# Stripe Integration Guide

***Version 18.1.0***

## Table of Contents

# 1. Summary

The Stripe LINK Cartridge facilitates integration between a Commerce Cloud Storefront and Stripe Payment Services; including Stripe Elements, Sources, Webhooks, and Alternate Payment methods.  Usage of Sources via Stripe.js, ability to create charges, and optional integration with Stripe's Relay service for embedded eCommerce solutions on social channels.

Contracting with Stripe is required for live, production deployment of the cartridge. Though the cartridge can be installed and tested using a freely available Stripe test account at https://dashboard.stripe.com. Please contact your Stripe Implementation Consultant for help with taking your Stripe account live.

Stripe LINK Cartridge integration requires the following steps:

1. **Import System Object Metadata** for managing Stripe configurations and supporting data points
2. **Import Services** for enabling external interfaces with the Stripe API
3. **Apply Custom Code** to the SFCC Storefront
4. **Configure the Storefront Settings** to connect to Stripe services and enable LINK cartridge integration to the storefront

# 2. Component Overview

## 2.1 Functional Overview

### 2.1.1 *Stripe Elements and Sources*

Stripe Elements modifies the default Commerce Cloud Credit Card collection and processing by using Stripe.js, a JavaScript library, to securely tokenize credit card data. Payments are then processed using the tokenized data, not the raw credit card information.

During checkout, the cartridge will create a source for any new cards or alternate payment methods entered by customers. This data is transformed into a Stripe Source. At the point of purchase, the stored, tokenized data is used to generate a Stripe Charge. Registered Customers can manage (add, delete) Reusable payment methods in their storefront-connected Stripe Account for re-use in subsequent storefront purchases.

## 2.2 Use Cases

### 2.2.1 *Stripe.js Sources*

When customers enter credit card or other payment information on the storefront, the information is tokenized via Stripe.js in a client (browser)-to-Stripe interactions. Unmasked credit card data is therefore never sent to the Commerce Cloud servers.

### 2.2.2 *Stripe Charges*

System will create a Stripe Charge (authorize or capture, based on Business Manager configuration) from a successfully created and submitted Basket. All Stripe Charges are created against a Stripe payment Source.

### 2.2.3 *AVS Auto-Fail Transactions*

Site administrators can select a variety of AVS statuses for which an Order should be auto-failed. If the Stripe Charge returns any of the selected statuses for either address_line1_check or address_zip_check the Order will be auto-failed and the Stripe Charge reversed. Note that these settings can also be managed on the Stripe Dashboard.

## 2.3 Limitations, Constraints

Stripe offers a number of standard services that are not supported by the cartridge. These include support for Subscriptions, Plans, and Coupons.

The Google Product Feed support included in the cartridge only supports a limited number of product attributes, including variation attributes for Size and Color. Additional attribute support requires customizing the product feed output.

The included RELAY OCAPI configurations are included as examples only.  A RELAY implementation will require additional configuration and testing along with the Stripe team.

## 2.4 Compatibility

Available since Commerce Cloud Platform Release 16.8, Site Genesis 103.0.0

The cartridge is available for installations on storefronts that support both Pipeline and Controller SiteGenesis implemenations.

## 2.5 Privacy, Payment

No unmasked credit card data is stored within Commerce Cloud. The cartridge tokenizes all payment data via direct client-to-Stripe communications and obscures any sensitive credit card data before it arrives on the Commerce Cloud servers. Similarly, all credit card data that is retrieved by Commerce Cloud from the Stripe servers is also masked and/or tokenized.

# 3. Implementation Guide

## 3.1 Setup

The Stripe LINK Cartridge contains several cartridges that are required for full functionality. Additionally, Pipeline and Controller support is broken out into two separate cartridges, thereby facilitating the installation and use of one or the other models.

Import all three cartridges into UX studio and associate them with a Server Connection.

### 3.1.1 *Cartridges*

#### 3.1.1.1 int_stripe

int_stripe contains all the script logic, templates, resource properties, and front-end code required for the base Stripe storefront integration.

1. js
    a. stripe.js
2. scripts
    a. hooks/afterPostPaymentInstrument.js
    b. hooks/afterSetShippingAddress.js
    c. hooks/authorize.js
    d. hooks/authorizeCreditCard.js
    e. job/processPayments.js
    f. payment/processor/STRIPE_CREDIT.js
    g. payment/processor/STRIPE.js
    h. service/stripe.ds
    i. service/stripeInit.ds
    j. webhook/webhook.ds
    k. hooks.json
    l. stripeHelper.ds
3. scss
    a. _stripe.scss
4. static
    a. stripe/jquery.payment.js
5. templates
    a. account/payment/makefdefault.isml
    b. account/payment/makedefaultlink.isml
    c. account/payment/stripeelements.isml
    d. account/payment/stripeinputs.isml
    e. checkout/billing/stripe_creditcard_fields.isml
    f. checkout/billing/stripe_paymentmethods.isml
    g. checkout/billing/stripecreditcardjson.isml
    h. feed/displayproductfeed.isml
    i. mail/orderfailed.isml
    j. mail/orderreceived.isml
    k. stripe/headerinclude.isml
    l. stripe/footerinclude.isml
    m. stripe/paymenterrors.isml
    n. stripe/paymentmethods.isml
    o. stripe/paymentoptions.isml
    p. stripe/summaryreview.isml
    q. util/apple.isml

r. webhook/renderResponse.isml
s. resources/account.properties
t. resources/checkout.properties
u. resources/order.properties

#### 3.1.1.2 int_stripe_controllers

int_stripe_controllers contains all the logic for managing the Stripe integration controller logic via Commerce Cloud's Controllers model. int_stripe_controllers should be used in favor of int_stripe_pipelines if the storefront will be using the latest version of SiteGenesis.

1. controllers
    a. Stripe.js

#### 3.1.1.3 int_stripe_pipelines

int_stripe_pipelines contains all the logic for managing the Stripe integration controller logic via Commerce Cloud's Pipelines controller model. int_stripe_pipelines can be used in case the integration requires the use of Pipelines for any reason.

Additional note: int_stripe_pipelines is required for the custom Product Feed job. The Commerce Cloud Platform, as of version 16.8, requires Pipeline code for custom scheduled jobs.

1. pipelines
    a. STRIPE_CREDIT.xml
    b. Stripe.xml
2. scripts
    a. feed/GenerateProductFeed.ds
    b. pipelinescripts/AddCard.ds
    c. pipelinescripts/AuthorizePayment.ds
    d. pipelinescripts/DeleteCard.ds
    e. pipelinescripts/DisplayProductFeeds.ds
    f. pipelinescripts/FetchCards.ds
    g. pipelinescripts/GetCustomerCreditCard.ds
    h. pipelinescripts/IsStripeEnabled.ds
    i. pipelinescripts/MakeDefault.ds
    j. pipelinescripts/RefundCharge.ds

#### 3.1.1.4 Metadata

1. stripe_jobs.xml
2. stripe_metadata.xml
3. stripe_custom-objects.xml
4. stripe_services.xml

## 3.2 Configuration

### 3.2.1 *Assign Cartridges to Site(s)*

#### 3.2.1.1 Site Cartridge Assignment

1. Navigate to Administration > Sites > Manage Sites
2. Click on the Site Name for the Storefront Site that will add Stripe Functionality

3. Select the “Settings” tab
4. Add “*int_stripe*” to the **Cartridges:** path, separating each cartridge in the list with “:”
    a. For example, “*app_storefront_controllers:app_storefront_core:int_stripe_controllers:int_stripe*”
5. Add either the “*int_stripe_controllers*” OR “*int_stripe_pipelines*” to the cartridge path
    a. Note that if both cartridges are added to the cartridge path then “*int_stripe_controllers*” code is executed when Stripe URLs are requested
6. Repeat steps 2 – 5 for each Storefront Site where Stripe will be implemented

#### 3.2.1.2 Business Manager Cartridge Assignment

Stripe cartridges need to be assigned to the Business Manager Site only if the implementation makes use of the Product Feed for use with Stripe Relay functionality. Note also that, at the time of this writing, the Pipelines cartridge–“int_stripe_pipelines”–is required for the custom Job Feed schedule rather than “int_stripe_controllers”.

- Navigate to Administration > Sites > Manage Sites
- Click on the Business Manager Site > “Manage the Business Manager site.” Link
- Add “*int_stripe:in_stripe_pipelines*” to the **Cartridges:** path, separating each cartridge in the list with “:”

### 3.2.2 *Import System and Custom Object Definitions*

1. Login to Business Manager and navigate to **Administration > Site Development > Import & Export**
2. Click “Upload” in the **Import & Export Files** section
3. In the **Upload Import Files > Upload File** section, select *stripe_metadata.xml* file from the metadata/ folder of the LINK Cartridge and click “Upload”
4. Return to the **Administration > Site Development > Import & Export** page
5. Click on “Import” in the **Import & Export > Meta Data** section
6. Select the radio button next to the *stripe_metadata.xml* file and click “Next >>”
7. Once the XML validation completes, click “Import”
8. After the import has completed, a Success status will display in the Status section
9. Repeat this process with the stripe_custom-objects.xml file.

### 3.2.3 *Import Services*

1. Login to Business Manager and navigate to **Administration > Operations > Import & Export**
2. Click “Upload” in the **Import & Export Files** section
3. In the **Upload Import Files > Upload File** section, select *stripe_services.xml* file from the metadata/ folder of the LINK Cartridge and click “Upload”
4. Return to the **Administration > Operations > Import & Export** page
5. Click on “Import” in the **Import & Export > Services** section
6. Select the radio button next to the *stripe_services.xml* file and click “Next >>”
7. Once the XML validation completes, click “Import”
8. After the import has completed, a Success status will display in the Status section

Imported Services:

- addCard
- updateCard
- deleteCard
- fetchCustomerCards
- authorizePayment (Stripe Charge)
- refundCharge

- createCustomer
- retrieveCustomer
- updateCustomer

### 3.2.4 *Import Custom Job Schedule*

1. Login to Business Manager and navigate to **Administration > Operations > Import & Export**
2. Click “Upload” in the **Import & Export Files** section
3. In the **Upload Import Files > Upload File** section, select *stripe_jobs.xml* file from the metadata/ folder of the LINK Cartridge and click “Upload”
4. Return to the **Administration > Operations > Import & Export** page
5. Click on “Import” in the **Import & Export > Job Schedules** section
6. Select the radio button next to the *stripe_jobs.xml* file and click “Next >>”
7. Once the XML validation completes, click “Import”
8. After the import has completed, a Success status will display in the Status section

### 3.2.5 *Custom Site Preferences*

Update the **Merchant Tools > Site Preferences > Custom Site Preferences > Stripe Configurations** with Site specific values.

1. **Stripe Secret API Key**
    a. Can be obtained through the Stripe Dashboard (https://dashboard.stripe.com/account/apikeys)
2. **Stripe Publishable API Key**
    a. Find along with Stripe Secret API Key
3. **Capture Funds on Stripe Charge**
    a. Default value: true (Yes)
    b. Set to false (No) to instead Authorize Stripe Charges
4. **AVS Address Fail Statuses**
    a. Select one or more AVS statuses. If the Stripe Address check returns any of the selected statuses, the Order will auto-fail and reverse the Stripe Charge
5. **AVS Zip Fail Statuses**
    a. Select one or more AVS statuses. If the Stripe Zip check returns any of the selected statuses, the Order will auto-fail and reverse the Stripe Charge
6. **Enable Stripe Relay Payment Authorization**
    a. If enabled, Orders placed via OCAPI will post Credit Card Charges against the Stripe Account associated with the secret and publishable API keys
7. **Product Feed Directory**
    a. The directory within the IMPEX WebDAV folder where the Product Feed .tsv file will be saved
8. **Product Feed File Name**
    a. File name for the actual Product Feed File
9. **Product Feed Default Brand Name**
    a. Update this field to force all brand attributes in the feed to use a single brand name. Leaving this field empty will result in falling back to Product.brand attribute value.
10. **Allowed APM Methods**
    a. Update this field, per site locale, to indicate which alternate payment methods are enabled for each locale. Enumeration of allowed Payment Methods from the Stripe API. See more here: https://stripe.com/docs/sources { "default": [ "p24", "eps", "sepa_debit", "ideal", "sofort", "bitcoin", "alipay", "bancontact", "giropay" ], "en_UK": [ "p24", "eps" ], "de_AT": [ "sofort", "ideal" ] }
11. **Stripe Card Element CSS Style**
    a. Enter the CSS styling that the Card element button should inherit to fit within the overall storefront styles. Style Configuration for Stripe Elements e.g, {"base":

{"fontFamily": "Arial, sans-serif","fontSize": "14px","color": "#C1C7CD"},"invalid": {"color": "red" } }

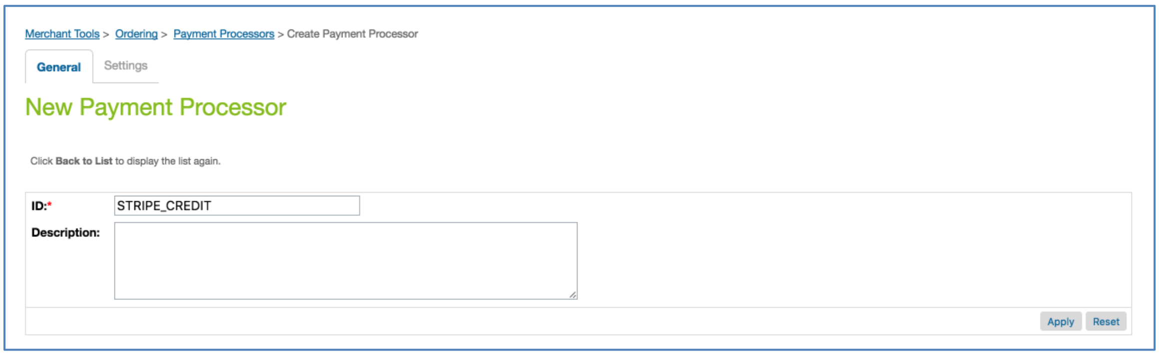
12. **Stripe API URL** - https://js.stripe.com/v3/
13. **Stripe Payment Request Button Style**
    a. For the payment request button, select the limited CSS styling that the button should display with. See [https://stripe.com/docs/stripe-js/elements/payment-request-button#styling-the-element](https://stripe.com/docs/stripe-js/elements/payment-request-button#styling-the-element)
14. **ApplePay Verification String**
    i. Enter the Apple verification string provided from the Stripe dashboard.
    ii. This is a one time enablement. The Stripe console will proxy the Apple Pay for Web verification String upon setup. This will need to be configured into the sandbox if the Payment Request Button will be used as a form of payment on the storefront.
15. **Stripe Payment Request Button payment Method**
    i. Configure the SFCC payment method id to use when the payment request button is selected (or keep the default). This should be set to the stripe payment request button payment method id. This is so that the cartridge can determine if it is enabled or not without imposing restrictions on the naming of the payment method id.
16. **Stripe Webhook Signing Secret**
    i. Enter the webhook signing secret provided by the stripe dashboard. Stripe will sign webhook calls and pass a validation to SFCC. SFCC will validate the contents of the message via this key.
17. **Stripe Currency Code** - USED ONLY FOR TESTING. This variable used only for testing purposes to simulate a different currency/locale on non-production instances when it's not empty. (e.g, eur or usd). This can be removed as part of implementation if desired but there is code to prevent the setting from being used on a production instance.
18. **Stripe allowed Webhook Statuses**
    i. Configure the allowed statuses for Webhooks to respond to (this need not be changed unless additional customizations are being made).
19. **Country Code** (Stripe Payment Request Button) - Country Code e.g, US. This will be the default country code for the Payment Request Button. Customization may be needed on a multi country single site in order to dynamically pass the country code rather than the site preference (if needed). https://stripe.com/docs/stripe-js/elements/payment-request-button#create-payment-request-instance

### 3.2.6 *Payment Settings Updates*

#### 3.2.6.1 Add New Payment Processors

There are two payment processors used in the Stripe cartridge. “STRIPE_CREDIT” is used for credit card handling while “STRIPE” is used for the asynchronous payment model (Bank transfers, GiroPay, etc).

If using Stripe credt cards, Navigate to **Merchant Tools > Ordering > Payment Processors** and click the "New" button. In the new window set the ID attribute to value "**STRIPE_CREDIT**" and click "Apply".

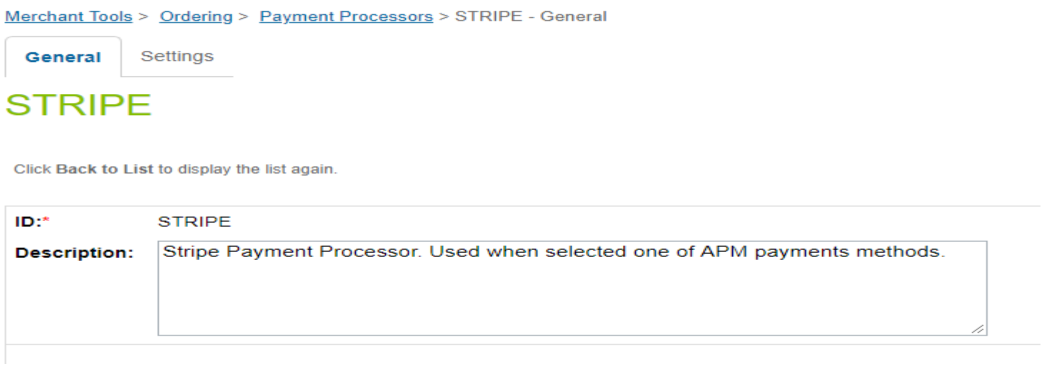

Merchant Tools > Ordering > Payment Processors > Create Payment Processor

New Payment Processor

Click **Back to List** to display the list again.

| | |
|---|---|
| **ID:*** | STRIPE_CREDIT |
| **Description:** | |

If using APM methods, again, click the "New" button. In the new window set the ID attribute to value "**STRIPE**" and click "Apply".  This payment method is for the non-credit card (APM methods):


Merchant Tools > Ordering > Payment Processors > STRIPE - General

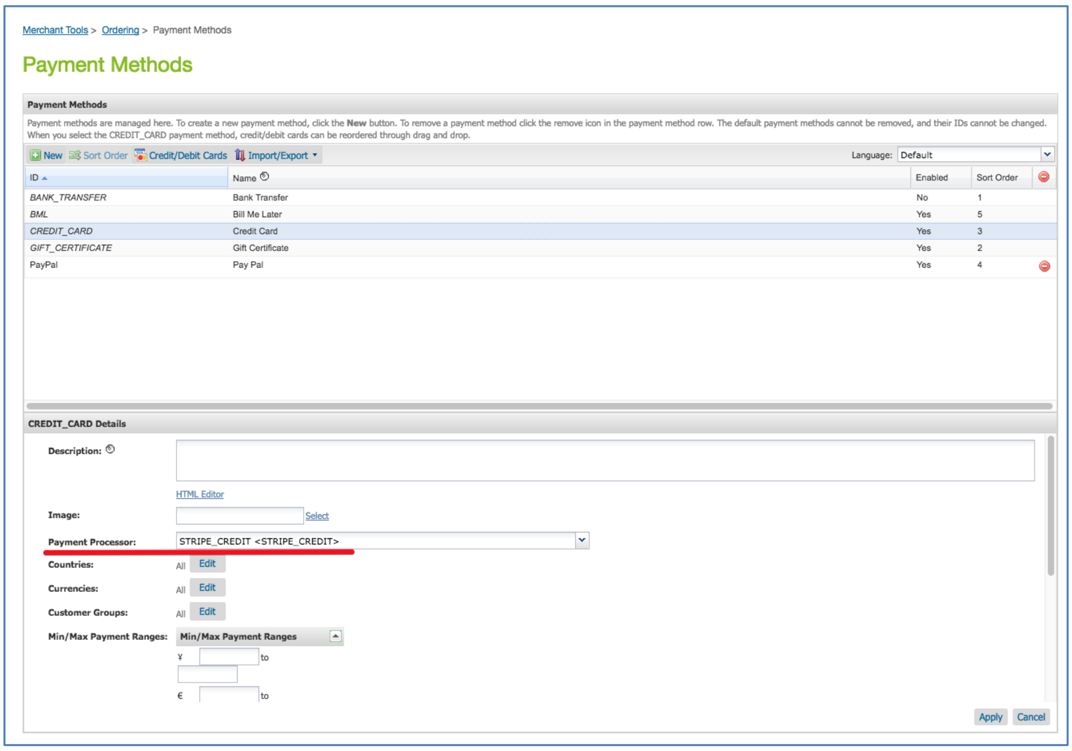
STRIPE

Click **Back to List** to display the list again.

| | |
|---|---|
| **ID:*** | STRIPE |
| **Description:** | Stripe Payment Processor. Used when selected one of APM payments methods. |

### 3.2.6.2 Update Payment Methods

Navigate to Merchant Tools > Ordering > Payment Methods, click on the CREDIT_CARD payment method and select the STRIPE_CREDIT payment processor in dropdown under the CREDIT_CARD Details section


Merchant Tools > Ordering > Payment Methods

Payment Methods

**Payment Methods**

Payment methods are managed here. To create a new payment method, click the **New** button. To remove a payment method click the remove icon in the payment method row. The default payment methods cannot be removed, and their IDs cannot be changed. When you select the CREDIT_CARD payment method, credit/debit cards can be reordered through drag and drop.

Language: Default

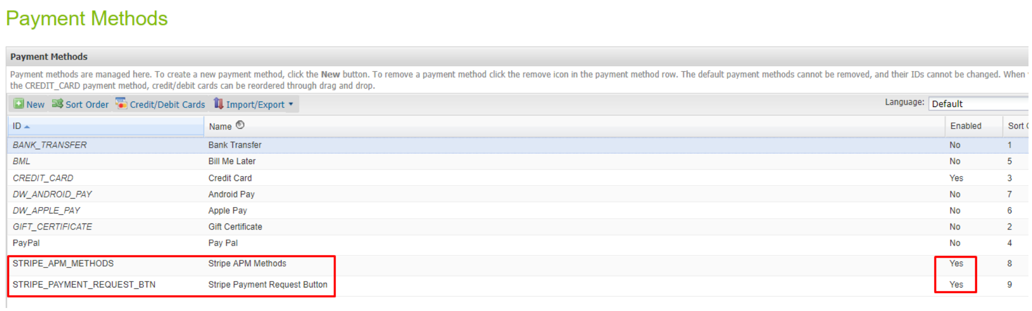
| ID | Name | Enabled | Sort Order |
|---|---|---|---|
| *BANK_TRANSFER* | Bank Transfer | No | 1 |
| *BML* | Bill Me Later | Yes | 5 |
| *CREDIT_CARD* | Credit Card | Yes | 3 |
| *GIFT_CERTIFICATE* | Gift Certificate | Yes | 2 |
| PayPal | Pay Pal | Yes | 4 |

**CREDIT_CARD Details**

| | |
|---|---|
| **Description:** | |
| | HTML Editor |
| **Image:** | Select |
| **Payment Processor:** | STRIPE_CREDIT <STRIPE_CREDIT> |
| **Countries:** | All Edit |
| **Currencies:** | All Edit |
| **Customer Groups:** | All Edit |
| **Min/Max Payment Ranges:** | Min/Max Payment Ranges |
| | ¥ to |
| | € to |

If using APM payment methods and/or the Payment Request Button then enable the desired payment methods as well:  The STRIPE_APM_METHODS will provide the ability to include all of the supported Stripe methods.  See https://stripe.com/payments/payment-methods-guide

To utilize the Stripe Payment Request Button, enable the “STRIPE_PAYMENT_REQUEST_BTN” payment method.        See https://stripe.com/docs/stripe-js/elements/payment-request-button


Payment Methods

**Payment Methods**

Payment methods are managed here. To create a new payment method, click the **New** button. To remove a payment method click the remove icon in the payment method row. The default payment methods cannot be removed, and their IDs cannot be changed. When the CREDIT_CARD payment method, credit/debit cards can be reordered through drag and drop.

New | Sort Order | Credit/Debit Cards | Import/Export | Language: Default

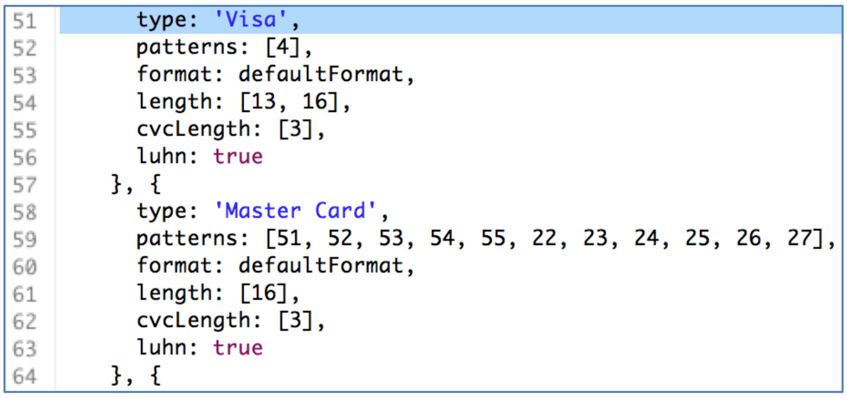
| ID | Name | Enabled | Sort |
|---|---|---|---|
| *BANK_TRANSFER* | Bank Transfer | No | 1 |
| *BML* | Bill Me Later | No | 5 |
| *CREDIT_CARD* | Credit Card | Yes | 3 |
| *DW_ANDROID_PAY* | Android Pay | No | 7 |
| *DW_APPLE_PAY* | Apple Pay | No | 6 |
| *GIFT_CERTIFICATE* | Gift Certificate | No | 2 |
| PayPal | Pay Pal | No | 4 |
| STRIPE_APM_METHODS | Stripe APM Methods | Yes | 8 |
| STRIPE_PAYMENT_REQUEST_BTN | Stripe Payment Request Button | Yes | 9 |

## 3.2.6.3 Credit Card Support

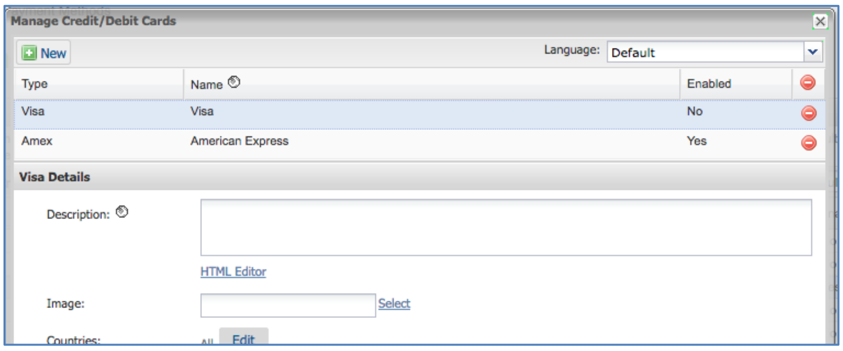
The base LINK Cartridge code contains support for all credit cards supported by Stripe. Note that the list of allowed cards on the storefront is still limited by the Credit/Debit Cards list in Business Manager (*Merchant Tool         s > Ordering > Payment Methods > Credit/Debit Cards*).

If your storefront requires updates to how the OMS handles credit card ‘type’ values, you need to add additional card types, and/or update how card names are displayed on the storefront, update the references to CC types in the following files accordingly:

1. int_stripe/cartridge/static/default/stripe/jquery.payment.js
    a. The value for the ‘type’ property in the $.payment.cards object must match the ‘Type’ value for the card in Business Manager



```
      type: 'Visa',
      patterns: [4],
      format: defaultFormat,
      length: [13, 16],
      cvcLength: [3],
      luhn: true
    }, {
      type: 'Master Card',
      patterns: [51, 52, 53, 54, 55, 22, 23, 24, 25, 26, 27],
      format: defaultFormat,
      length: [16],
      cvcLength: [3],
      luhn: true
    }, {
```


**Manage Credit/Debit Cards**

New | Language: Default

| Type | Name | Enabled |
|---|---|---|
| Visa | Visa | No |
| Amex | American Express | Yes |

**Visa Details**

Description:

HTML Editor

Image: Select

Countries: All Edit

2. int_stripe/cartridge/templates/resources/checkout.properties

a. Use this file to define the Storefront display of the card name. If introducing new cards, add an additional definition to match the ‘type’ and ‘Type’ values defined in jquery.payment.js and Business Manager as described above.

### 3.2.7 *Enable/Disable Stripe Payments*

To enable/disable Stripe Payments on the Storefront, update the Payment Processor for Payment Method “CREDIT_CARD” to use Payment Processor “STRIPE_CREDIT”.

If any other Payment Processor is defined for “CREDIT_CARD”, Stripe is considered disabled.

Please refer to section 3.2.6.2 above for managing the Payment Methods > Payment Processor setting.

### 3.2.8 *Product Feed Setup*

#### 3.2.8.1 BM Configurations

Three configurations described earlier–Product Feed Directory, Product Feed File Name, Product Feed Default Brand Name–control some basic settings for the product feed file name and brand name setting. Note that these can all be left blank and the feed job will proceed using the default settings.

#### 3.2.8.2 System Object Extensions

Import the metadata for metadata/stripe_metadata.xml file as described previously.

#### 3.2.8.3 Custom Scheduled Job

Import the Job Schedule for GenerateStripeProductFeed using the metadata/stripe_jobs.xml file as described previously.

The job will need to be customized to match the Site needs. This includes enabling the job, assigning an appropriate Site, and scheduling based on the implementation’s requirements.

If using APM Payment methods (non credit card) you will need to import the job schedule for “stripe-process-webhook-notifications” using the metadata/stripe_jobs.xml file as described previously.

The job will need to be customized to match the Site needs. This includes enabling the job, assigning an appropriate Site, and scheduling based on the implementation’s requirements.

#### 3.2.8.4 Dynamic Mapping for URL

For importing the Feed into Stripe, add a Dynamic Mapping to make sure Stripe can handle the requested .tsv filetype.

1. Navigate to **Merchant Tools > Site URLs > Dynamic Mapping**
2. Add a new rule: **/productfeed.tsv p,,,Stripe-DisplayProductFeed,,,**

The file can now be imported into the Stripe Dashboard. Visit http://dashboard.stripe.com, click Relay > Products. Then click the 'Manage Feed' link to open up the modal. Enter the DWRE URL for the feed into the Feed URL and format box, selecting Google as the feed type.

## 3.3 Custom Code

Make the following updates to the Storefront Code. Examples provided are based on SiteGenesis version (103.1.11). The initial integration effort should take from ½ to 2 days, based on a SiteGenesis installation. Below are the customizations made to SiteGenesis code. Please refer to the folder "SGTouchPoints" to get the data if you prefer to copy/paste.

### 3.3.1 *Controllers*

Controller updates are only required if integrating with Controllers instead of using Pipelines

**Controller: Home.js**

app_storefront_controllers/cartridge/controllers/Home.js

- Update controller Home.js as below by adding applePay() function preferably towards the bottom of the file.

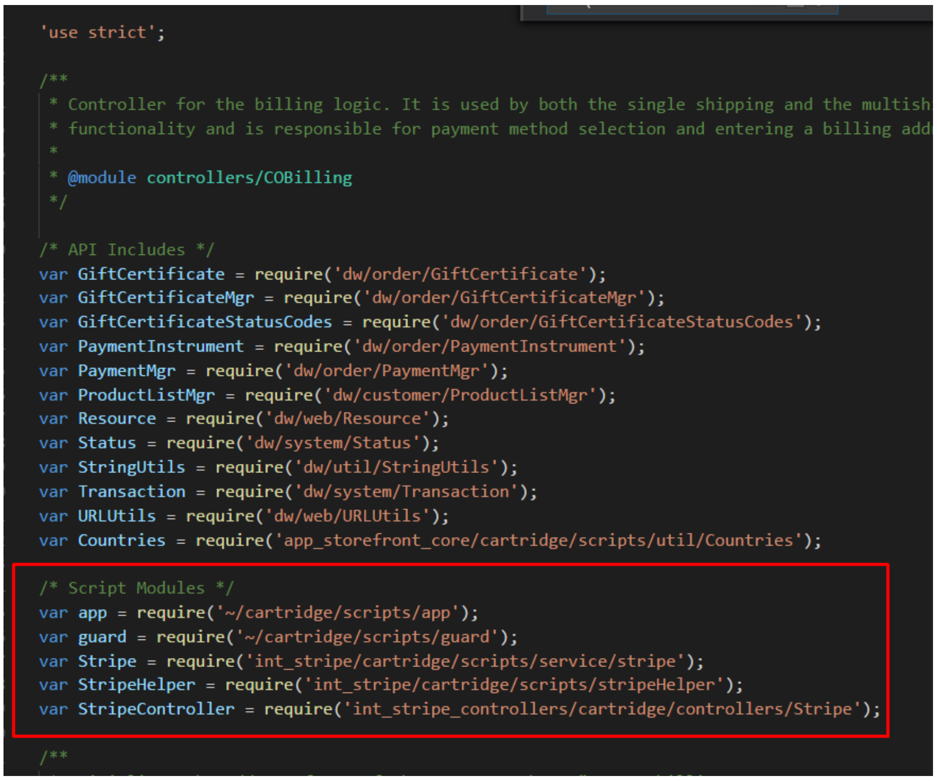
```
function applepay() {
            app.getView().render('util/apple');
}

exports.Apple = guard.ensure(['get'], applepay);
```

**Controller: COBilling.js**

app_storefront_controllers/cartridge/controllers/COBilling.js

- Add the following to the top of the file:

```
var Stripe = require('int_stripe/cartridge/scripts/service/stripe');
var StripeHelper = require('int_stripe/cartridge/scripts/stripeHelper');
var StripeController = require('int_stripe_controllers/cartridge/controllers/Stripe');
```

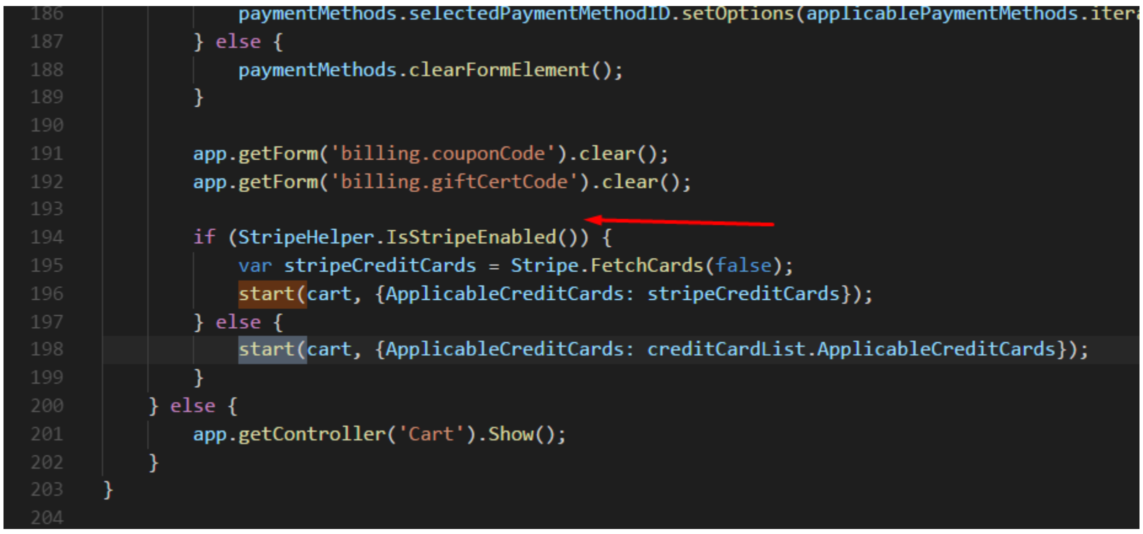


```
'use strict';

/**
 * Controller for the billing logic. It is used by both the single shipping and the multish
 * functionality and is responsible for payment method selection and entering a billing add
 *
 * @module controllers/COBilling
 */

/* API Includes */
var GiftCertificate = require('dw/order/GiftCertificate');
var GiftCertificateMgr = require('dw/order/GiftCertificateMgr');
var GiftCertificateStatusCodes = require('dw/order/GiftCertificateStatusCodes');
var PaymentInstrument = require('dw/order/PaymentInstrument');
var PaymentMgr = require('dw/order/PaymentMgr');
var ProductListMgr = require('dw/customer/ProductListMgr');
var Resource = require('dw/web/Resource');
var Status = require('dw/system/Status');
var StringUtils = require('dw/util/StringUtils');
var Transaction = require('dw/system/Transaction');
var URLUtils = require('dw/web/URLUtils');
var Countries = require('app_storefront_core/cartridge/scripts/util/Countries');

/* Script Modules */
var app = require('~/cartridge/scripts/app');
var guard = require('~/cartridge/scripts/guard');
var Stripe = require('int_stripe/cartridge/scripts/service/stripe');
var StripeHelper = require('int_stripe/cartridge/scripts/stripeHelper');
var StripeController = require('int_stripe_controllers/cartridge/controllers/Stripe');

/**
```

- Add the following if statement to function publicStart() **after** the line
  **app.getForm('billing.giftCertCode').clear();**

Note: The existing call to *start(cart, {ApplicableCreditCards: reditCardList.ApplicableCreditCards})* *is being updated.*

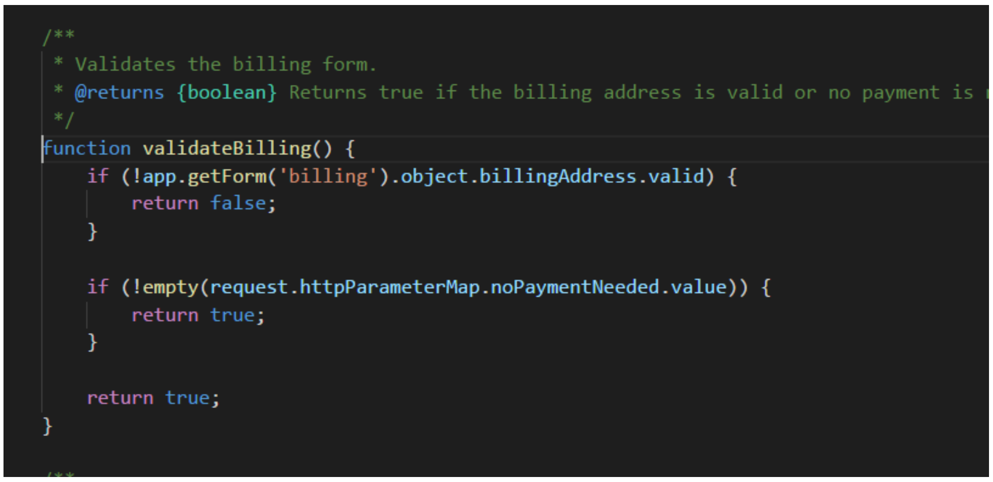
```
if (StripeHelper.IsStripeEnabled()) {
        var stripeCreditCards = Stripe.FetchCards(false);
        start(cart, {ApplicableCreditCards: stripeCreditCards});
} else {
        start(cart, {ApplicableCreditCards: creditCardList.ApplicableCreditCards});
}
```



```
            paymentMethods.selectedPaymentMethodID.setOptions(applicablePaymentMethods.itera
        } else {
            paymentMethods.clearFormElement();
        }

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

        if (StripeHelper.IsStripeEnabled()) {
            var stripeCreditCards = Stripe.FetchCards(false);
            start(cart, {ApplicableCreditCards: stripeCreditCards});
        } else {
            start(cart, {ApplicableCreditCards: creditCardList.ApplicableCreditCards});
        }
    } else {
        app.getController('Cart').Show();
    }
}
```

- Update function validateBilling() by removing the if statement below:

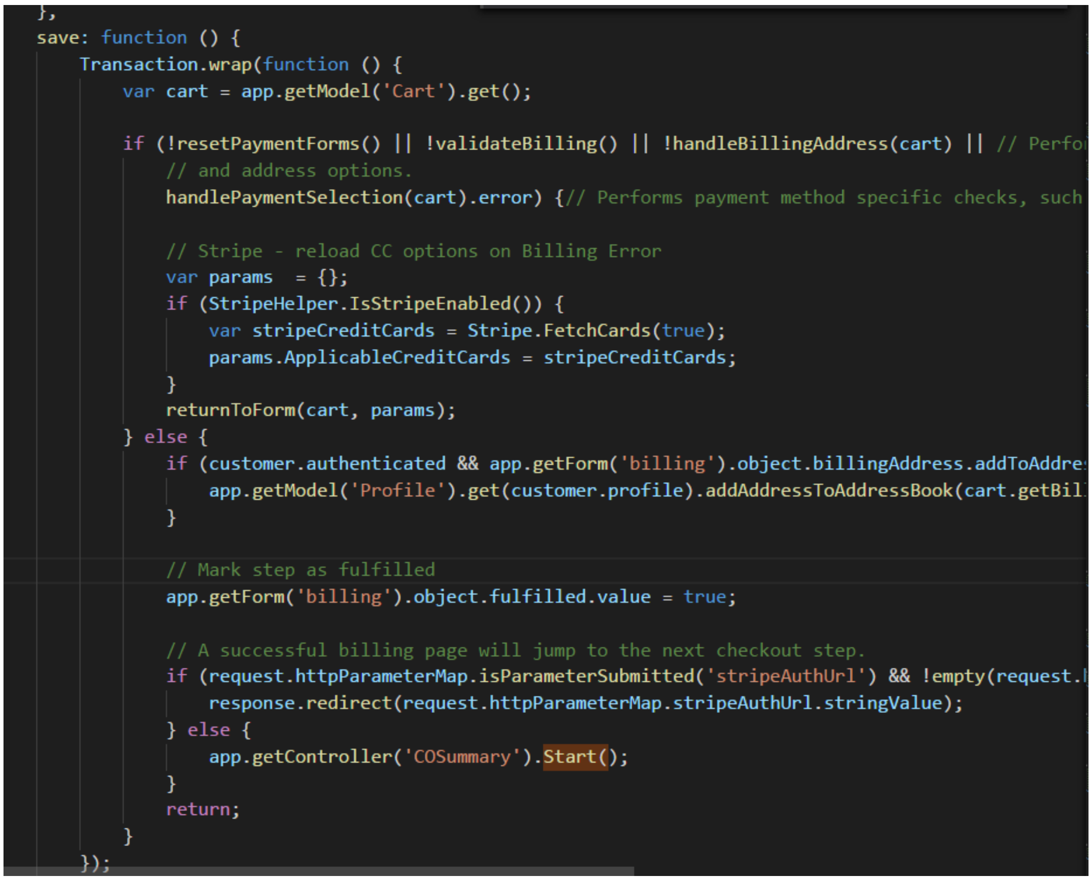
```
    if (!empty(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value)
&&
app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(Payme
ntInstrument.METHOD_CREDIT_CARD)) {
        if (!app.getForm('billing').object.valid) {
            return false;
        }
    }
```



```
/**
 * Validates the billing form.
 * @returns {boolean} Returns true if the billing address is valid or no payment is
 */
function validateBilling() {
    if (!app.getForm('billing').object.billingAddress.valid) {
        return false;
    }

    if (!empty(request.httpParameterMap.noPaymentNeeded.value)) {
        return true;
    }

    return true;
}
```

- Update billing() / save() function as below:

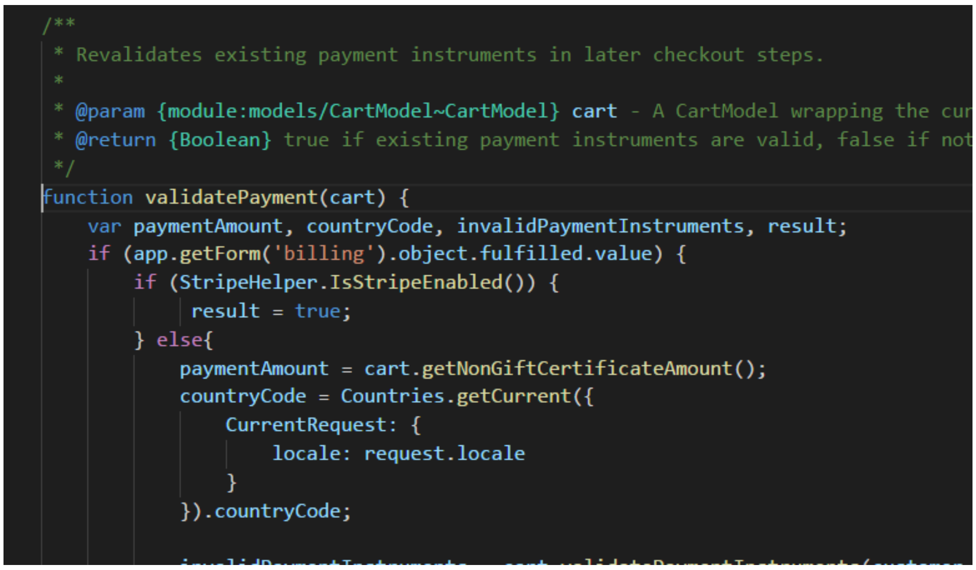
```
              Transaction.wrap(function () {
              var cart = app.getModel('Cart').get();

              if (!resetPaymentForms() || !validateBilling() || !handleBillingAddress(cart) || // Performs validation
steps, based upon the entered billing address
                            // and address options.
                                      handlePaymentSelection(cart).error) {// Performs payment method specific
checks, such as credit card verification.

                                                       // Stripe - reload CC options on Billing Error
                                                       var params  = {};
                                                       if (StripeHelper.IsStripeEnabled()) {
                                                                     var stripeCreditCards = Stripe.FetchCards(true);
                                                                     params.ApplicableCreditCards = stripeCreditCards;
                                                       }
                                                       returnToForm(cart, params);
              } else {
                            if (customer.authenticated &&
app.getForm('billing').object.billingAddress.addToAddressBook.value) {

              app.getModel('Profile').get(customer.profile).addAddressToAddressBook(cart.getBillingAddress())
;
                            }

                            // Mark step as fulfilled
                            app.getForm('billing').object.fulfilled.value = true;

                            // A successful billing page will jump to the next checkout step.
                            if (request.httpParameterMap.isParameterSubmitted('stripeAuthUrl') &&
!empty(request.httpParameterMap.stripeAuthUrl.stringValue)) {
                                          response.redirect(request.httpParameterMap.stripeAuthUrl.stringValue);
                            } else {
                                          app.getController('COSummary').Start();
                            }
                            return;
              }
          });
```



```
},
save: function () {
    Transaction.wrap(function () {
        var cart = app.getModel('Cart').get();

        if (!resetPaymentForms() || !validateBilling() || !handleBillingAddress(cart) || // Perfo
            // and address options.
            handlePaymentSelection(cart).error) {// Performs payment method specific checks, such

            // Stripe - reload CC options on Billing Error
            var params  = {};
            if (StripeHelper.IsStripeEnabled()) {
                var stripeCreditCards = Stripe.FetchCards(true);
                params.ApplicableCreditCards = stripeCreditCards;
            }
            returnToForm(cart, params);
        } else {
            if (customer.authenticated && app.getForm('billing').object.billingAddress.addToAddre
                app.getModel('Profile').get(customer.profile).addAddressToAddressBook(cart.getBil
            }

            // Mark step as fulfilled
            app.getForm('billing').object.fulfilled.value = true;

            // A successful billing page will jump to the next checkout step.
            if (request.httpParameterMap.isParameterSubmitted('stripeAuthUrl') && !empty(request.
                response.redirect(request.httpParameterMap.stripeAuthUrl.stringValue);
            } else {
                app.getController('COSummary').Start();
            }
            return;
        }
    });
```

- Update validatePayment() function by adding the if statement below this line - if (app.getForm('billing').object.fulfilled.value).

*if (StripeHelper.IsStripeEnabled()) {*
    *result = true;*
*} else{*
    *……*
        *}*



```
/**
 * Revalidates existing payment instruments in later checkout steps.
 *
 * @param {module:models/CartModel~CartModel} cart - A CartModel wrapping the cur
 * @return {Boolean} true if existing payment instruments are valid, false if not
 */
function validatePayment(cart) {
    var paymentAmount, countryCode, invalidPaymentInstruments, result;
    if (app.getForm('billing').object.fulfilled.value) {
        if (StripeHelper.IsStripeEnabled()) {
            result = true;
        } else{
            paymentAmount = cart.getNonGiftCertificateAmount();
            countryCode = Countries.getCurrent({
                CurrentRequest: {
                    locale: request.locale
                }
            }).countryCode;

            invalidPaymentInstruments = cart.validatePaymentInstruments(customer
```

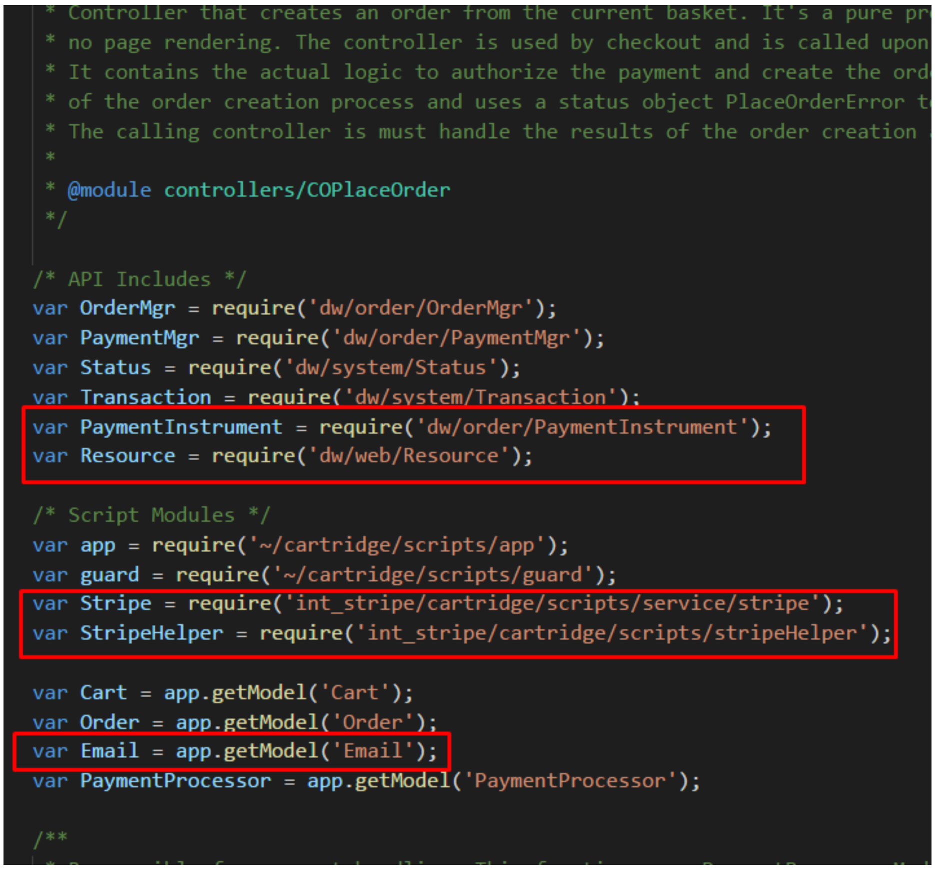
**Controller: COPlaceOrder.js**

app_storefront_controllers/cartridge/controllers/COPlaceOrder.js

- Add the following to the top of the file:
  *var PaymentInstrument = require('dw/order/PaymentInstrument');*
  *var Resource = require('dw/web/Resource');*

- Declare the following in the script modules section
  *var Email = app.getModel('Email');*

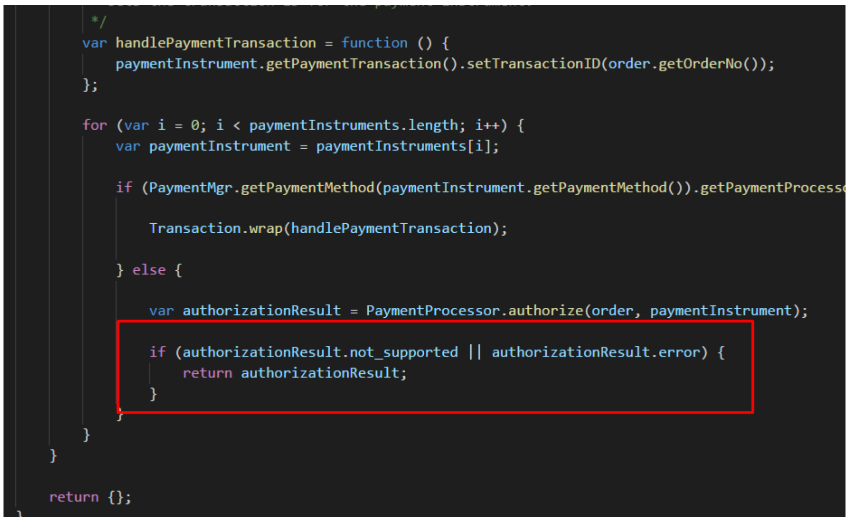
- Add the following script module includes right after the “guard” include:
  *var Stripe = require('int_stripe/cartridge/scripts/service/stripe');*
  *var StripeHelper = require('int_stripe/cartridge/scripts/stripeHelper');*



```
 * Controller that creates an order from the current basket. It's a pure pr
 * no page rendering. The controller is used by checkout and is called upon
 * It contains the actual logic to authorize the payment and create the ord
 * of the order creation process and uses a status object PlaceOrderError t
 * The calling controller is must handle the results of the order creation 
 *
 * @module controllers/COPlaceOrder
 */

/* API Includes */
var OrderMgr = require('dw/order/OrderMgr');
var PaymentMgr = require('dw/order/PaymentMgr');
var Status = require('dw/system/Status');
var Transaction = require('dw/system/Transaction');
var PaymentInstrument = require('dw/order/PaymentInstrument');
var Resource = require('dw/web/Resource');

/* Script Modules */
var app = require('~/cartridge/scripts/app');
var guard = require('~/cartridge/scripts/guard');
var Stripe = require('int_stripe/cartridge/scripts/service/stripe');
var StripeHelper = require('int_stripe/cartridge/scripts/stripeHelper');

var Cart = app.getModel('Cart');
var Order = app.getModel('Order');
var Email = app.getModel('Email');
var PaymentProcessor = app.getModel('PaymentProcessor');

/**
```

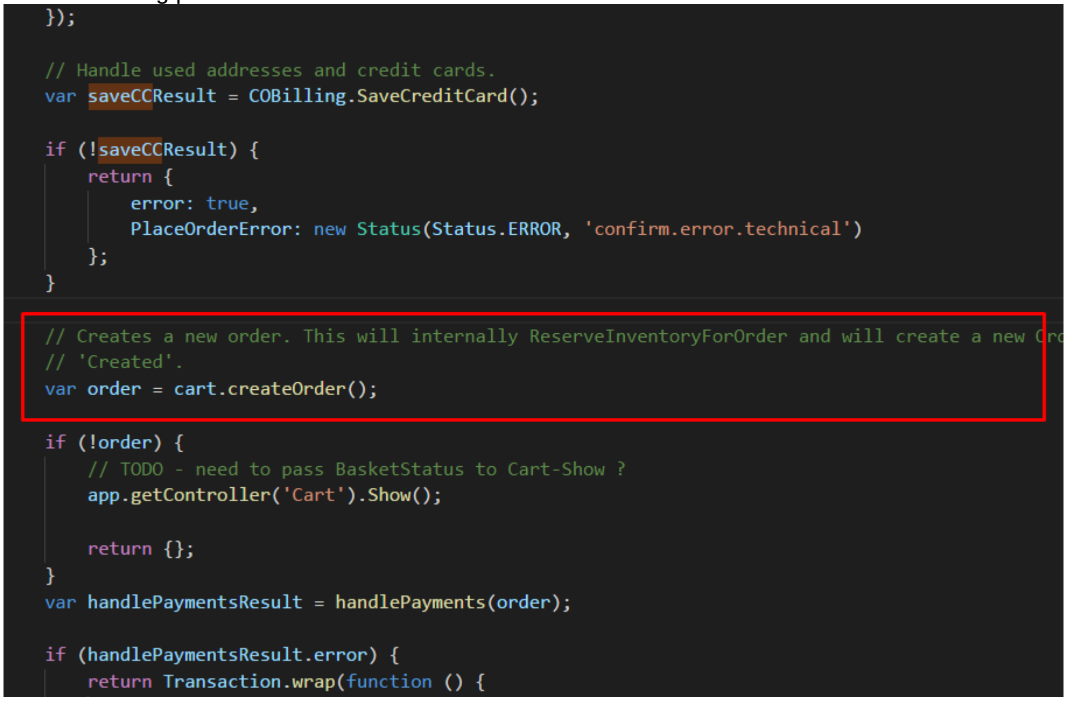
- Update handlePayments() function by returning the authorizationResult object instead of error true json.

```
if (authorizationResult.not_supported || authorizationResult.error) {
        return authorizationResult;
    }
```



```
         */
        var handlePaymentTransaction = function () {
            paymentInstrument.getPaymentTransaction().setTransactionID(order.getOrderNo());
        };

        for (var i = 0; i < paymentInstruments.length; i++) {
            var paymentInstrument = paymentInstruments[i];

            if (PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcesso

                Transaction.wrap(handlePaymentTransaction);

            } else {

                var authorizationResult = PaymentProcessor.authorize(order, paymentInstrument);

                if (authorizationResult.not_supported || authorizationResult.error) {
                    return authorizationResult;
                }
            }
        }
    }

    return {};
```

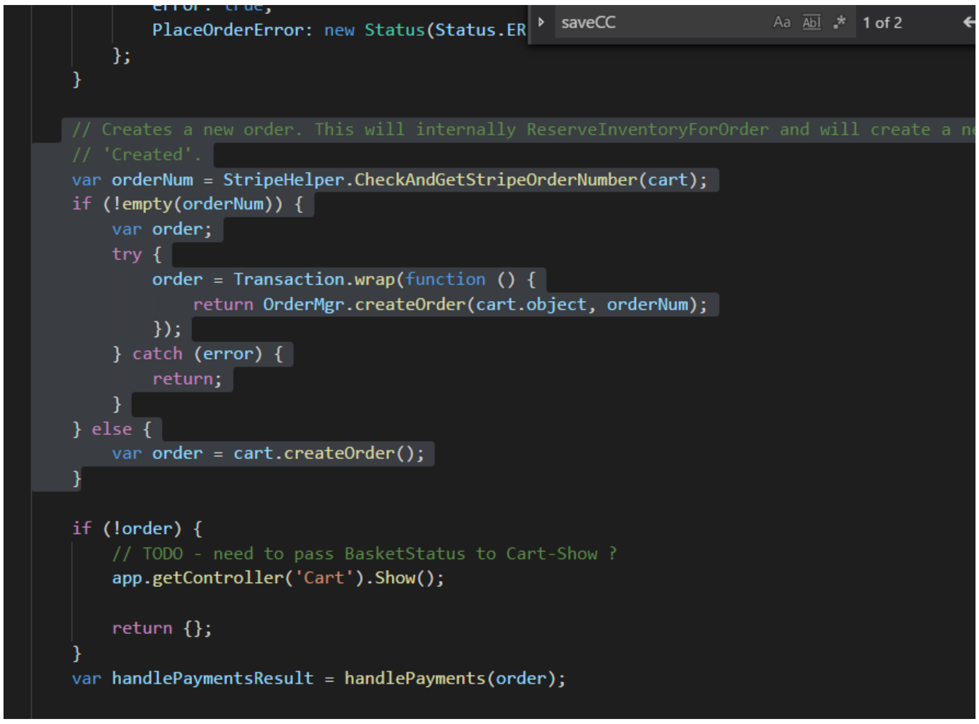
- Update start() function as below right after this line “var saveCCResult = COBilling.SaveCreditCard();” until the closing parenthesis.



```
    });

    // Handle used addresses and credit cards.
    var saveCCResult = COBilling.SaveCreditCard();

    if (!saveCCResult) {
        return {
            error: true,
            PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
        };
    }

    // Creates a new order. This will internally ReserveInventoryForOrder and will create a new Or
    // 'Created'.
    var order = cart.createOrder();

    if (!order) {
        // TODO - need to pass BasketStatus to Cart-Show ?
        app.getController('Cart').Show();

        return {};
    }
    var handlePaymentsResult = handlePayments(order);

    if (handlePaymentsResult.error) {
        return Transaction.wrap(function () {
```

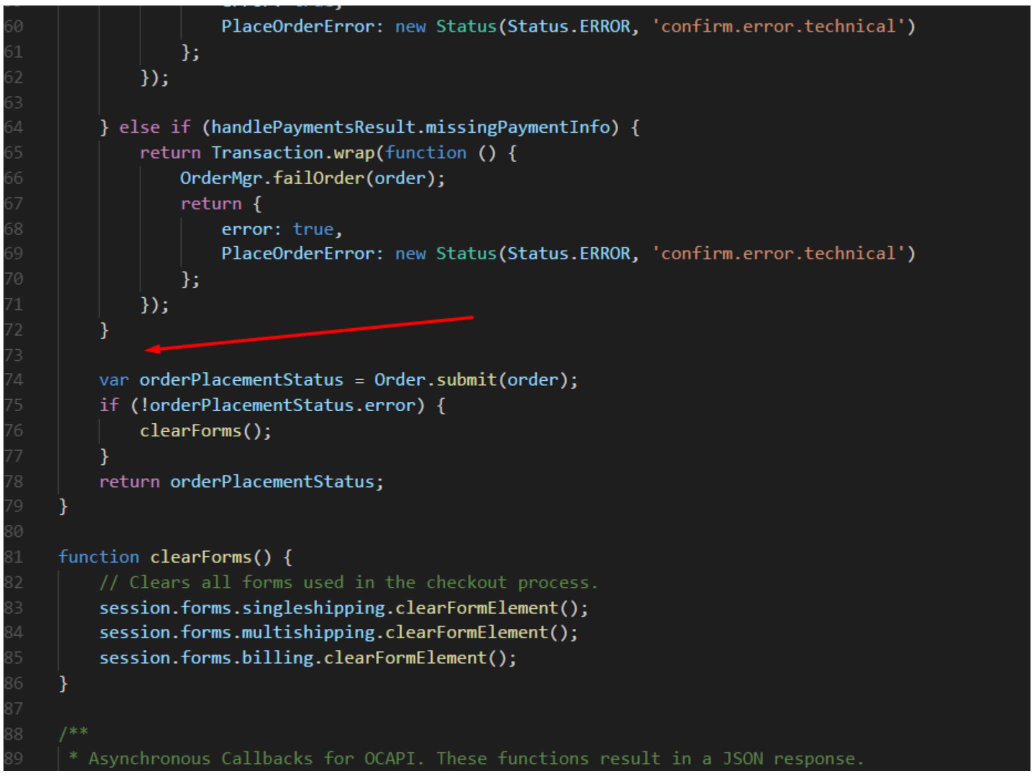
- 
- With the following code:

```
// Creates a new order. This will internally ReserveInventoryForOrder and will create a new Order with status
  // 'Created'.
  var orderNum = StripeHelper.CheckAndGetStripeOrderNumber(cart);
  if (!empty(orderNum)) {
            var order;
      try {
          order = Transaction.wrap(function () {
              return OrderMgr.createOrder(cart.object, orderNum);
          });
      } catch (error) {
          return;
      }
  } else {
      var order = cart.createOrder();
  }
```



```
            PlaceOrderError: new Status(Status.ER
        };
    }

    // Creates a new order. This will internally ReserveInventoryForOrder and will create a ne
    // 'Created'.
    var orderNum = StripeHelper.CheckAndGetStripeOrderNumber(cart);
    if (!empty(orderNum)) {
        var order;
        try {
            order = Transaction.wrap(function () {
                return OrderMgr.createOrder(cart.object, orderNum);
            });
        } catch (error) {
            return;
        }
    } else {
        var order = cart.createOrder();
    }

    if (!order) {
        // TODO - need to pass BasketStatus to Cart-Show ?
        app.getController('Cart').Show();

        return {};
    }
    var handlePaymentsResult = handlePayments(order);
```

*Later down the method after the "else if" block for missing payment info is closed:*



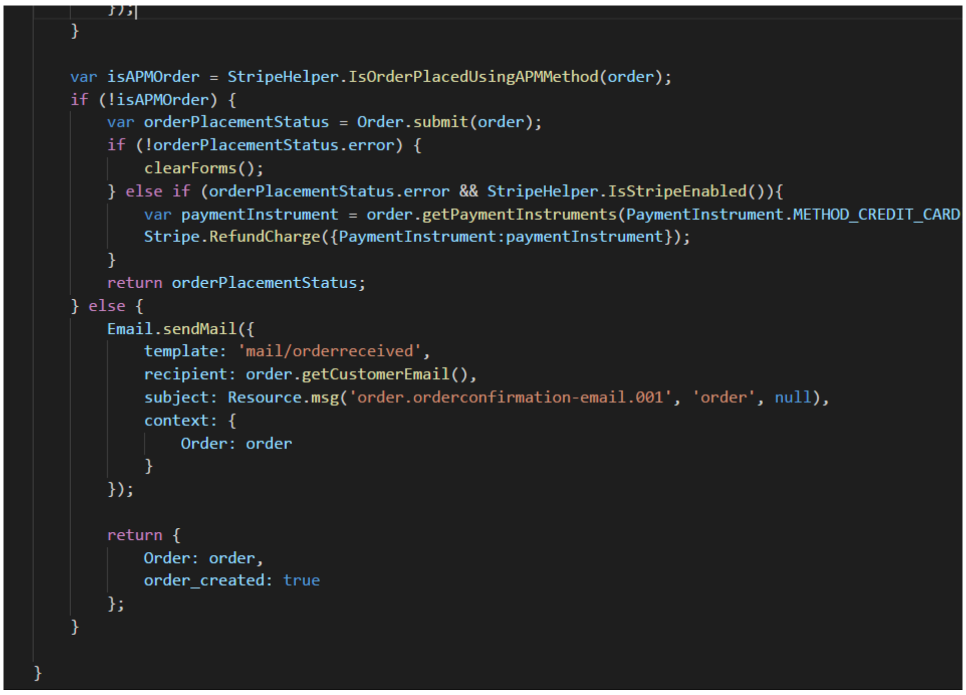
```
                PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
            };
        });

    } else if (handlePaymentsResult.missingPaymentInfo) {
        return Transaction.wrap(function () {
            OrderMgr.failOrder(order);
            return {
                error: true,
                PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
            };
        });
    }

    var orderPlacementStatus = Order.submit(order);
    if (!orderPlacementStatus.error) {
        clearForms();
    }
    return orderPlacementStatus;
}

function clearForms() {
    // Clears all forms used in the checkout process.
    session.forms.singleshipping.clearFormElement();
    session.forms.multishipping.clearFormElement();
    session.forms.billing.clearFormElement();
}

/**
 * Asynchronous Callbacks for OCAPI. These functions result in a JSON response.
```

Add the following code:

```
var isAPMOrder = StripeHelper.IsOrderPlacedUsingAPMMethod(order);
    if (!isAPMOrder) {
            var orderPlacementStatus = Order.submit(order);
            if (!orderPlacementStatus.error) {
                clearForms();
                } else if (orderPlacementStatus.error &&
StripeHelper.IsStripeEnabled()){
```

```
                    var paymentInstrument =
order.getPaymentInstruments(PaymentInstrument.METHOD_CREDIT_CARD)[0];
                    Stripe.RefundCharge({PaymentInstrument:paymentInstrument});
                }
            return orderPlacementStatus;
    } else {
        Email.sendMail({
            template: 'mail/orderreceived',
            recipient: order.getCustomerEmail(),
            subject: Resource.msg('order.orderconfirmation-email.001', 'order', null),
            context: {
                Order: order
            }
        });

            return {
                Order: order,
                order_created: true
            };
    }
```

**Resulting in:**



```
        });
    }

    var isAPMOrder = StripeHelper.IsOrderPlacedUsingAPMMethod(order);
    if (!isAPMOrder) {
        var orderPlacementStatus = Order.submit(order);
        if (!orderPlacementStatus.error) {
            clearForms();
        } else if (orderPlacementStatus.error && StripeHelper.IsStripeEnabled()){
            var paymentInstrument = order.getPaymentInstruments(PaymentInstrument.METHOD_CREDIT_CARD
            Stripe.RefundCharge({PaymentInstrument:paymentInstrument});
        }
        return orderPlacementStatus;
    } else {
        Email.sendMail({
            template: 'mail/orderreceived',
            recipient: order.getCustomerEmail(),
            subject: Resource.msg('order.orderconfirmation-email.001', 'order', null),
            context: {
                Order: order
            }
        });

        return {
            Order: order,
            order_created: true
        };
    }

}
```

**Controller: PaymentInstruments.js**

app_storefront_controllers/cartridge/controllers/PaymentInstruments.js

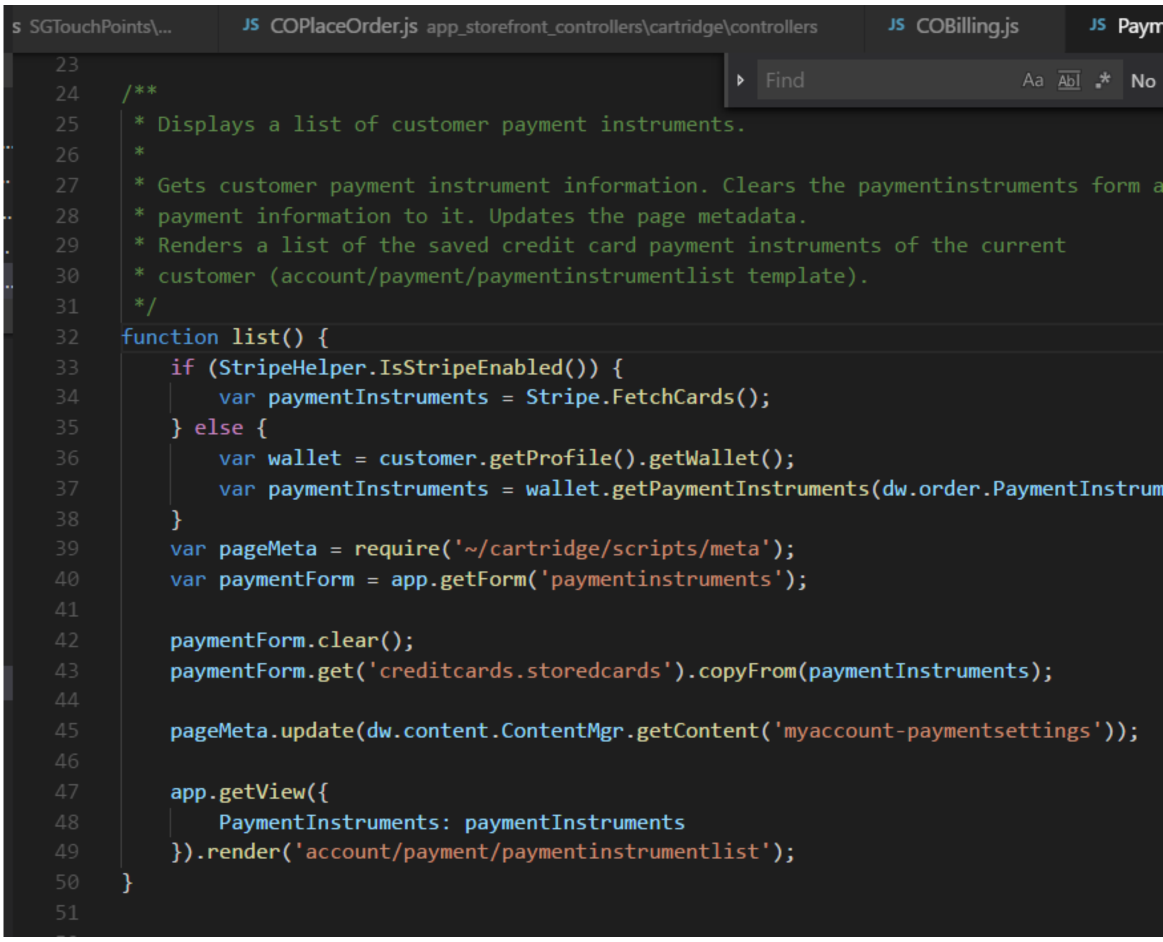
- Add Stripe include files right after "guard" include:

```
var Stripe = require('int_stripe/cartridge/scripts/service/stripe');
var StripeHelper =
require('int_stripe/cartridge/scripts/stripeHelper');
```

- Update the list() function as below:

```
if (StripeHelper.IsStripeEnabled()) {
        var paymentInstruments = Stripe.FetchCards();
} else {
        var wallet = customer.getProfile().getWallet();
        var paymentInstruments =
wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD_CREDIT_CARD);
}
```

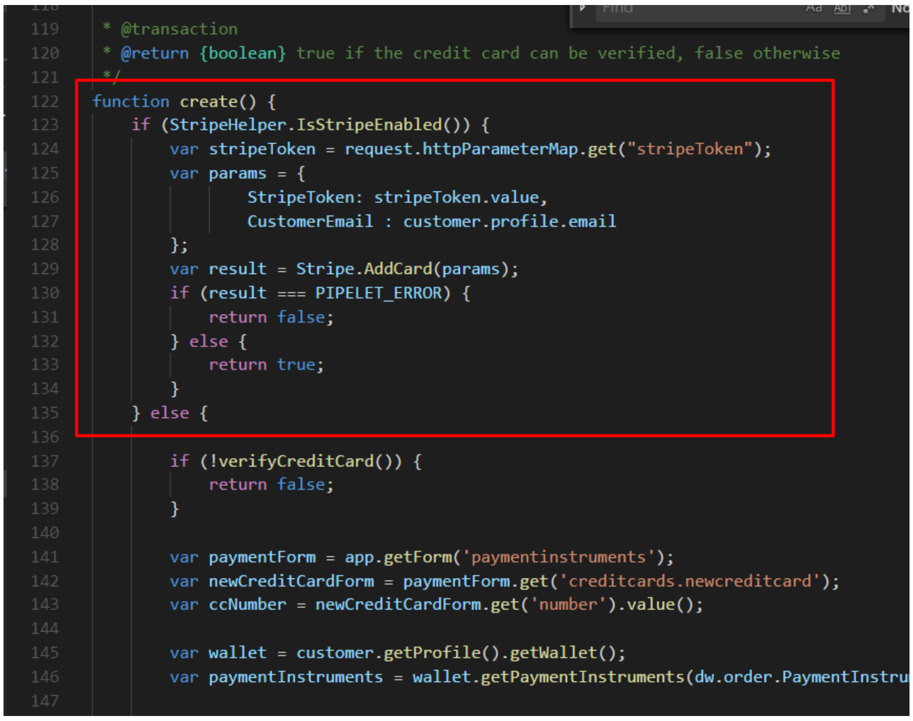
Which will look like this:


s SGTouchPoints\...
COPlaceOrder.js app_storefront_controllers\cartridge\controllers
COBilling.js
Paym
Find
Aa
No

```
/**
 * Displays a list of customer payment instruments.
 *
 * Gets customer payment instrument information. Clears the paymentinstruments form a
 * payment information to it. Updates the page metadata.
 * Renders a list of the saved credit card payment instruments of the current
 * customer (account/payment/paymentinstrumentlist template).
 */
function list() {
    if (StripeHelper.IsStripeEnabled()) {
        var paymentInstruments = Stripe.FetchCards();
    } else {
        var wallet = customer.getProfile().getWallet();
        var paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrum
    }
    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'));

    app.getView({
        PaymentInstruments: paymentInstruments
    }).render('account/payment/paymentinstrumentlist');
}
```

- Update create() function at the beginning with the following:
  Note: Don’t forget to add the closing parenthesis.

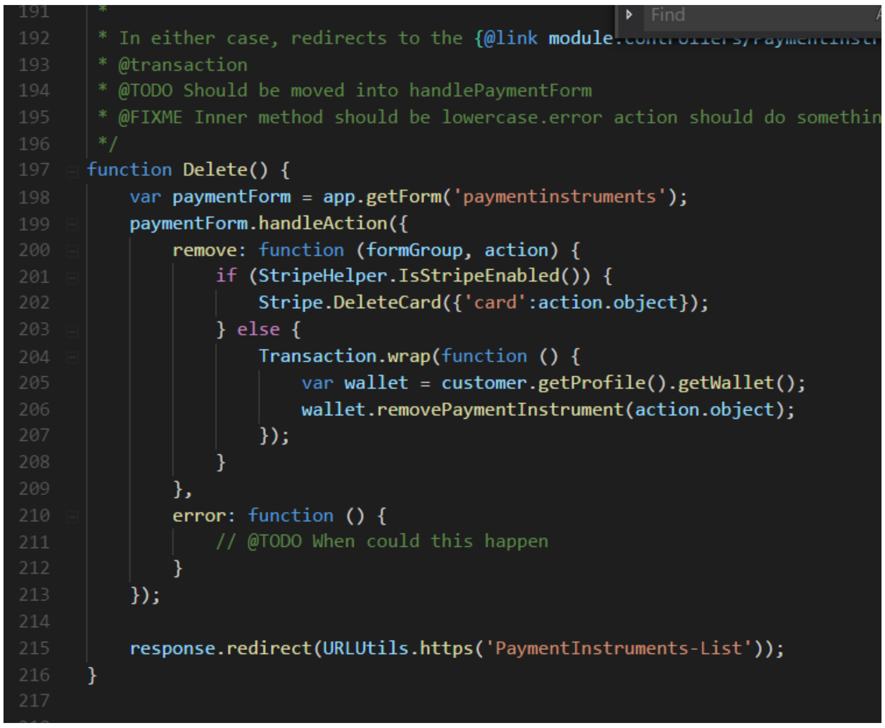
```
if (StripeHelper.IsStripeEnabled()) {
       var stripeToken = request.httpParameterMap.get("stripeToken");
    var params = {
              StripeToken: stripeToken.value,
              CustomerEmail : customer.profile.email
    };
    var result = Stripe.AddCard(params);
    if (result === PIPELET_ERROR) {
        return false;
    } else {
        return true;
    }
} else {
```



```
 * @transaction
 * @return {boolean} true if the credit card can be verified, false otherwise
 */
function create() {
    if (StripeHelper.IsStripeEnabled()) {
        var stripeToken = request.httpParameterMap.get("stripeToken");
        var params = {
                StripeToken: stripeToken.value,
                CustomerEmail : customer.profile.email
        };
        var result = Stripe.AddCard(params);
        if (result === PIPELET_ERROR) {
            return false;
        } else {
            return true;
        }
    } else {

        if (!verifyCreditCard()) {
            return false;
        }

        var paymentForm = app.getForm('paymentinstruments');
        var newCreditCardForm = paymentForm.get('creditcards.newcreditcard');
        var ccNumber = newCreditCardForm.get('number').value();

        var wallet = customer.getProfile().getWallet();
        var paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstru
```

- Update the Delete() function **remove** function with the following:

```
remove: function (formGroup, action) {
        if (StripeHelper.IsStripeEnabled()) {
                Stripe.DeleteCard({'card':action.object});
        } else {
                Transaction.wrap(function () {
                 var wallet = customer.getProfile().getWallet();
                 wallet.removePaymentInstrument(action.object);
            });
        }
}
```



```
 *
 * In either case, redirects to the {@link module.controllers/PaymentInstr
 * @transaction
 * @TODO Should be moved into handlePaymentForm
 * @FIXME Inner method should be lowercase.error action should do somethin
 */
function Delete() {
    var paymentForm = app.getForm('paymentinstruments');
    paymentForm.handleAction({
        remove: function (formGroup, action) {
            if (StripeHelper.IsStripeEnabled()) {
                Stripe.DeleteCard({'card':action.object});
            } else {
                Transaction.wrap(function () {
                    var wallet = customer.getProfile().getWallet();
                    wallet.removePaymentInstrument(action.object);
                });
            }
        },
        error: function () {
            // @TODO When could this happen
        }
    });

    response.redirect(URLUtils.https('PaymentInstruments-List'));
}
```

### 3.3.2 *Scripts*

**Script: account.js**

app_storefront_core/cartridge/js/pages/account.js

- Add file include for stripe right after *validator* as below:
  - Note: Don't forget to add a comma at the end of *validator* declaration.

```
stripe = require('../stripe');
```

- Update initPaymentEvents() function callback as below:

```
                 },
callback: function () {
               if (SitePreferences.STRIPE_ENABLED) {
                              stripe.initMyAccount();
               }
}
```

- Update initializePaymentForm() by commenting out the following:

```
//e.preventDefault();
//dialog.close();
```

**Script: billing.js**

app_storefront_core/cartridge/js/pages/checkout/billing.js

- Add stripe js include right after util include:
  Note: Don't forget to add a comma at the end of *util* declaration.

```
stripe = require('../../stripe');
```

- Update updatePaymentMethod() function with the following declaration:

```
var $paymentMethodsList = $('.payment-method-options');
```

- Update updatePaymentMethod() function by updating the following <u>after</u> *"$selectedPaymentMethod.addClass('payment-method-expanded');"*

```
$('#stripe-payment-errors').html('');
//remove previous payment method selection
$paymentMethodsList.find(':checked').removeAttr('checked');
```

- Update exports.init function by adding the following at the end of the function:

```
if (SitePreferences.STRIPE_ENABLED) {
       stripe.initBilling();
}
```

**Script: stripe.js (New File)**

- Copy file **int_stripe/cartridge/js/stripe.js** and paste to location below:

  *app_storefront_core/cartridge/js/stripe.js*

**DW Script: Resource.ds**

app_storefront_core/cartridge/scripts/util/Resource.ds

Add in the resources json after COULD_NOT_SAVE_ADDRESS:

```
        STRIPE_TAX_MSG                                     :
Resource.msg('order.ordersummary.ordertax','order',null),
        STRIPE_SHIPPING_MSG                       :
Resource.msg('order.ordersummary.ordershipping','order',null),
```

- Add the following file module includes right after ProductAvailabilityModel include:

```
var StripeHelper = require('int_stripe/cartridge/scripts/stripeHelper');
var TaxMgr = require('dw/order/TaxMgr');
```

- Update **ResourceHelper.getUrls** function at the end with the url settings below:

```
            ,
      // Stripe
      stripeBillingSelectCC        : URLUtils.https('Stripe-SelectCreditCard').toString()
```

- Update **ResourceHelper.getPreferences** function at the end with the following new references:

```
,
STRIPE_ENABLED: StripeHelper.IsStripeEnabled(),
STRIPE_CARD_STYLE: StripeHelper.getElementsStyle(),
STRIPE_PR_STYLE: StripeHelper.getPRStyle(),
STRIPE_COUNTRY: StripeHelper.getPRCountry(),
STRIPE_CURRENCY: session.currency.getCurrencyCode().toLowerCase(),
STRIPE_TAX: (TaxMgr.getTaxationPolicy() == TaxMgr.TAX_POLICY_GROSS ? 'gross' : 'net')
```

***DW Script: ValidatePaymentInstruments.js***
app_storefront_core/cartridge/scripts/checkout/ValidatePaymentInstruments.ds

- Add the stripe helper include file right after *"importScript("checkout/Utils.ds");"*

```
var StripeHelper = require('int_stripe/cartridge/scripts/stripeHelper');
```

- Add the following if statement at the beginning of execute function:

```
      if (!StripeHelper.IsStripeEnabled()) {
```

- Add closing parenthesis for the above if statement right before *"return PIPELET_NEXT;"*

**DW Script: common.js**
app_storefront_controllers/cartridge/scripts/payment/common.js

- Update validatePaymentInstruments() function with the following code. Add before var method = PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod());

```
//Ignore Stripe payment instruments
             if (paymentInstrument.paymentMethod === 'STRIPE_APM_METHODS') {
                     if
(require('int_stripe/cartridge/scripts/stripeHelper').ValidatePaymentMethod(paymentInstrument)) {
                     continue;
                     }
             }
```

### 3.3.3 *Forms*

**Form: creditcard.xml**
app_storefront_core/cartridge/forms/default/creditcard.xml

Update xml with the following data only:

```
<?xml version="1.0"?>
<form xmlns="http://www.demandware.com/xml/form/2008-04-19">

    <!-- field for credit card type -->
    <field formid="type" label="creditcard.type" type="string" mandatory="true"
binding="creditCardType"
        missing-error="creditcard.typemissing">
        <options optionid-binding="cardType" value-binding="cardType" label-binding="name"/>
    </field>
```

```
  <!-- field for credit card owner -->
    <field formid="owner" label="creditcard.ownerlabel" type="string" mandatory="true" max-
length="40" binding="creditCardHolder"
      missing-error="creditcard.ownermissingerror"/>

  <!-- optional flags -->
  <field formid="saveCard" label="creditcard.savecard" type="boolean" mandatory="false" default-
value="true" />

  <!-- confirm action to save the card details -->
    <action formid="confirm" valid-form="true"/>

</form>
```

### 3.3.4 *Templates*

**Template: minicreditcard.isml**

app_storefront_core/cartridge/templates/default/account/payment/minicreditcard.isml

- Update script with the following data right before the end of <isscript> and replace existing.

```
 var stripeEnabled = require('int_stripe/cartridge/scripts/stripeHelper').IsStripeEnabled();
 if (stripeEnabled) {
   ccTypeName = "cardtype." + ccType;
   ccType = Resource.msg(ccTypeName,"checkout",null);
 }

 if ('isCreditCard' in pdict.card && !pdict.card.isCreditCard) {
   ccOwner = pdict.card.sourceHolder;
   ccType = dw.web.Resource.msg('stripe.' + pdict.card.sourceType, 'checkout',
pdict.card.sourceType);
 }

</isscript>
<isif condition="${'isCreditCard' in pdict.card && !pdict.card.isCreditCard && pdict.card.sourceType
== 'ach_credit_transfer'}">
 <div class="cc-type"><isprint value="${ccType}"/></div>
 <div class="cc-owner"><isprint value="${ccOwner}"/></div>
 <div class="cc-number">${Resource.msg('account.ach_credit_transfer.accountnumber','account',null)}
<isprint value="${pdict.card.sourceInfo.account_number}"/></div>
 <div class="cc-number">${Resource.msg('account.ach_credit_transfer.bank','account',null)} <isprint
value="${pdict.card.sourceInfo.bank_name}"/></div>
 <div class="cc-number">${Resource.msg('account.ach_credit_transfer.routing_number','account',null)}
<isprint value="${pdict.card.sourceInfo.routing_number}"/></div>
<iselseif condition="${'isCreditCard' in pdict.card && !pdict.card.isCreditCard}">
 <div class="cc-type"><isprint value="${ccType}"/></div>
<iselse>
 <isif condition="${ccOwner && ccType && ccNumber}">
   <div class="cc-owner"><isprint value="${ccOwner}"/></div>
   <div class="cc-type"><isprint value="${ccType}"/></div>
   <div class="cc-number"><isprint value="${ccNumber}"/></div>
   <isif condition="${pdict.show_expiration}"><div class="cc-
exp">${Resource.msg('account.payment.minicardcard.expires','account',null)}
${Resource.msgf('global.creditcard.expirydate.separator','locale',null,
StringUtils.formatNumber(ccMonth, "00"), StringUtils.formatNumber(ccYear, "0000"))}</div>
   </isif>
 </isif>
</isif>
```

**Template: paymentmethods.isml**

app_storefront_core/cartridge/templates/default/checkout/billing/paymentmethods.isml

- Add the following after “<div class="payment-method-options form-indent">”

```
 <isloop items="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.options}"
var="paymentMethodType">
   <isif condition="${paymentMethodType.value.equals('STRIPE_PAYMENT_REQUEST_BTN')}">
```

```
        <div id="payment-request-button"></div>
        <isbreak/>
    </isif>
  </isloop>
```

- Add the following after “<isif condition="${paymentMethodType.value.equals(dw.order.PaymentInstrument.METHOD_GIFT_CERTIFICATE)}"><is continue/></isif>”

```
<isif
condition="${paymentMethodType.value.equals('STRIPE_PAYMENT_REQUEST_BTN')}"><iscontinue/></isif>
```

- Add the include file below after closing the loop for script above:
```
   <isinclude template="stripe/paymentoptions"/>
```

- Update the credit card block with the following:

```
   <isscript>
        <iscomment>Check if Stripe is enabled</iscomment>
        var stripeEnabled = require('int_stripe/cartridge/scripts/stripeHelper').IsStripeEnabled();
   </isscript>

   <div class="payment-method <isif condition="${empty(pdict.selectedPaymentID) ||
pdict.selectedPaymentID=='CREDIT_CARD'}">payment-method-expanded</isif>" data-method="CREDIT_CARD">

        <iscomment>display select box with stored credit cards if customer is
authenticated</iscomment>
        <isif condition="${pdict.CurrentCustomer.authenticated &&
!empty(pdict.ApplicableCreditCards)}">

               <div class="form-row">
                       <isif condition="${stripeEnabled}">
                               <iscomment>Stripe Payment Methods</iscomment>
                               <isinclude template="checkout/billing/stripe_paymentmethods"/>
                       <iselse/>
                                       // Default Site Genesis / CC Saved Cards code
                               <label
class="label">${Resource.msg('billing.selectcreditcard','checkout',null)}</label>
                               <div class="field-wrapper">
                                       <select
name="${pdict.CurrentForms.billing.paymentMethods.creditCardList.htmlName}" id="creditCardList"
class="input-select">
                                               <option value=""
selected="selected">${Resource.msg('billing.creditcardlistselect','checkout',null)}</option>
                                               <isloop items="${pdict.ApplicableCreditCards}"
var="creditCardInstr">
                                                       <option value="${creditCardInstr.UUID}">(<isprint
value="${creditCardInstr.creditCardType}"/>) <isprint
value="${creditCardInstr.maskedCreditCardNumber}"/> -
${Resource.msg('billing.creditcardlistexp','checkout',null)} <isprint
value="${creditCardInstr.creditCardExpirationMonth}" formatter="00" />.<isprint
value="${creditCardInstr.creditCardExpirationYear}" formatter="0000" /></option>
                                               </isloop>
                                       </select>
                               </div>
                       </isif>
               </div>

               <div class="form-row form-row-button">
                       <button id="credit-card-select-go"
name="${pdict.CurrentForms.billing.creditCardSelect.htmlName}" type="submit" value="Go"
class="simple-submit">Select</button>
               </div>
        </isif>

        <isif condition="${stripeEnabled}">
               <iscomment>Stripe replacement CC fields</iscomment>
               <isinclude template="checkout/billing/stripe_creditcard_fields"/>
        <iselse/>
            // Default Site Genesis / CC form fields code
```

```
               <isinputfield
formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.owner}" type="input"/>

               <isif condition="${pdict.CurrentCustomer.authenticated}">
                      <isinputfield
formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.saveCard}" type="checkbox"/>
               </isif>
        </isif>
   </div>
```

- Add the stripe payment methods include file before the end of </fieldset>

```
   <isinclude template="stripe/paymentmethods"/>
```

**Template: ordertotals.isml**
app_storefront_core/cartridge/templates/default/components/order/ordertotals.isml

- Add the 2 hidden fields in bold to your script:

```
 <isif condition="${dw.order.TaxMgr.getTaxationPolicy() == dw.order.TaxMgr.TAX_POLICY_NET}">
                     <tr class="order-sales-tax">
                            <td>${Resource.msg('order.ordersummary.ordertax','order',null)}</td>
                            <td>
                                   <isif condition="${LineItemCtnr.totalTax.available}">
                                          <isprint value="${LineItemCtnr.totalTax}"/>
 input type="hidden" name="ordertaxvalue" id="ordertaxvalue" value="${(LineItemCtnr.totalTax.value *
 100).toFixed(0)}"/>
                                   <iselse/>
                                          ${Resource.msg('global.symbol.dash','locale',null)}
   <input type="hidden" name="ordertaxvalue" id="ordertaxvalue" value="0"/>
                                   </isif>
                            </td>
                     </tr>
              </isif>
```

- Add the 2 hidden fields in bold to your script:

```
 <td class="order-value"><isprint value="${orderTotalValue}"/></td>
   <input type="hidden" name="ordertotalvalue" id="ordertotalvalue" value="${(orderTotalValue.value *
 100).toFixed(0)}"/>
   <input type="hidden" name="ordershipvalue" id="ordershipvalue"
 value="${(LineItemCtnr.shippingTotalPrice.value * 100).toFixed(0)}"/>
```

**Template: minibillinginfo.isml**
app_storefront_core/cartridge/templates/default/checkout/billing/minibillinginfo.isml

- Update the if statement on line 44 as below:

```
                <isif condition="${paymentInstr.paymentMethod == 'STRIPE_APM_METHODS'}">
                     <isinclude template="checkout/billing/stripeminibillinginfo"/>
                <iselse>
                    <div><isprint
value="${dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).name}" /></div>
                    <isminicreditcard card="${paymentInstr}" show_expiration="${true}"/>
                    <div>
                        ${Resource.msg('minibillinginfo.amount', 'checkout', null)}: <span
class="minibillinginfo-amount"><isprint value="${paymentInstr.paymentTransaction.amount}"/></span>
                    </div><!-- END: payment-amount -->
                </isif>
```

**Template: confirmation.isml**
app_storefront_core/cartridge/templates/default/checkout/confirmation/confirmation.isml

- Replace the h1 tag with the following:

```
        <isif
condition="${require('int_stripe/cartridge/scripts/stripeHelper').IsOrderPlacedUsingAPMMethod(pdict.O
rder)}">
                <h1>${Resource.msg('stripe.confirmation.thankyou','checkout',null)}</h1>
        <iselse>
                <h1>${Resource.msg('confirmation.thankyou','checkout',null)}</h1>
        </isif>
```

**Template: orderdetails.isml**
app_storefront_core/cartridge/templates/default/components/order/orderdetails.isml

- Update the div on line 63 <*div class="payment-type"*> with the following.

```
                 <isif
condition="${require('int_stripe/cartridge/scripts/stripeHelper').IsOrderPlacedUsingAPMMethod(Order)}
">
                        <isinclude template="checkout/billing/stripeminibillinginfo"/>
                 <iselse>
                        <div class="payment-type"><isprint
value="${dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).name}" /></div>
                        <isminicreditcard card="${paymentInstr}" show_expiration="${false}"/>
                        <div class="payment-amount">
                                <span
class="label">${Resource.msg('global.amount','locale',null)}:</span>
                                <span class="value"><isprint
value="${paymentInstr.paymentTransaction.amount}"/></span>
                        </div><!-- END: payment-amount -->
                </isif>
Add the following before closing the div <div class="shipping-status">
<isif
condition="${require('int_stripe/cartridge/scripts/stripeHelper').IsOrderPlacedUsingAPMMethod(Order)}
">
<div class="error">${Resource.msg('stripe.confirmation.shippingnote', 'checkout', null)}</div>
</isif>
```

**Template: orderdetailsemail.isml**
app_storefront_core/cartridge/templates/default/components/order/orderdetailsemail.isml

- Update everything inside the loop *<isloop items="${Order.getPaymentInstruments()}"* with the following:

```
<isif
condition="${require('int_stripe/cartridge/scripts/stripeHelper').IsOrderPlacedUsingAPMMethod(Order)}
">
                        <isinclude template="checkout/billing/stripeminibillinginfo"/>
                 <iselse>
                        <div><isprint
value="${dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).name}" /></div>
                        <isif
condition="${dw.order.PaymentInstrument.METHOD_GIFT_CERTIFICATE.equals(paymentInstr.paymentMethod)}">
                                <isprint value="${paymentInstr.maskedGiftCertificateCode}"/><br />
                        </isif>
                        <isminicreditcard card="${paymentInstr}" show_expiration="${false}"/>
                        <div>
                                <span
class="label">${Resource.msg('global.amount','locale',null)}:</span>
                                <span class="value"><isprint
value="${paymentInstr.paymentTransaction.amount}"/></span>
                        </div><!-- END: payment-amount →
</isif>
```

- Add the following after the comment *<iscomment>render a box for each shipment</iscomment>*

```
<isif
condition="${require('int_stripe/cartridge/scripts/stripeHelper').IsOrderPlacedUsingAPMMethod(Order)}
">
<div class="error">${Resource.msg('stripe.confirmation.shippingnote', 'checkout', null)}</div>
</isif>
```

**Template: htmlhead.isml**
app_storefront_core/cartridge/templates/default/components/header/htmlhead.isml

- Add the stripe header include file to the bottom of this file.

```
<isinclude template="stripe/headerinclude"/>
```

**Template: footer_UI.isml**
app_storefront_core/cartridge/templates/default/components/footer/footer_UI.isml

- Add the following script include file at the end of this file.

```
<iscomment>Stripe Payment Type javascript library</iscomment>
<script type="text/javascript" src="${URLUtils.staticURL('stripe/jquery.payment.js')}"></script>
```

**Template: paymentinstrumentlist.isml**
app_storefront_core/cartridge/templates/default/account/payment/paymentinstrumentlist.isml

- Update the if statement as below:

```
<isif condition="${pdict.PaymentInstruments.size() > 0}">
       <isscript>
              stripeAPMMethodsEnable = 
require('int_stripe/cartridge/scripts/stripeHelper').IsStripePaymentMethodsEnabled();
       </isscript>
       <ul class="payment-list">
              <isloop items="${pdict.CurrentForms.paymentinstruments.creditcards.storedcards}" 
var="creditcard" status="loopstate">
              <isif condition="${!creditcard.object.isCreditCard && !stripeAPMMethodsEnable}">
                     <iscontinue/>
              </isif>
              <li class="<isif condition="${loopstate.first}">first <iselseif 
condition="${loopstate.last}">last </isif>${pdict.PaymentInstruments[loopstate.count - 
1].creditCardType}">
                     <isminicreditcard card="${creditcard.object}" show_expiration="${true}"/>
                     <form action="${URLUtils.url('PaymentInstruments-Delete')}" name="payment-
remove" method="post" id="creditcards_${loopstate.count}">
                            <fieldset>
                                   <button type="submit" class="button-text delete" 
value="${Resource.msg('account.paymentinstrumentlist.deletecard','account',null)}" 
name="${creditcard.remove.htmlName}">

 ${Resource.msg('account.paymentinstrumentlist.deletecard','account',null)}
                                   </button>
                                   <input type="hidden" 
name="${pdict.CurrentForms.paymentinstruments.secureKeyHtmlName}" 
value="${pdict.CurrentForms.paymentinstruments.secureKeyValue}"/>
                            </fieldset>
                     </form>
                     <iscomment>Stripe - Make Default Link</iscomment>
                     <isinclude template="account/payment/makedefaultlink"/>
              </li>
              </isloop>
       </ul>
   </isif>
```

**Template: paymentinstrumentdetails.isml**
app_storefront_core/cartridge/templates/default/account/payment/paymentinstrumentdetails.isml

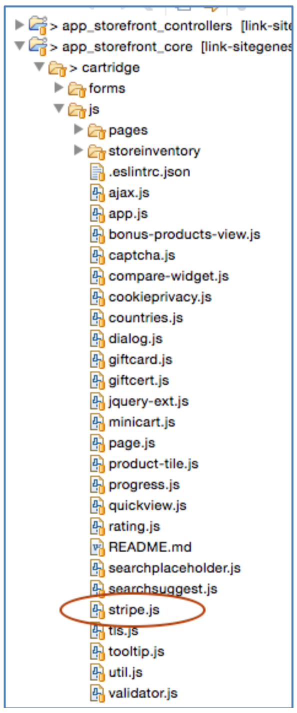
- Update the following after the closing **</isscript>** statement until **<input type="hidden"**

```
              <iscomment>Stripe Payment Errors</iscomment>
              <isinclude template="stripe/paymenterrors"/>
              <isscript>
                     <iscomment>Check if Stripe is enabled</iscomment>
                     var stripeEnabled = 
 require('int_stripe/cartridge/scripts/stripeHelper').IsStripeEnabled();
              </isscript>
              <isif condition="${stripeEnabled}">
                     <iscomment>Stripe Payment Methods</iscomment>
                     <isinputfield 
 formfield="${pdict.CurrentForms.paymentinstruments.creditcards.newcreditcard.owner}" type="input" 
 attributes="${ownerAttributes}" />
                     <isinclude template="account/payment/stripeelements"/>
              <iselse/>
```

```
                    <iscomment>
                           Original stripe implementation
                    </iscomment>
                    <isinputfield
formfield="${pdict.CurrentForms.paymentinstruments.creditcards.newcreditcard.owner}" type="input"
attributes="${ownerAttributes}" />
                    <iscomment>Stripe - hide type field</iscomment>
                    <input type="hidden"
name="${pdict.CurrentForms.paymentinstruments.creditcards.newcreditcard.type.htmlName}"
value="${pdict.CurrentForms.paymentinstruments.creditcards.newcreditcard.type.htmlValue}"/>
                    <isinputfield
formfield="${pdict.CurrentForms.paymentinstruments.creditcards.newcreditcard.number}"
dynamicname="true" type="input" attributes="${numberAttributes}"/>
                    <div class="form-label-text">
                           <span class="required-indicator">&#8226; </span>

  ${Resource.msg('account.paymentinstrumentdetails.expires','account',null)}
                    </div>
                    <isscript>
                                  var currentCountry =
require('~/cartridge/scripts/util/Countries').getCurrent(pdict);
                    </isscript>

                    <isdynamicform
formobject="${pdict.CurrentForms.paymentinstruments.creditcards.newcreditcard.expiration}"
formdata="${currentCountry.dynamicForms.expirationInfo}" />

                    <iscomment>Stripe Additional Inputs</iscomment>
                    <isinclude template="account/payment/stripeinputs"/>

                    <div class="form-row form-row-button">
                           <button id="applyBtn" type="submit"
name="${pdict.CurrentForms.paymentinstruments.creditcards.create.htmlName}"
value="${Resource.msg('global.apply','locale',null)}">
                                  ${Resource.msg('global.apply', 'locale', null)}
                           </button>
                           <button class="cancel cancel-button simple" type="submit"
name="${pdict.CurrentForms.paymentinstruments.creditcards.cancel.htmlName}"
value="${Resource.msg('global.cancel','locale',null)}">
                                  ${Resource.msg('global.cancel', 'locale', null)}
                           </button>
                    </div>
             </isif>
```

### 3.3.5 *JavaScript*

**Add custom Stripe JavaScript to the Storefront by copying js/stripe.js from the int_stripe cartridge to the js/ folder of the Storefront Cartridge (core)**

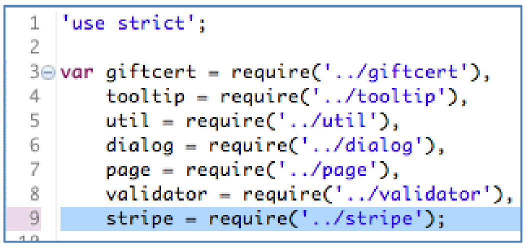


```
▶ > app_storefront_controllers  [link-site
▼ > app_storefront_core  [link-sitegenes
  ▼ > cartridge
    ▶ forms
    ▼ js
      ▶ pages
      ▶ storeinventory
        .eslintrc.json
        ajax.js
        app.js
        bonus-products-view.js
        captcha.js
        compare-widget.js
        cookieprivacy.js
        countries.js
        dialog.js
        giftcard.js
        giftcert.js
        jquery-ext.js
        minicart.js
        page.js
        product-tile.js
        progress.js
        quickview.js
        rating.js
        README.md
        searchplaceholder.js
        searchsuggest.js
        stripe.js
        tls.js
        tooltip.js
        util.js
        validator.js
```

**Make the following updates to *pages/account.js***

1. Add stripe object reference

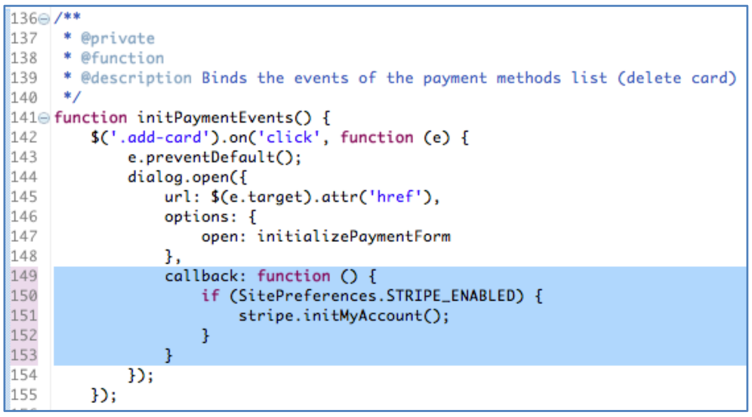
   stripe = require('../stripe');



```
'use strict';

var giftcert = require('../giftcert'),
    tooltip = require('../tooltip'),
    util = require('../util'),
    dialog = require('../dialog'),
    page = require('../page'),
    validator = require('../validator'),
    stripe = require('../stripe');
```

2. Add the following code to the initPaymentEvents() function as an additional property in the $('.add-card') event

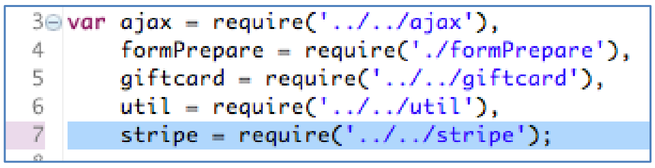
```
callback: function () {
   if (SitePreferences.STRIPE_ENABLED) {
      stripe.initMyAccount();
   }
}
```


```
/**
 * @private
 * @function
 * @description Binds the events of the payment methods list (delete card)
 */
function initPaymentEvents() {
    $('.add-card').on('click', function (e) {
        e.preventDefault();
        dialog.open({
            url: $(e.target).attr('href'),
            options: {
                open: initializePaymentForm
            },
            callback: function () {
                if (SitePreferences.STRIPE_ENABLED) {
                    stripe.initMyAccount();
                }
            }
        });
    });
```

**Make the following updates to *pages/checkout/billing.js***

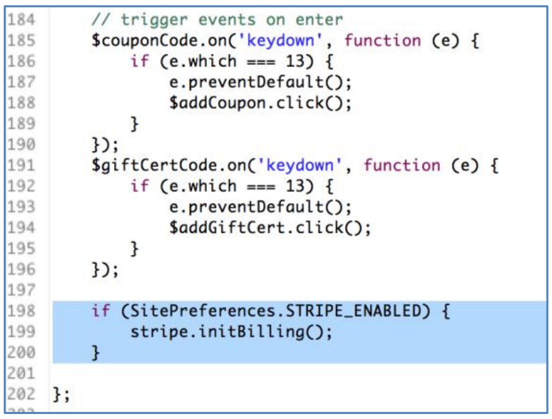
1. Add the stripe object reference to the require section

   stripe = require('../../stripe');


```
var ajax = require('../../ajax'),
    formPrepare = require('./formPrepare'),
    giftcard = require('../../giftcard'),
    util = require('../../util'),
    stripe = require('../../stripe');
```

2. Add the following code to the end of the exports.init function

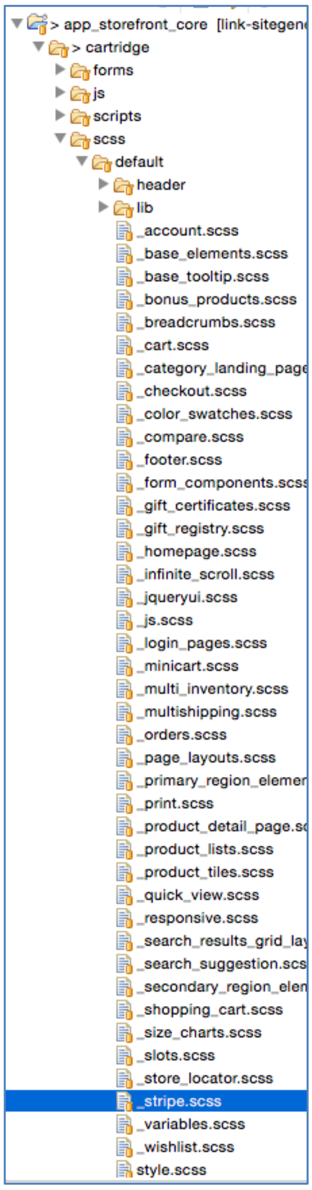
```
if (SitePreferences.STRIPE_ENABLED) {
    stripe.initBilling();
}
```


```
    // trigger events on enter
    $couponCode.on('keydown', function (e) {
        if (e.which === 13) {
            e.preventDefault();
            $addCoupon.click();
        }
    });
    $giftCertCode.on('keydown', function (e) {
        if (e.which === 13) {
            e.preventDefault();
            $addGiftCert.click();
        }
    });

    if (SitePreferences.STRIPE_ENABLED) {
        stripe.initBilling();
    }

};
```

## 3.3.6 *CSS*

Add _stripe.scss styles to the storefront

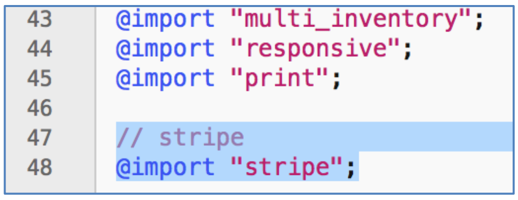
1. Copy/paste int_stripe/cartridge/scss/_stripe.css to [storefront_cartridge]/cartridge/scss/_stripe.css



```
▼ > app_storefront_core  [link-sitegen
  ▼ > cartridge
    ▶ forms
    ▶ js
    ▶ scripts
    ▼ scss
      ▼ default
        ▶ header
        ▶ lib
          _account.scss
          _base_elements.scss
          _base_tooltip.scss
          _bonus_products.scss
          _breadcrumbs.scss
          _cart.scss
          _category_landing_page
          _checkout.scss
          _color_swatches.scss
          _compare.scss
          _footer.scss
          _form_components.scss
          _gift_certificates.scss
          _gift_registry.scss
          _homepage.scss
          _infinite_scroll.scss
          _jqueryui.scss
          _js.scss
          _login_pages.scss
          _minicart.scss
          _multi_inventory.scss
          _multishipping.scss
          _orders.scss
          _page_layouts.scss
          _primary_region_elemer
          _print.scss
          _product_detail_page.sc
          _product_lists.scss
          _product_tiles.scss
          _quick_view.scss
          _responsive.scss
          _search_results_grid_lay
          _search_suggestion.scs
          _secondary_region_elen
          _shopping_cart.scss
          _size_charts.scss
          _slots.scss
          _store_locator.scss
          _stripe.scss
          _variables.scss
          _wishlist.scss
          style.scss
```

2. Update [storefront_cartridge]/cartridge/scss/default/style.scss to reference _stripe.scss

   // stripe
   @import "stripe";



```
@import "multi_inventory";
@import "responsive";
@import "print";

// stripe
@import "stripe";
```

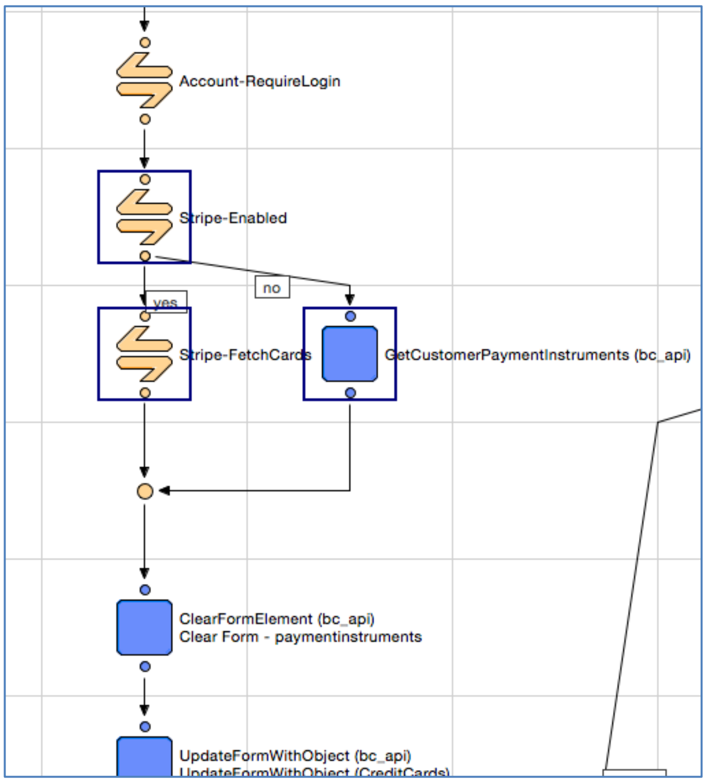
3. At this point the styles can be customized to match any required site theming

### 3.3.7 *Pipelines*

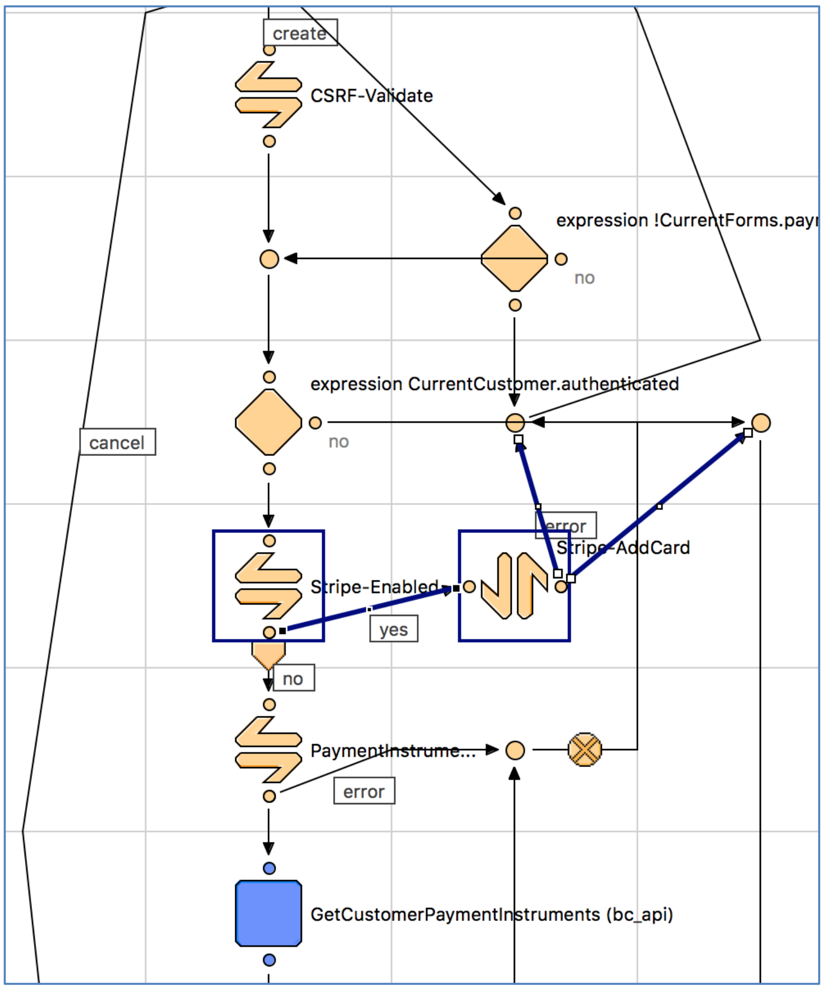
Pipeline updates are only required if integrating with Pipelines instead of using Controllers

**Update PaymentInstruments.xml with the following customizations**

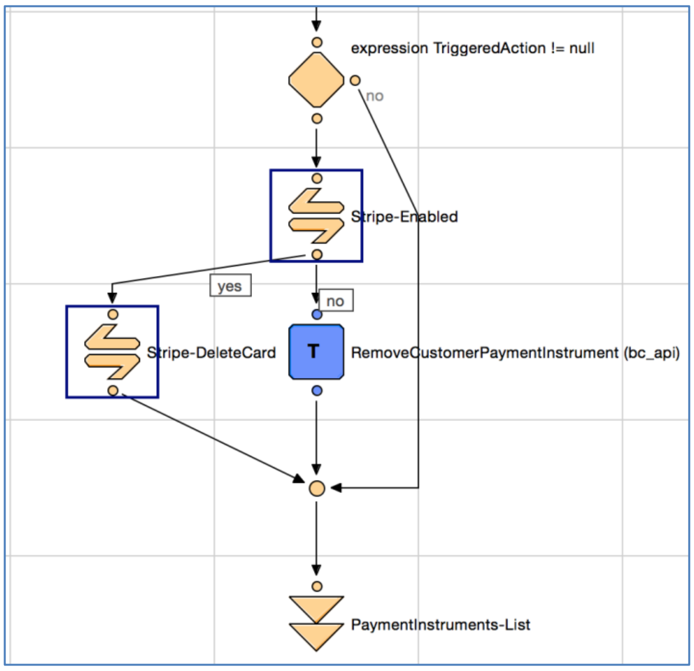
1. Update PaymentInstruments-List pipeline to add call nodes to Stripe-Enabled with a ‘yes’ connector directing to an additional call node Stripe-FetchCards and a ‘no’ connector directing to the default GetCustomerPaymentInstruments pipelet


Account-RequireLogin
Stripe-Enabled
no
yes
Stripe-FetchCards
GetCustomerPaymentInstruments (bc_api)
ClearFormElement (bc_api)
Clear Form - paymentinstruments
UpdateFormWithObject (bc_api)
UpdateFormWithObject (CreditCards)

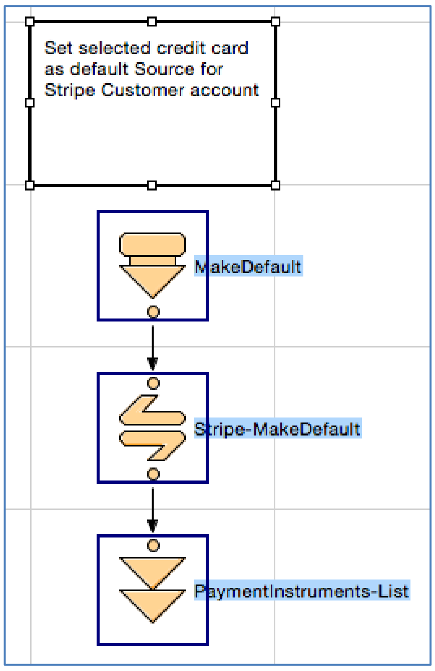
2. Update PaymentInstruments-Add pipeline to add call node just after the ‘CurrentCustomer.authenticated’ to Stripe-Enabled. From the Stripe-Enabled call node, add two connectors: 1. ‘no’ connector to the existing PaymentInstruments-VerifyCreditCard call node and set the connector’s Transaction Control property to “Begin Transaction”; 2: ‘yes’ connector to a new Stripe-AddCard call node. From the Stripe-AddCard call node add a ‘next’ connector to redirect to the end of the pipeline. From the Stripe-AddCard, add a second ‘error’ connector that connects back to the error direction back to the initial account/payment/paymentinstrumentdetails Interaction Node. Connect the error connector to the same node where the rollback transaction connector connects to.


create
CSRF-Validate
expression !CurrentForms.payı
no
expression CurrentCustomer.authenticated
cancel
no
error
Stripe-AddCard
Stripe-Enabled
yes
no
PaymentInstrume...
error
GetCustomerPaymentInstruments (bc_api)

3. Update PaymentInstruments-Delete pipeline to add a call node to Stripe-Enabled with a ‘yes’ connector directing to a new Stripe-DeleteCard call node and a ‘no’ connector directing to the default RemoveCustomerPaymentInstrument pipelet

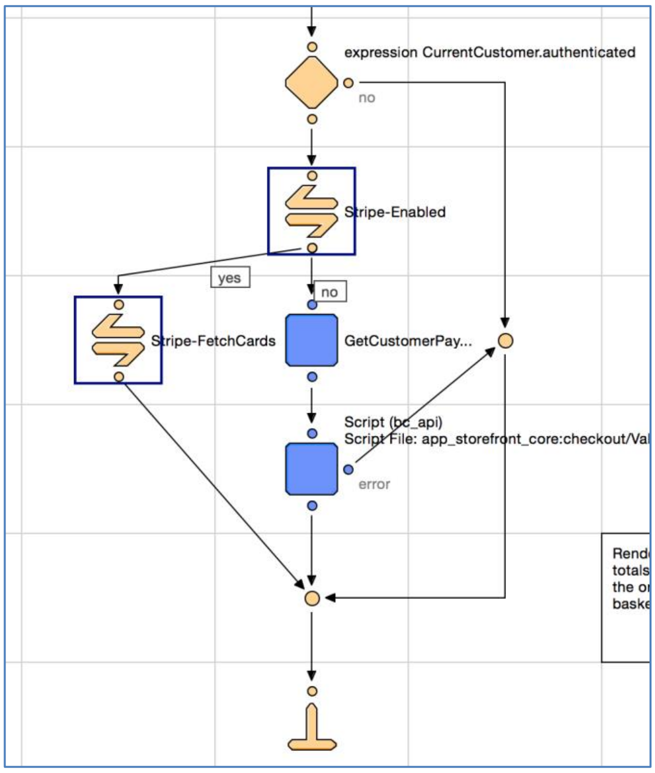

expression TriggeredAction != null
no
Stripe-Enabled
yes
no
Stripe-DeleteCard
T
RemoveCustomerPaymentInstrument (bc_api)
PaymentInstruments-List

4. Add a new Pipeline named PaymentInstruments-MakeDefault
   Start Node: MakeDefault
   Call Node: Stripe-MakeDefault
   Jump Node: PaymentInstruments-List

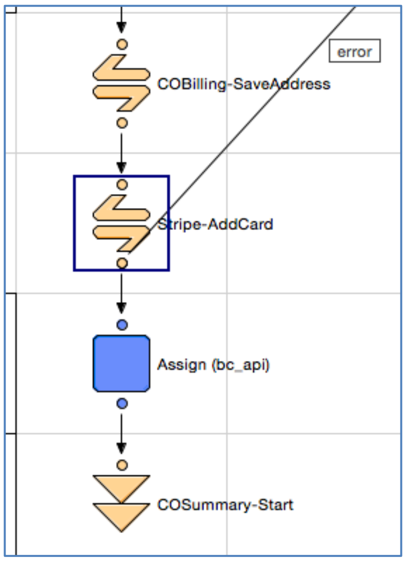

Set selected credit card as default Source for Stripe Customer account
MakeDefault
Stripe-MakeDefault
PaymentInstruments-List

**Update COBilling.xml with the following customizations**

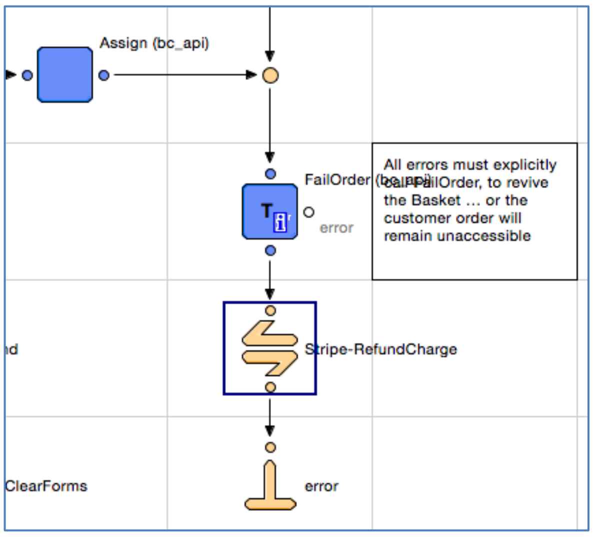
1. Update COBilling-InitCreditCardList pipeline to add a call node to Stripe-Enabled with a ‘yes’ connector directing to a new Stripe-FetchCards call node and a ‘no’ connector directing to the default GetCustomerPaymentInstruments pipelet



2. Add a call node to the Stripe-AddCard in the COBilling-Start pipeline just after the COBilling-SaveAddress call node. Send an 'error' connector to the standard error direction



**Update the COPlaceOrder-Start pipeline in *COPlaceOrder.xml* to add a Stripe-RefundCharge call node just after the FailOrder pipelet**


Assign (bc_api)
FailOrder (bc_api)
error
All errors must explicitly call FailOrder, to revive the Basket ... or the customer order will remain unaccessible
d
Stripe-RefundCharge
ClearForms
error

## 3.4 Apple Pay JS and Payment Request Button

Using the Stripe Payment Request button will Proxy ApplePay JS (for the Web) from your storefront.  In order to do this Stripe will pass an ApplePay domain verification string as part of the setup in the Stripe dashboard.  This verification string can be set into the Commerce Cloud site preference created in the cartridge for that reason.

Included in the cartridge is a controller method in Home.js for applepay() which accomplish the domain verification that is needed to authorize the storefront.

This controller method will lookup the site preference apple verification and output to the root of the domain.

Setup a Dynamic mapping Rule in Business Manager to Redirect the root domain to the controller

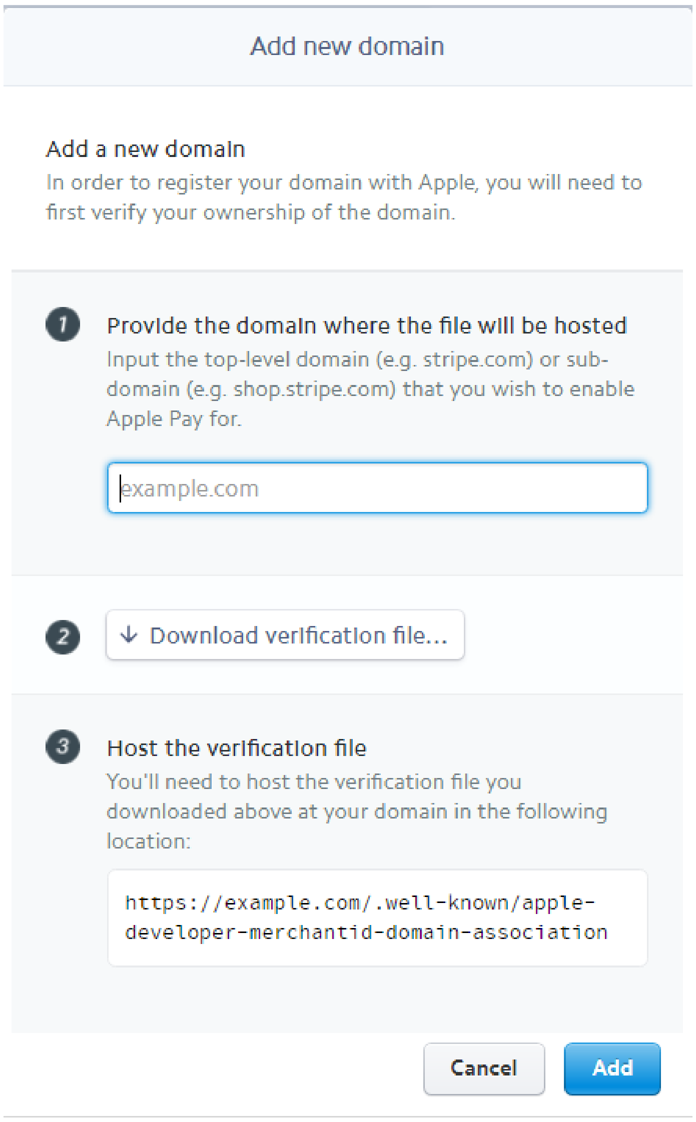
```
/.well-known/apple* p,,,Home-Apple,,{_querystring}
```

The ISML will look up the site preference of "applePayVerificationString" and output direct to the browser.

Next, modify the RedirectURL.js controller so that it contains the bold code below:

```

	if (!location) {
        app.getView().render('util/redirecterrorutil/redirecterror');
		} else {
			// Manually send to the apple page if detected
			if (location.indexOf("Home-Apple") > 0 ) {
			app.getView().render('util/apple');
			} else {
				app.getView({
					Location: location
				}).render('util/redirectpermanent');
			}
		}
	}
```

Navigate on the Stripe dashboard to Payments / ApplePay and click “Add new Domain”:


Add new domain

Add a new domain

In order to register your domain with Apple, you will need to first verify your ownership of the domain.

1. Provide the domain where the file will be hosted

Input the top-level domain (e.g. stripe.com) or sub-domain (e.g. shop.stripe.com) that you wish to enable Apple Pay for.

example.com

2. Download verification file...

3. Host the verification file

You'll need to host the verification file you downloaded above at your domain in the following location:

```
https://example.com/.well-known/apple-developer-merchantid-domain-association
```

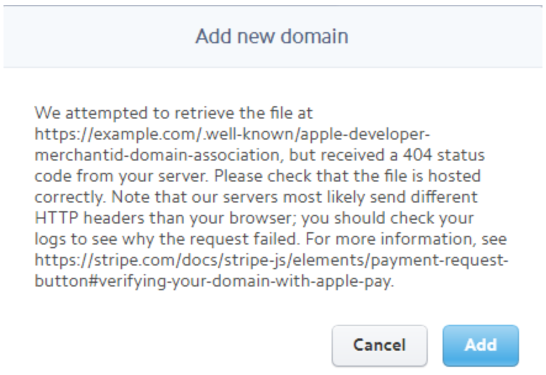
Fill out the desired information and click “add”

Download the verification file, copy that value into your Stripe cartridge preference for “applePayVerificationString”

Next, configure the Stripe dashboard to point at your storefront:

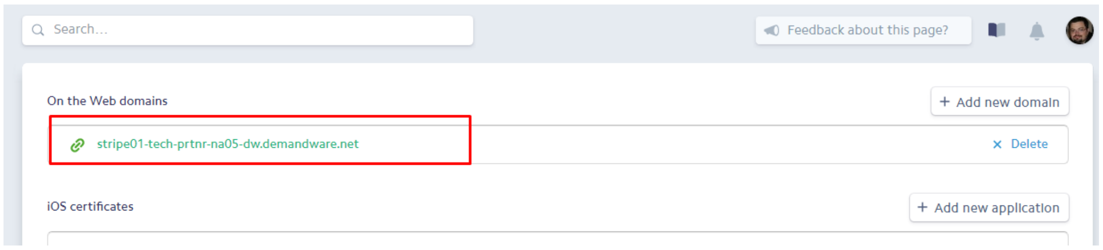
*https://example.com/.well-known/apple-developer-merchantid-domain-association*

If there are issues with the above, Stripe will not be able to validate your domain:


Add new domain

We attempted to retrieve the file at https://example.com/.well-known/apple-developer-merchantid-domain-association, but received a 404 status code from your server. Please check that the file is hosted correctly. Note that our servers most likely send different HTTP headers than your browser; you should check your logs to see why the request failed. For more information, see https://stripe.com/docs/stripe-js/elements/payment-request-button#verifying-your-domain-with-apple-pay.

Once Stripe has validated your domain it will be reflected in the Stripe dashboard:


Search...

Feedback about this page?

On the Web domains

+ Add new domain

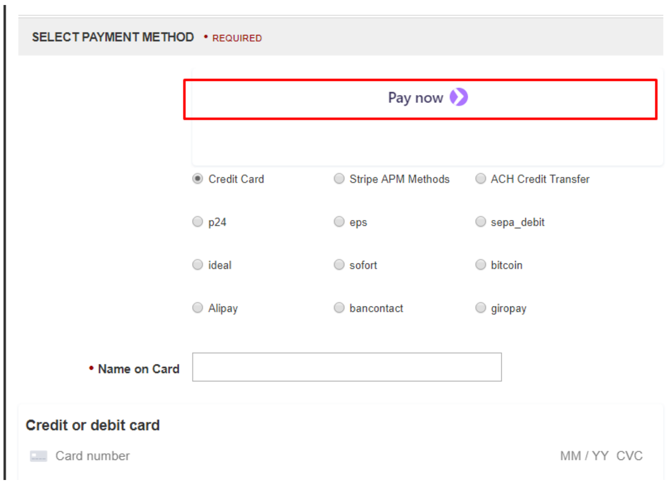
stripe01-tech-prtnr-na05-dw.demandware.net

× Delete

iOS certificates

+ Add new application

The storefront Payment Request Button can now be utilized to transact ApplePay JS transactions as proxied through the Stripe Payment Request Button:


SELECT PAYMENT METHOD • REQUIRED

Pay now

(●) Credit Card ( ) Stripe APM Methods ( ) ACH Credit Transfer

( ) p24 ( ) eps ( ) sepa_debit

( ) ideal ( ) sofort ( ) bitcoin

( ) Alipay ( ) bancontact ( ) giropay

• Name on Card

Credit or debit card

Card number MM / YY CVC

## 3.5 External Interfaces

Stripe functionality relies heavily on external calls to the Stripe services. All external interfaces use the Service Framework to communicate with the Stripe API.

Stripe accounts are free to create and use. Most communications with Stripe services are logged and easily accessible in the Stripe Dashboard (http://dashboard.stripe.com). It is highly encouraged to use the Stripe Dashboard to monitor and test your integration.

## 3.6 Testing

Stripe test values can be found in the Stripe documentation (https://stripe.com/docs/testing). This includes a number of test Credit Cards for use in testing a wide variety of scenarios. However, the test Credit Cards only work while using your test secret and publishable API keys. Further, you cannot use real Credit Card numbers with your test API keys.

Monitoring and testing the integration against the Stripe Dashboard is highly encouraged. Aside from what Credit Card numbers can be used, Stripe functions largely the same with both test and live transactions. Once you've satisfactorily completed and tested your integration, merely change your two Stripe API keys to take your integration live.

# 4. Operations, Maintenance

## 4.1 Data Storage

The Stripe LINK cartridge extends Commerce Cloud to store several data points.

**Customer Profile**: Stripe Customer ID, used to retrieve information about the customer's record in your Stripe account.

**Custom Site Preferences**: noted in detail above.

The Stripe LINK cartridge does work to remove any PCI data that may otherwise be stored on the DWRE system through use of Stripe.js tokenization and custom code to keep credit card data from being submitted to the DWRE servers.

## 4.2 Availability

Please refer to the Stripe Service Level Agreement to determine specific up-times for the service. In case the service fails, there is no fail-over to allow transactions to proceed. Users will instead be provided with friendly error messaging.

## 4.3 Support

For defects or recommendations on improvements, please contact Stripe Support (https://support.stripe.com).

# 5. User Guide

## 5.1 Roles, Responsibilities

There are no recurring tasks required by the merchant. Once configurations and job schedules are set up, the functionality runs on demand.

## 5.2 Business Manager

Business Manager settings and configuration notes are described in detail in the Configurations section.

## 5.3 Storefront Functionality

### 5.3.1 *Credit Card Tokenization*

Stripe.js credit card tokenization requires the inclusion of JavaScript on the payment forms, both during Checkout > Billing as well as My Account > Saved Payment Instruments. Additionally, the credit card ‘type’ form fields are automatically detected and updated rather than requiring user selection.

### 5.3.2 *Saved Credit Cards*

When an authenticated customer selects a saved credit card on the Checkout > Billing page, they will see a list of their Stripe-saved payment Sources as radio buttons rather than the default SiteGenesis <select/> options.

# 6. Known Issues

The LINK Cartridge has no known issues.

## 7. Optional Additional Items

The cartridge also includes starter code and examples for implementing Stripe RELAY features as needed. This includes both a product catalog feed as well as a configuration set for OCAPI on RELAY enabled projects to create/modify/delete basket items, submit payments, and complete orders from social-embedded interfaces. This code is not covered under the certified LINK cartridge but included as a base for reference. Additional scoping and configuration will be required. Contact your Stripe account manager for more details.

The following OCAPI Endpoints would be used in a RELAY Integration:

- Authorize Customer /customers/auth
- Create Basket /baskets
- Set Customer Email (guest checkout) /baskets/{basket_id}/customer
- Add Item to Basket /baskets/{basket_id}/items
- Get Shipping Methods /baskets/{basket_id}/shipments/{shipment_id}/shipping_methods
- Set Shipping Method /baskets/{basket_id}/shipments/{shipment_id}/shipping_method
- Set Shipping Address /baskets/{basket_id}/shipments/{shipment_id}/shipping_address
- Set Billing Address /baskets/{basket_id}/billing_address
- Create Basket Payment Instrument /baskets/{basket_id}/payment_instruments
- Create Order /orders
- Authorize Credit Card /orders/{order_id}/payment_instruments

Please refer to Commerce Cloud's OCAPI documentation regarding OCAPI, how it works, and how to get it set up. In addition, Custom Hooks will be required (samples included) to interact with Stripe cartridge code as needed in your particular use case / application.

## 8. Release History

| **Version** | **Date** | **Changes** |
|---|---|---|
| 18.1.0 | <DATE> | Update to use Stripe elements, sources, payment request button, webhooks and asynchronous payments |
| 16.1.0 | <DATE> | Initial release |