address, subscription) will be stored to the custom attributes of the Sales Force Commerce Cloud System Objects (Product, Product Line item, Order, Basket).

Customer information such as name, email, address and some payment information will be stored at Subscribe Pro for any customers that have purchased subscription products. This information will be used to process future recurring orders for the customer.

## Availability

Subscribe Pro hosted services and API's perform with an uptime of 99.9% reliability.

If external services are not available for products, subscription options will not be available for subscription products. The end user will still be able to purchase products as normal on the platform but subscription options will not be available.

If external services are not available for My Account, the end user will see a My Subscription page without any content.

If external services are not available for transmitting subscription order details, via a scheduled job, an email should be sent to the configured contact and they will need to manually process the subscription.

## Support

Support will be provided through the Subscribe Pro support system.

All requests should include: details of issue, email, phone, and URL.

Requests can be made at the following location:  
https://www.subscribepro.com/support/

**Technical Support Contact:**   
John Sparwasser   
john.sparwasser@subscribepro.com

# User Guide

## Roles, Responsibilities

*<LIST RECURRING TASKS THAT NEED TO BE FULFILLED BY CUSTOMER, MERCHANT TO RUN THE INTEGRATION, E.G. MANUAL FEED OF CATALOG DATA INTO 3RD PARTY SERVICE, IF APPLICABLE>*

## Business Manager

Subscribe Pro group is added to Custom Site Preference Groups (*Site > Site Preferences > Custom Preferences*). It contains following attributes:

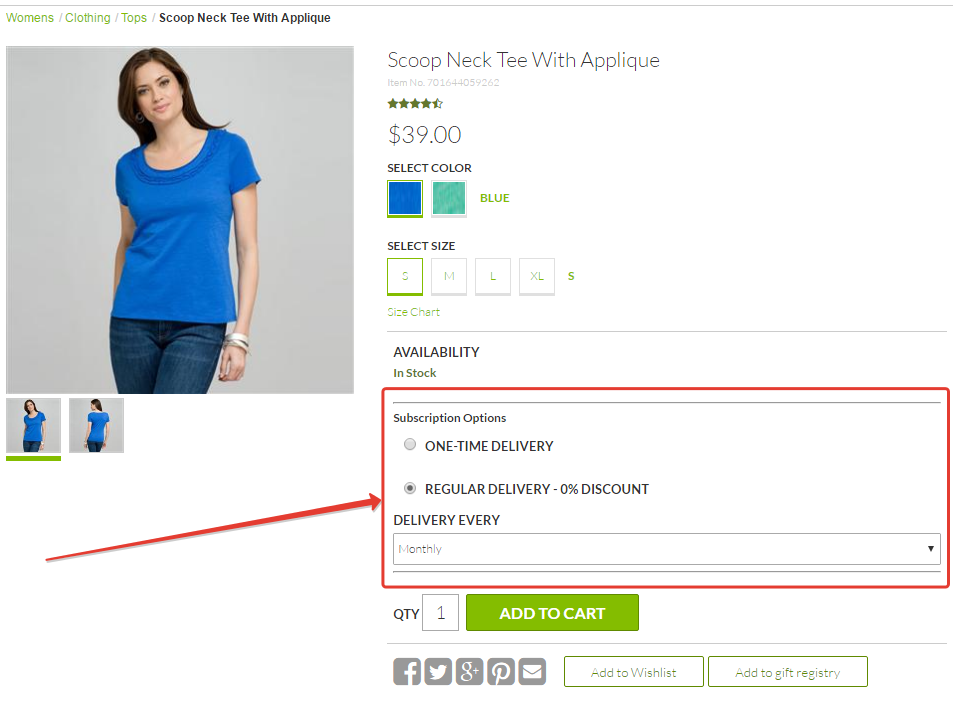
A screenshot of a cell phone

Description automatically generated

## Storefront Functionality

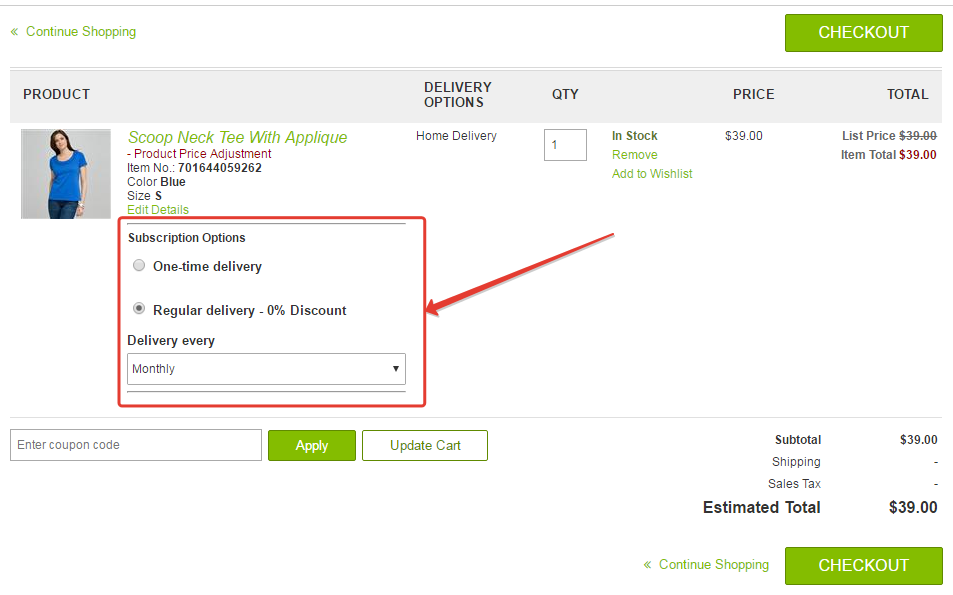
### Product Detail Page

When the product page template has been updated, you should be able to see subscription options for product: one time delivery or regular delivery and drop down with delivery intervals if regular delivery is available. This information will be shown only for products for which subscription is configured in SFCC Business Manager and in Subscribe Pro Business Manager.



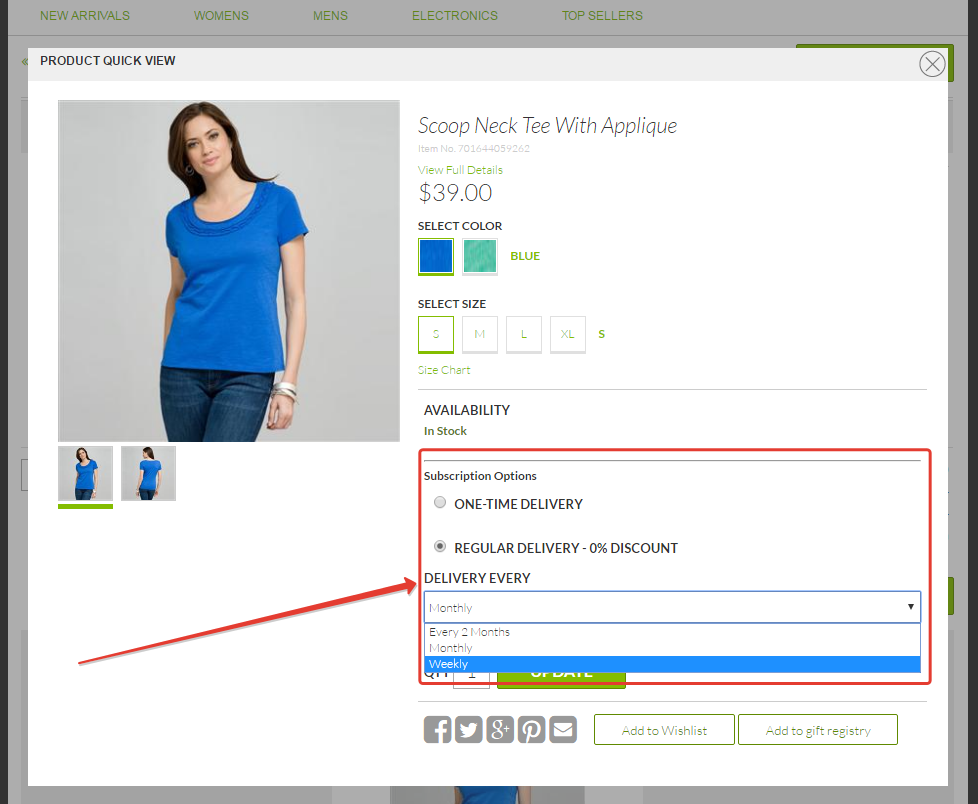
### Cart Page

If any product with subscription was added to basket, on cart page you will see the same subscription options as on PDP for such product line item. At this step subscription options can be changed. To apply changes you’ll need to click on ‘Update Cart’ button.



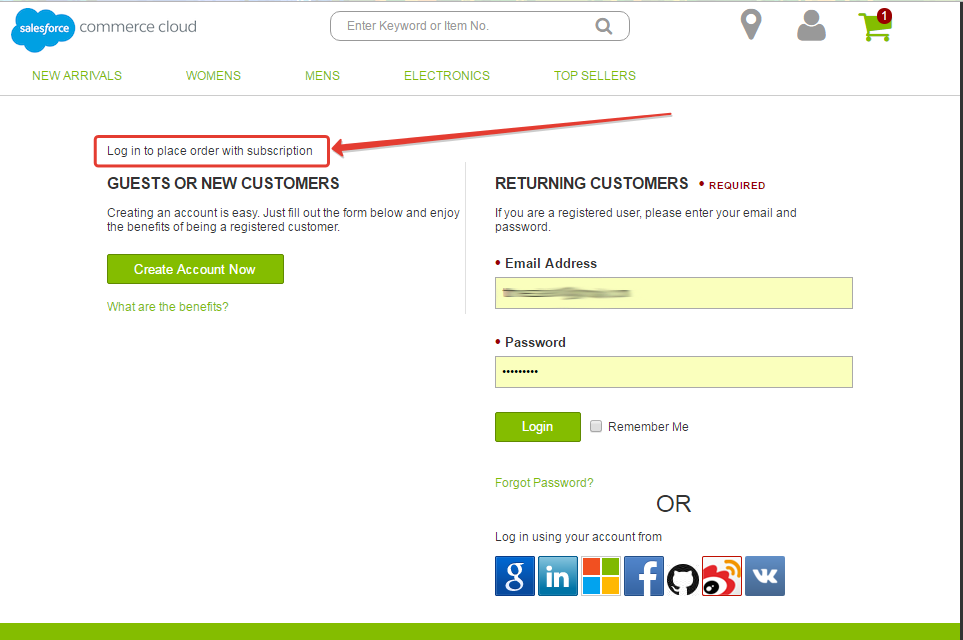
### Product Page (Edit Details Quickview)

While on cart page you can click on the ‘Edit Details’ link. It will show Quick View containing PDP, where you’ll see subscription options. They can be changed. To apply changes click on ‘Update’ button.



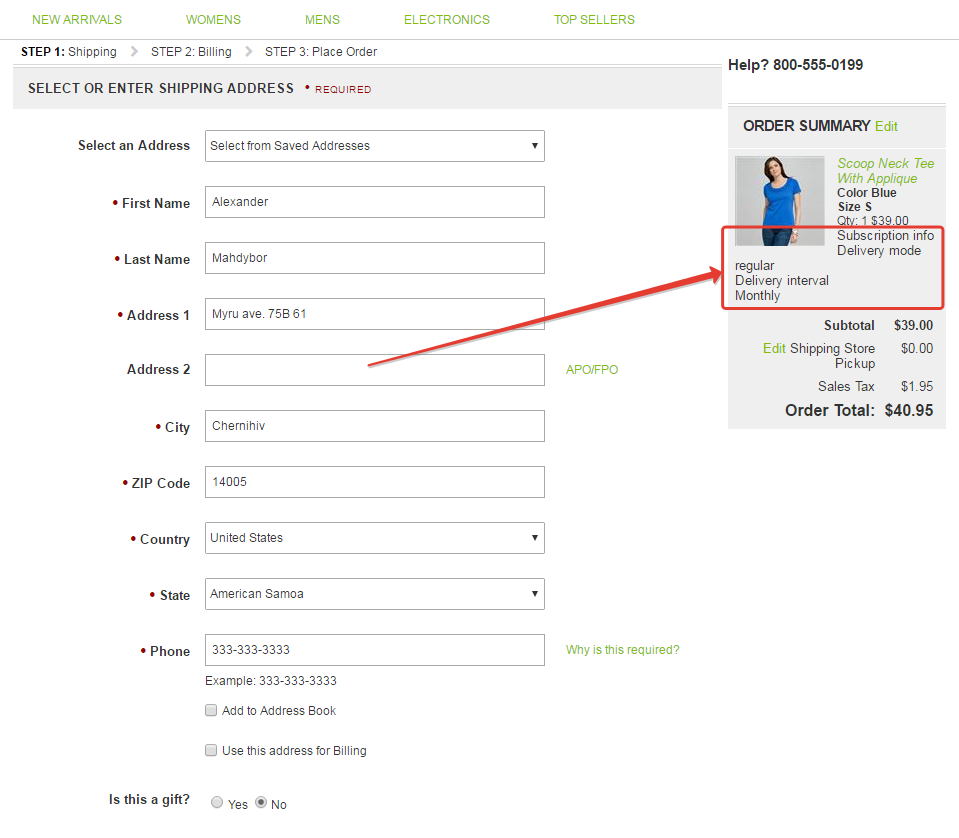
### Login Page

If customer has any product with regular subscription option selected, he/she must be logged in as a registered customer to place order with subscriptions. In other case customer will see error message informing that he/she should log in to proceed. ‘Checkout as Guest’ option is excluded.



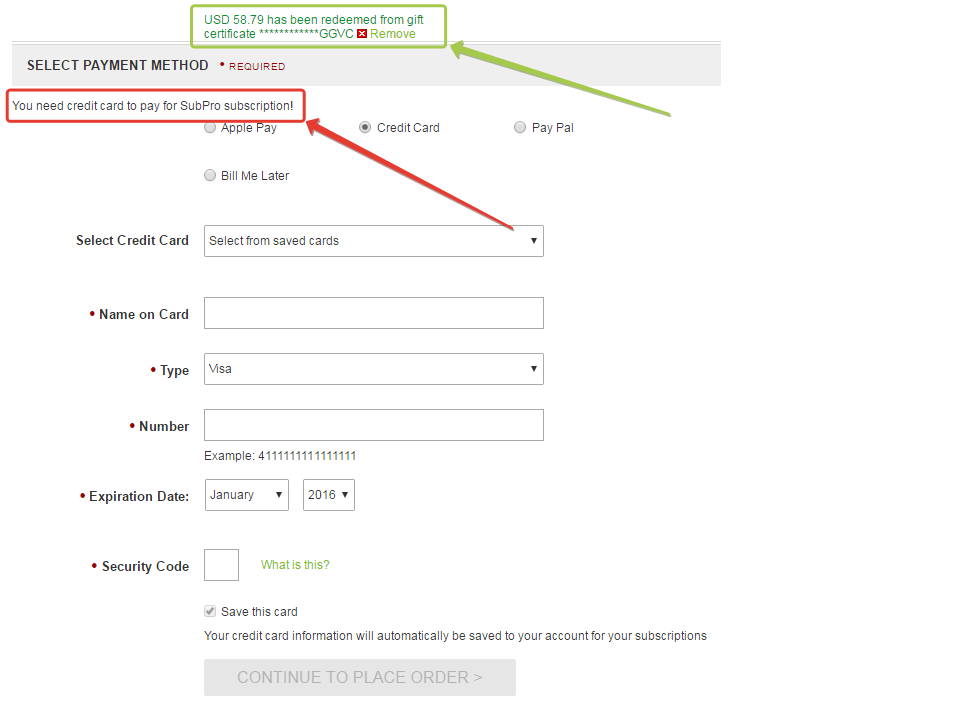
### Order Summary Right Rail and Minicart

For products with subscription selected options will be shown on Order Summary Right Rail block and on Minicart.

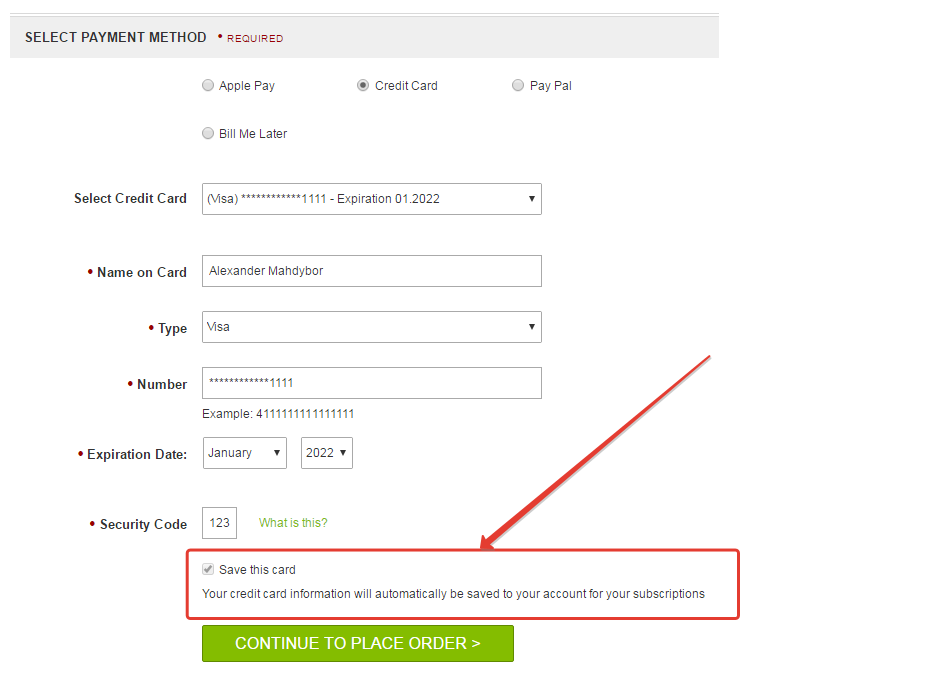


### Checkout Billing Page

To place order with a subscription the Credit Card payment method is required, even if customer has completely paid for order with alternative payment methods, without a credit card, using a gift card for example. The user must be required to enter a credit card number for the subscription. The message informing customer about it will be shown.

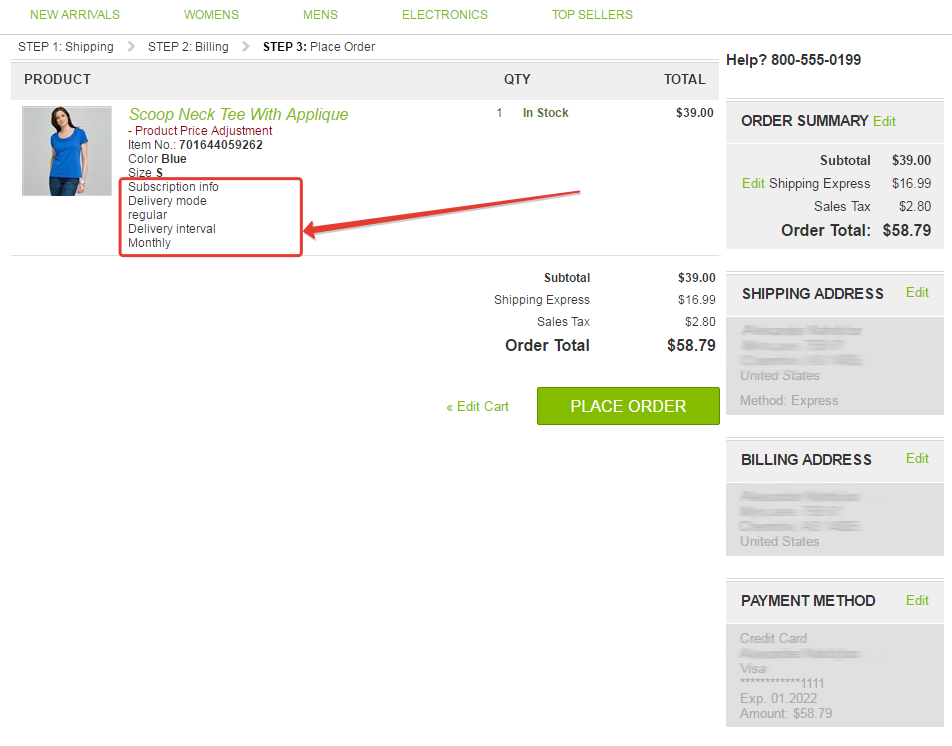


Also if credit card was selected its information will be force saved to customer’s account to pay for recurring subscriptions. Message, informing customer about it, will be shown.



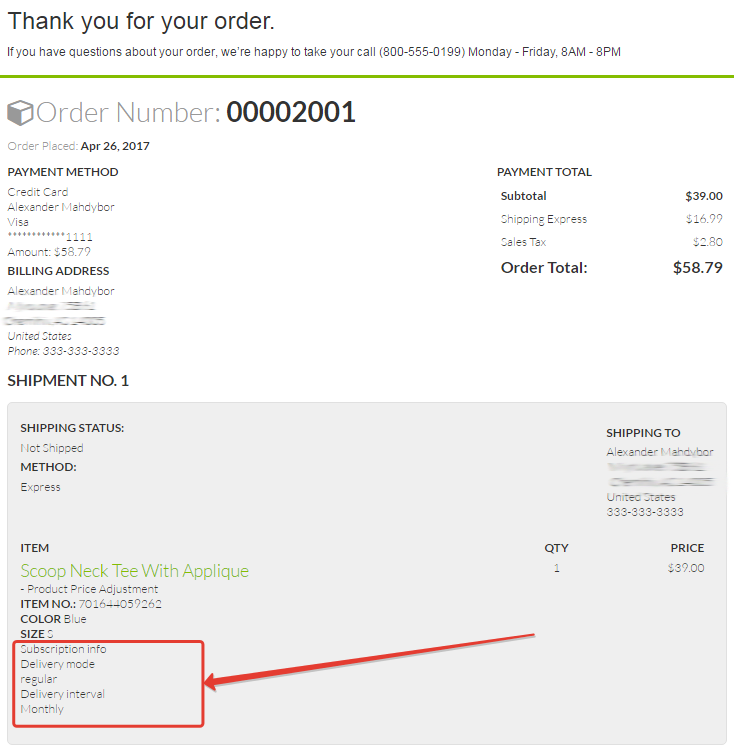
### Order Summary Page

When the order summary page template has been updated, you should be able to see selected subscription options for product in order.



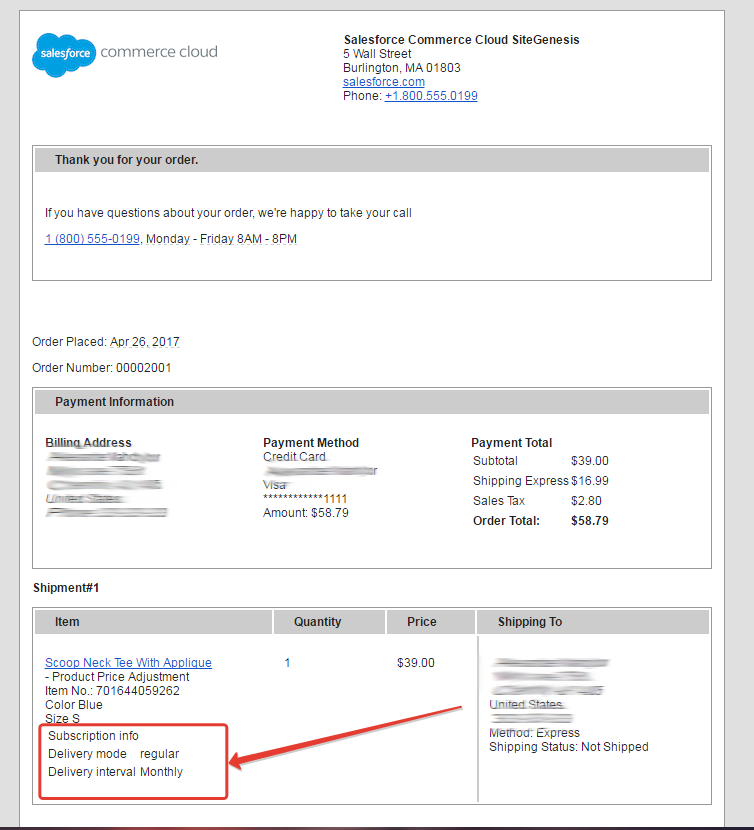
### Order Confirmation Page

When the order confirmation page template has been updated, you should be able to see selected subscription options for product in order.



### Order Confirmation Email

When the order confirmation email template has been updated, you should be able to see selected subscription options for product in order in Order Confirmation email sent to customer.



## Failover

If the Subscribe Pro platform is unreachable, with our base integration, customers will still be able to check out on the SFCC front-end with non-subscription items. The job that creates subscriptions in Subscribe Pro via our API would return errors, but once the connection is restored, it would retroactively create subscriptions missed during that time. If you customize the cartridge functionality to further rely on Subscribe Pro’s API, that functionality would become unusable if our system is unreachable.

Subscribe Pro has an excellent record of up-time. You can view our Status Page, which includes a history of all incidents affecting our service here: <https://status.subscribepro.com/>. We encourage all customers to subscribe to status page updates to remain informed of the status of any service interruptions.

# Release History

|  |  |
| --- | --- |
| **Version** | **Changes** |
| 17.1.0 | Initial release |
| 19.1.0 | Re-certification |
| 20.1.0 | Re-certification |