**Stripe Integration Guide**

Version 16.1.0

****



Table of Contents

1. Summary 1-4

2. Component Overview 2-5

2.1 Functional Overview 2-5

2.1.1 Stripe Tokenization and Charges 2-5

2.1.2 Stripe Relay 2-5

2.2 Use Cases 2-5

2.2.1 Stripe.js Tokenization 2-5

2.2.2 Stripe Charges 2-5

2.2.3 AVS Auto-Fail Transactions 2-5

2.2.4 OCAPI-based purchases (Stripe Relay) 2-5

2.3 Limitations, Constraints 2-6

2.4 Compatibility 2-6

2.5 Privacy, Payment 2-6

3. Implementation Guide 3-7

3.1 Setup 3-7

3.1.1 Cartridges 3-7

3.2 Configuration 3-8

3.2.1 Assign Cartridges to Site(s) 3-8

3.2.2 Import System Object Definitions 3-9

3.2.3 Import Services 3-9

3.2.4 Import Custom Job Schedule 3-9

3.2.5 Custom Site Preferences 3-10

3.2.6 Payment Settings Updates 3-10

3.2.7 Product Feed Setup 3-11

3.2.8 OCAPI Settings 3-12

3.3 Custom Code 3-13

3.3.1 Controllers 3-13

3.3.2 Pipelines 3-18

3.3.3 Forms 3-21

3.3.4 Templates 3-22

3.3.5 Scripts 3-25

3.3.6 JavaScript 3-26

3.3.7 CSS 3-28

3.4 External Interfaces 3-30

3.5 Testing 3-30

4. Operations, Maintenance 4-31

4.1 Data Storage 4-31

4.2 Availability 4-31

4.3 Support 4-31

5. User Guide 5-32

5.1 Roles, Responsibilities 5-32

5.2 Business Manager 5-32

5.3 Storefront Functionality 5-32

5.3.1 Credit Card Tokenization 5-32

5.3.2 Saved Credit Cards 5-32

6. Known Issues 6-33

7. Release History 7-34

# Summary

The Stripe LINK Cartridge facilitates integration between a Demandware Storefront and Stripe Payment Services; including Credit Card tokenization via Stripe.js, ability to create charges, and 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. **Import and customize OCAPI settings** for enabling Relay services
4. **Import and Schedule Job Process** for managing your product feed
5. **Apply Custom Code** to the DWRE Storefront
6. **Configure the Storefront Settings** to connect to Stripe services and enable LINK cartridge integration to the storefront

# Component Overview

## Functional Overview

### *Stripe Tokenization and Charges*

Stripe tokenization modifies the default Demandware 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 tokenize any new cards entered by customers. This data is transformed into a Stripe Customer object and an associated payment Source. At the point of purchase, the stored, tokenized data is used to generate a Stripe Charge. Registered Customers can manage (add, delete) Credit Cards in their storefront-connected Stripe Account for re-use in subsequent storefront purchases.

### *Stripe Relay*

Stripe Relay LINK cartridge updates enable Stripe Relay to interact with the Storefront to manage the checkout process from an external system. This includes allowing Stripe Relay to create/modify/delete basket items, submit payments, and complete orders from social-embedded interfaces. For example, Stripe Relay allows customers to purchase goods directly within Twitter. Additionally, Stripe Relay cartridge functionality includes the creation of a Google Product Feed that can be used to import product details into a Stripe Account.

## Use Cases

### *Stripe.js Tokenization*

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

### 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 Customer ID and associated payment Source.

### 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.

### OCAPI-based purchases (Stripe Relay)

Customer can purchase Demandware-based products from Stripe Relay-enabled third party interfaces, including embedded products on Twitter.

## 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.

## Compatibility

Available since Demandware 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.

## Privacy, Payment

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

# Implementation Guide

## 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.

### *Cartridges*

#### 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
   1. stripe.js
2. scripts
   1. hooks/afterPostPaymentInstrument.js
   2. hooks/afterSetShippingAddress.js
   3. hooks/authorize.js
   4. hooks/authorizeCreditCard.js
   5. payment/processor/STRIPE\_CREDIT.js
   6. service/stripe.ds
   7. service/stripeInit.ds
   8. hooks.json
   9. stripeHelper.ds
3. scss
   1. \_stripe.scss
4. static
   1. stripe/jquery.payment.js
5. templates
   1. account/payment/makefdefault.isml
   2. account/payment/makedefaultlink.isml
   3. account/payment/stripeinputs.isml
   4. checkout/billing/stripe\_creditcard\_fields.isml
   5. checkout/billing/stripe\_paymentmethods.isml
   6. checkout/billing/stripecreditcardjson.isml
   7. feed/displayproductfeed.isml
   8. stripe/footerinclude.isml
   9. resources/account.properties
   10. resources/checkout.properties

#### int\_stripe\_controllers

int\_stripe\_controllers contains all the logic for managing the Stripe integration controller logic via Demandware’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
   1. Stripe.js

#### int\_stripe\_pipelines

int\_stripe\_pipelines contains all the logic for managing the Stripe integration controller logic via Demandware’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 Demandware Platform, as of version 16.8, requires Pipeline code for custom scheduled jobs.

1. pipelines
   1. STRIPE\_CREDIT.xml
   2. Stripe.xml
2. scripts
   1. feed/GenerateProductFeed.ds
   2. pipelinescripts/AddCard.ds
   3. pipelinescripts/AuthorizePayment.ds
   4. pipelinescripts/DeleteCard.ds
   5. pipelinescripts/DisplayProductFeeds.ds
   6. pipelinescripts/FetchCards.ds
   7. pipelinescripts/GetCustomerCreditCard.ds
   8. pipelinescripts/IsStripeEnabled.ds
   9. pipelinescripts/MakeDefault.ds
   10. pipelinescripts/RefundCharge.ds

#### Metadata

1. OCAPI Settings.txt
2. stripe\_jobs.xml
3. stripe\_metadata.xml
4. stripe\_services.xml

## Configuration

### Assign Cartridges to Site(s)

#### 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 “:”
   1. 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
   1. 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

#### 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 “:”

### Import System 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

### 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

### 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

### Custom Site Preferences

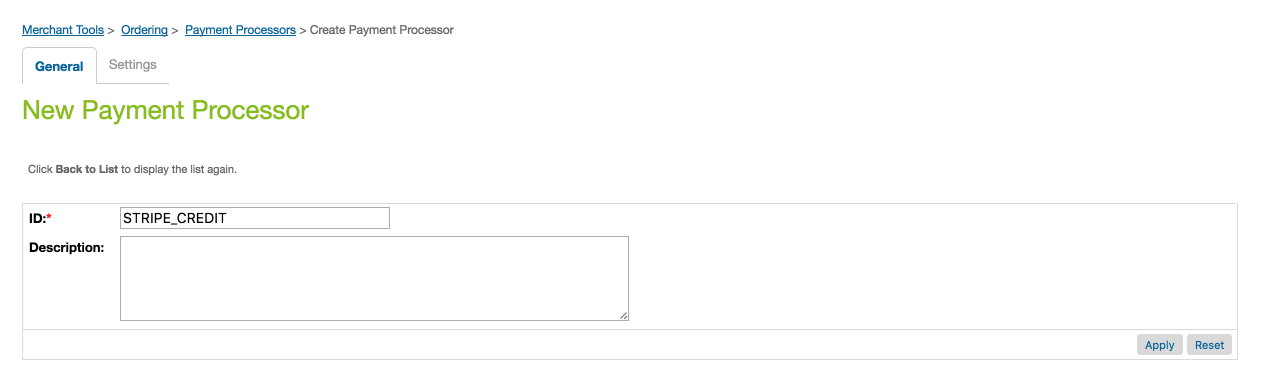
Update the **Merchant Tools > Site Preferences > Custom Site Preferences > Stripe Configurations** with Site specific values.

1. **Stripe Secret API Key**
   1. Can be obtained through the Stripe Dashboard (<https://dashboard.stripe.com>/account/apikeys)
2. **Stripe Publishable API Key**
   1. Find along with Stripe Secret API Key
3. **Capture Funds on Stripe Charge**
   1. Default value: true (Yes)
   2. Set to false (No) to instead Authorize Stripe Charges
4. **AVS Address Fail Statuses**
   1. 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**
   1. 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**
   1. 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**
   1. The directory within the IMPEX WebDAV folder where the Product Feed .tsv file will be saved
8. **Product Feed File Name**
   1. File name for the actual Product Feed File
9. **Product Feed Default Brand Name**
   1. 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.

### Payment Settings Updates

#### Add New Payment Processor

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".



#### 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

#### Credit Card Support

The base LINK Cartridge code contains support for all credit cards supported by Stripe. However, if you need to make updates to how the OMS handles credit card ‘type’ values and/or how they 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
2. int\_stripe/cartridge/templates/resources/checkout.properties

### Product Feed Setup

#### 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.

#### 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.

#### 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.

### OCAPI Settings

Stripe Relay relies on DWRE’s Open Commerce API (OCAPI). Starter settings are included in the Stripe LINK Cartridge in metadata/OCAPI Settings.txt. Please refer to Demandware’s OCAPI documentation regarding OCAPI, how it works, and how to get it set up: <https://documentation.demandware.com/DOC1/topic/com.demandware.dochelp/OCAPI/16.8/usage/GettingStartedWithOCAPI.html>

#### Steps to import and manage OCAPI Settings

1. Copy the contents of metadata/OCAPI Settings.txt into Business Manager.
   1. Navigate to **Administration > Site Development > Open Commerce API Settings**
   2. Open the “OCAPI Settings.txt” file and update the “client\_id”: “[Your OCAPI Client ID]” property to use a valid client ID created through Demandware Accounts.
   3. Copy and paste the contents from the “OCAPI Settings.txt” file to the textarea in Business Manager.
2. Update the settings
   1. Customize the various property values in the JSON object to modify security and/or limit access to the OCAPI endpoints.

#### Endpoints used for the 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

#### Custom Hooks

The Stripe LINK cartridge provides hooks for several of the OCAPI methods to help with integrating with Site customizations.

1. afterPostPaymentinstrument
   1. An empty placeholder hook for use in customizations post payment authorizations
2. afterPutShippingAddress
   1. An empty placeholder hook for use in customizations after a shipping address is added to a basket. This can be used to add custom tax calculation logic or other customizations specifically dependent on the basket shipping address.
3. authorize
   1. The hook used to authorize payments against a Stripe Token. Can also be extended with any custom authorization logic.
4. authorizeCreditCard
   1. The hook used to authorize payments against a Stripe Token. Can also be extended with any custom authorization logic.

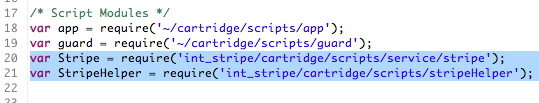
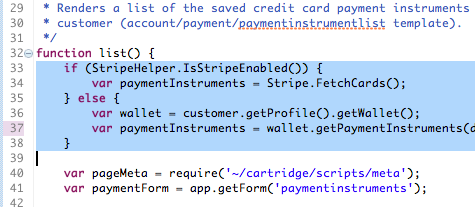
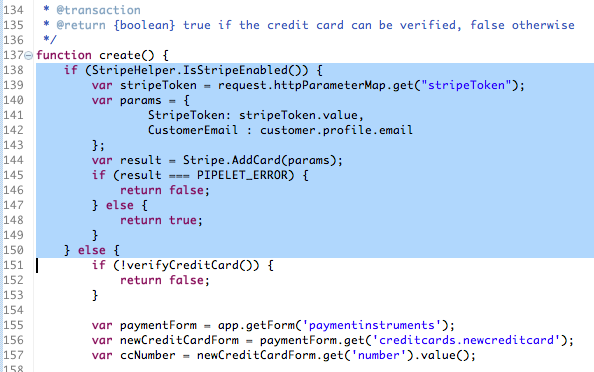
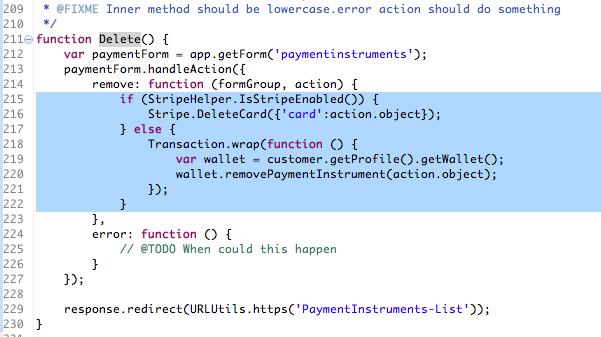
## Custom Code

Make the following updates to the Storefront Code. Examples provided are based on SiteGenesis 16.8 (103.0.0). The initial integration effort should take from ½ to 2 days, based on a SiteGenesis installation.

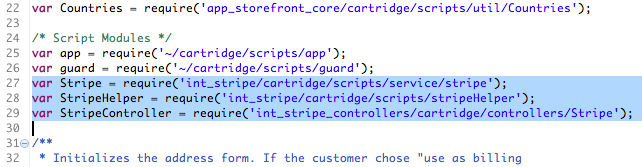
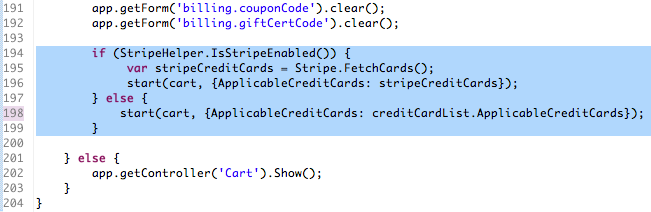
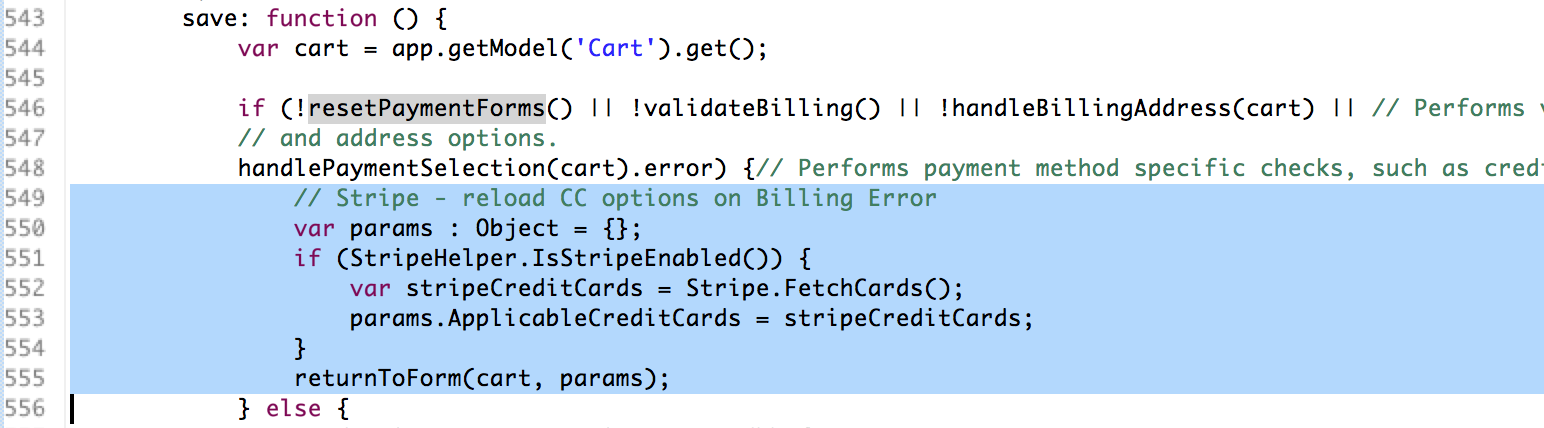
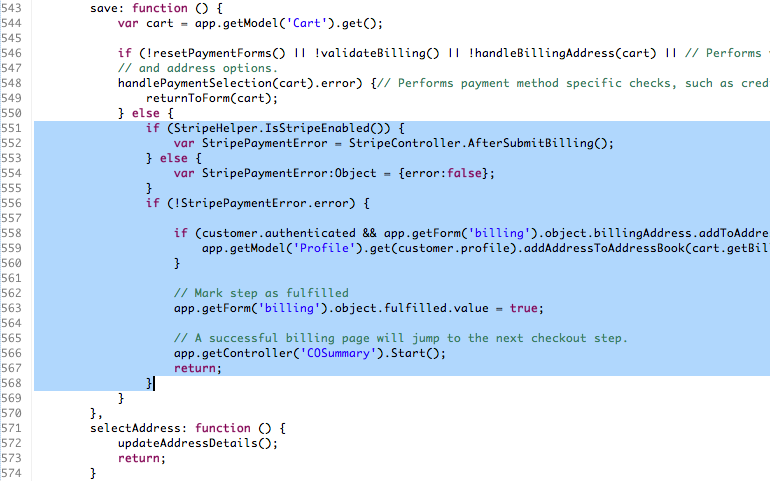
### Controllers

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

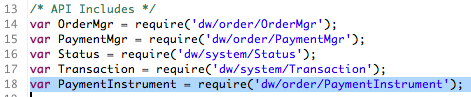
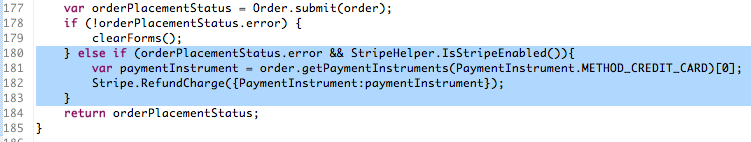
**Update the controller PaymentInstruments.js with the following custom code**

1. Add Stripe Services and StripeHelper to the Script Module includes   
     
   **var** Stripe = require('int\_stripe/cartridge/scripts/service/stripe');  
   **var** StripeHelper = require('int\_stripe/cartridge/scripts/stripeHelper');  
     
   
2. Replace the following code in the list() function:  
     
   **var** wallet = customer.getProfile().getWallet();  
   **var** paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  
     
   With:  
     
   **if** (StripeHelper.IsStripeEnabled()) {  
    **var** paymentInstruments = Stripe.FetchCards();  
   } **else** {  
    **var** wallet = customer.getProfile().getWallet();  
    **var** paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  
   }  
     
     
   
3. Add the following code to the beginning of the create() function  
     
   **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** {  
    // insert default code from original create() function  
   }  
     
   
4. Replace the following code in the Delete() function:  
     
   Transaction.wrap(**function** () {  
    **var** wallet = customer.getProfile().getWallet();  
    wallet.removePaymentInstrument(action.object);  
   });  
   With:  
     
   **if** (StripeHelper.IsStripeEnabled()) {  
    Stripe.DeleteCard({'card':action.object});  
   } **else** {  
    Transaction.wrap(**function** () {  
    **var** wallet = customer.getProfile().getWallet();  
    wallet.removePaymentInstrument(action.object);  
    });  
   }  
     
   

**Update the controller COBilling.js to add the following code**

1. Add Stripe Services and StripeHelper to the Script Module includes  
     
   **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');  
     
   ****
2. Replace the following code in the publicStart() function  
     
   start(cart, {ApplicableCreditCards: creditCardList.ApplicableCreditCards});  
   With: **if** (StripeHelper.IsStripeEnabled()) {  
    **var** stripeCreditCards = Stripe.FetchCards();  
    start(cart, {ApplicableCreditCards: stripeCreditCards});  
   } **else** {  
    start(cart, {ApplicableCreditCards: creditCardList.ApplicableCreditCards});  
   }  
     
   
3. Replace the following code in the billing() function, save: function() object  
     
   returnToForm(cart);  
     
   With:  
     
   // Stripe - reload CC options on Billing Error  
   **var** params : Object = {};  
   **if** (StripeHelper.IsStripeEnabled()) {  
    **var** stripeCreditCards = Stripe.FetchCards();  
    params.ApplicableCreditCards = stripeCreditCards;  
   }  
   returnToForm(cart, params);  
     
   
4. Add the following code to the billing() function, save: function() object  
     
   **if** (StripeHelper.IsStripeEnabled()) {  
    **var** StripePaymentError = StripeController.AfterSubmitBilling();  
   } **else** {  
    **var** StripePaymentError:Object = {error:**false**};  
   }  
   **if** (!StripePaymentError.error) {  
    // Default SiteGenesis Code  
   }  
     
   

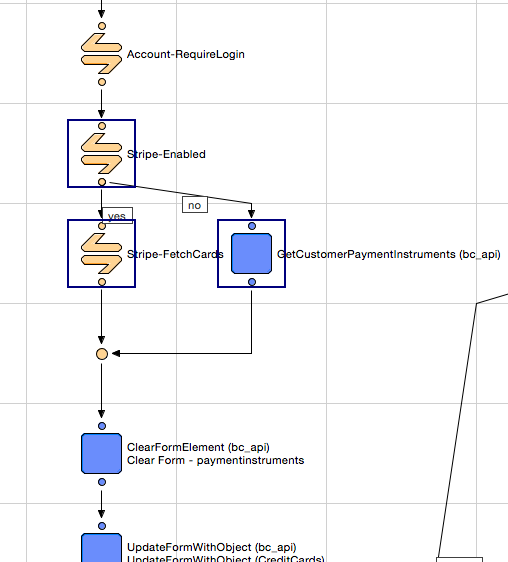
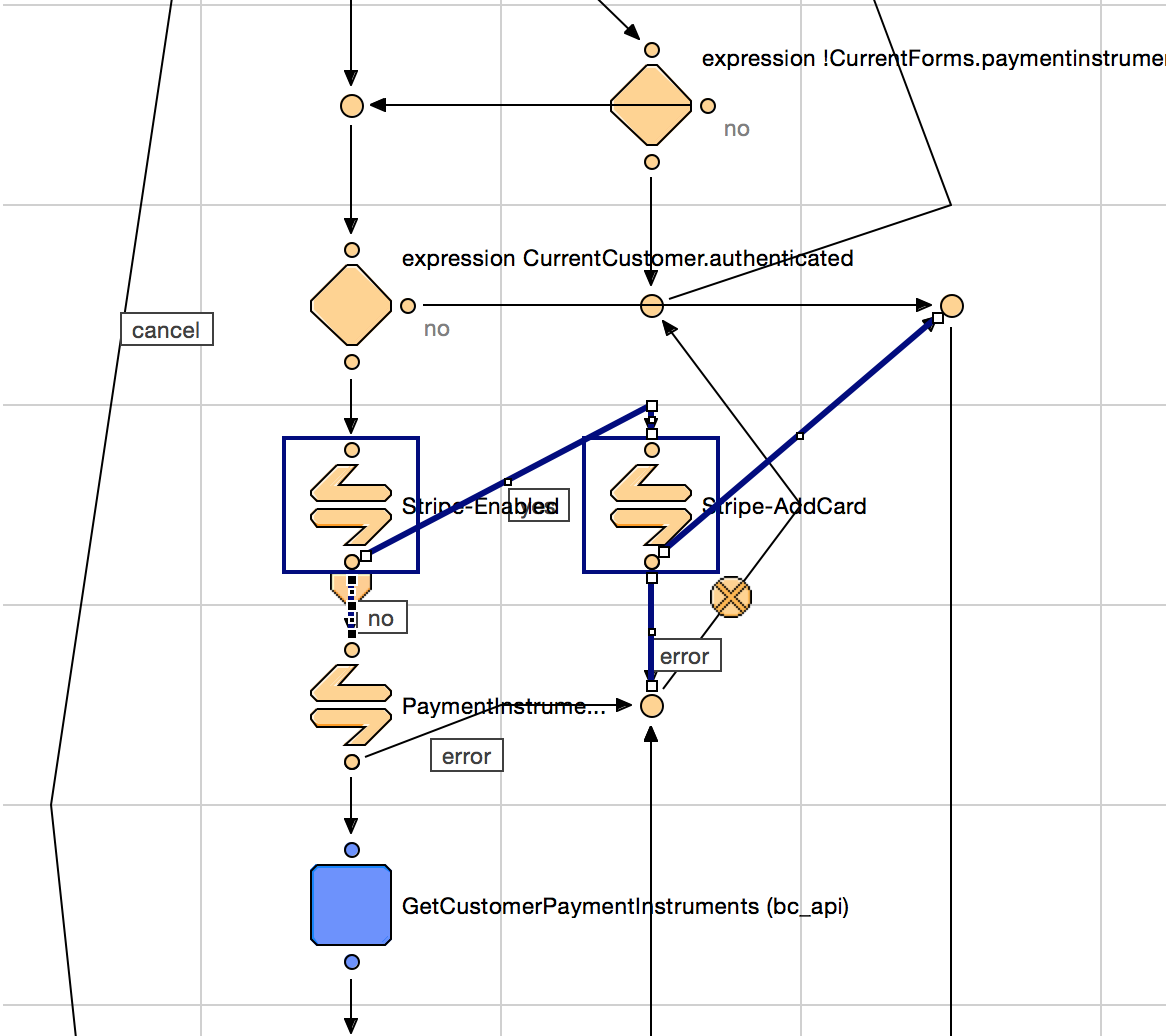
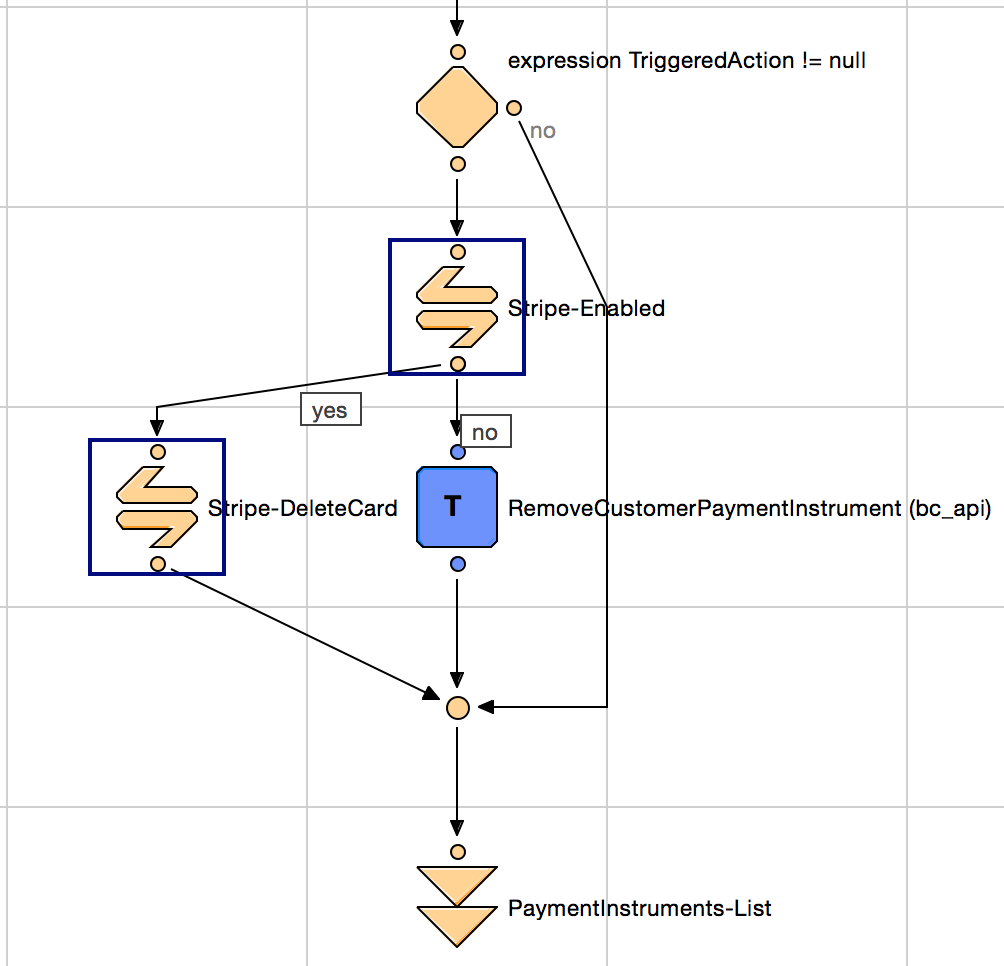
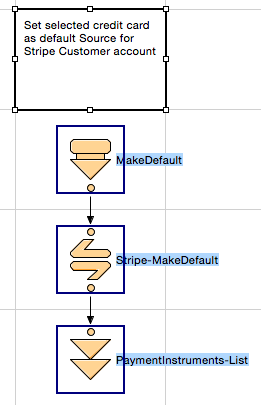
**Update the controller COPlaceOrder.js to add the following code**

1. Add PaymentInstrument to the API Includes section  
     
   **var** PaymentInstrument = require('dw/order/PaymentInstrument');  
     
   
2. Add Stripe Services and StripeHelper to the Scripts Modules section  
     
   **var** Stripe = require('int\_stripe/cartridge/scripts/service/stripe');  
   **var** StripeHelper = require('int\_stripe/cartridge/scripts/stripeHelper');  
     
   
3. Add the following code in the start() function just before the return statement  
     
   } **else** **if** (orderPlacementStatus.error && StripeHelper.IsStripeEnabled()){  
    **var** paymentInstrument = order.getPaymentInstruments(PaymentInstrument.METHOD\_CREDIT\_CARD)[0];  
    Stripe.RefundCharge({PaymentInstrument:paymentInstrument});  
   }  
     
   

### Pipelines

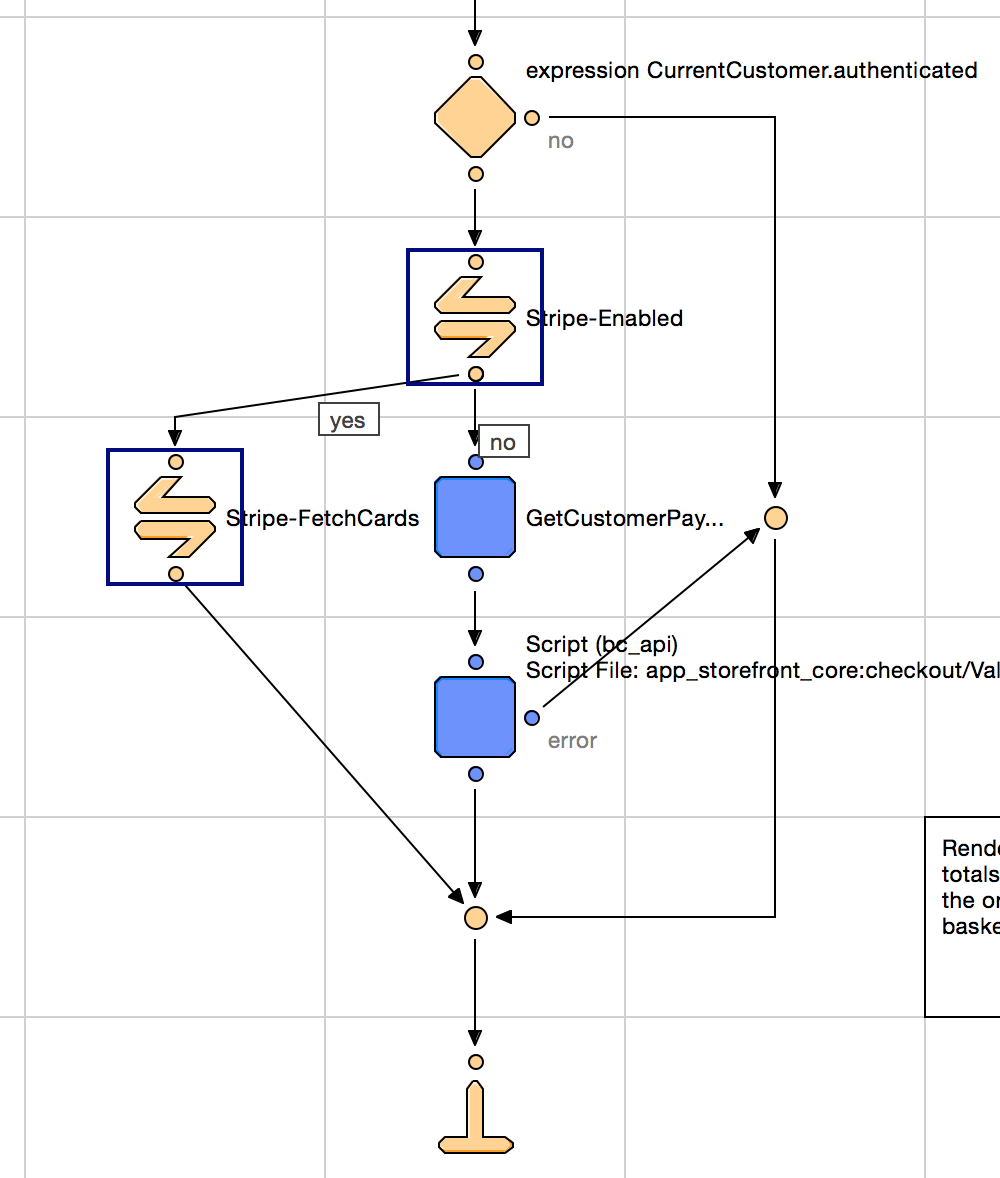
Pipeline updates are only required if integrating with Pipelines instead of using Controllers

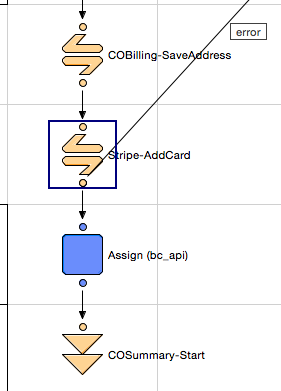
**Update PaymentInstruments.xml with the following customizations**

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

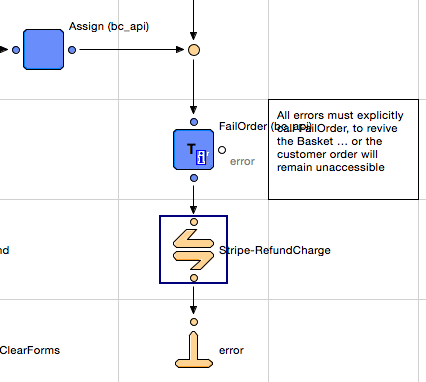
**Update COBilling.xml with the following customizations**

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



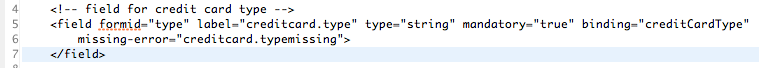
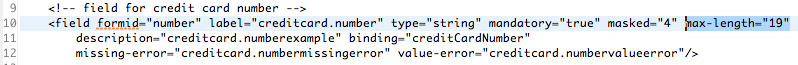
1. Add a call node to the Stripe-AddCard in the COBilling-Start pipeline just after the COBilling-SaveAddress call node. Send an ‘error’ connector to the standard error direction  
     
   

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



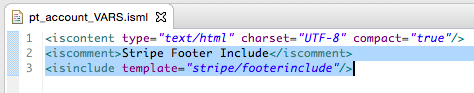
### Forms

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

1. Remove the credit card <options/> element: <options optionid-binding="cardType" value-binding="cardType" label-binding="name"/>  
     
   
2. Increate the max length for the creditcard.number field from 16 to 19  
     
   

### Templates

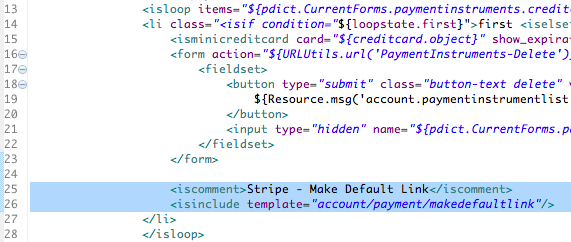
**Update *account/pt\_account\_VARS.isml* and *checkout/pt\_checkout\_VARS.isml* to add the Stripe JavaScript page includes.**  
<iscomment>Stripe Footer Include</iscomment>  
<isinclude template=*"stripe/footerinclude"*/>



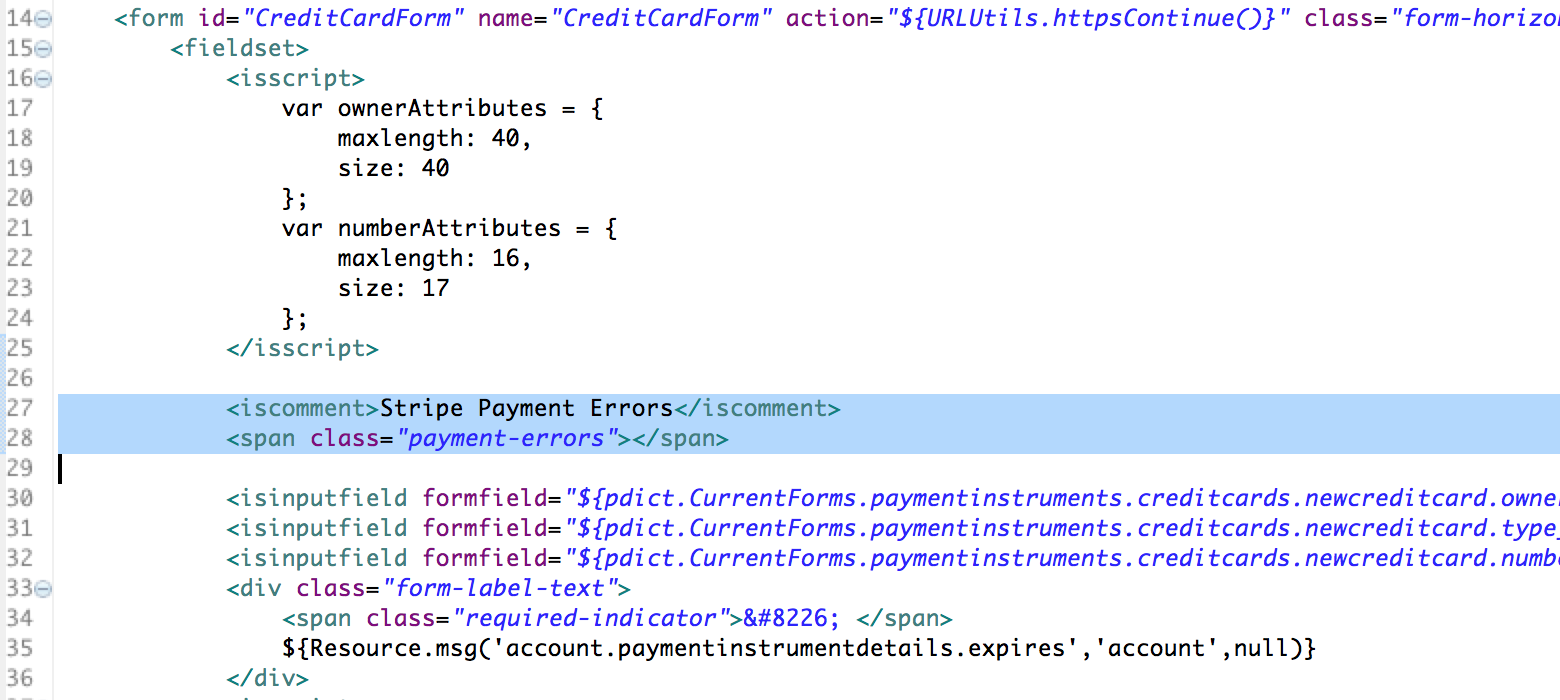
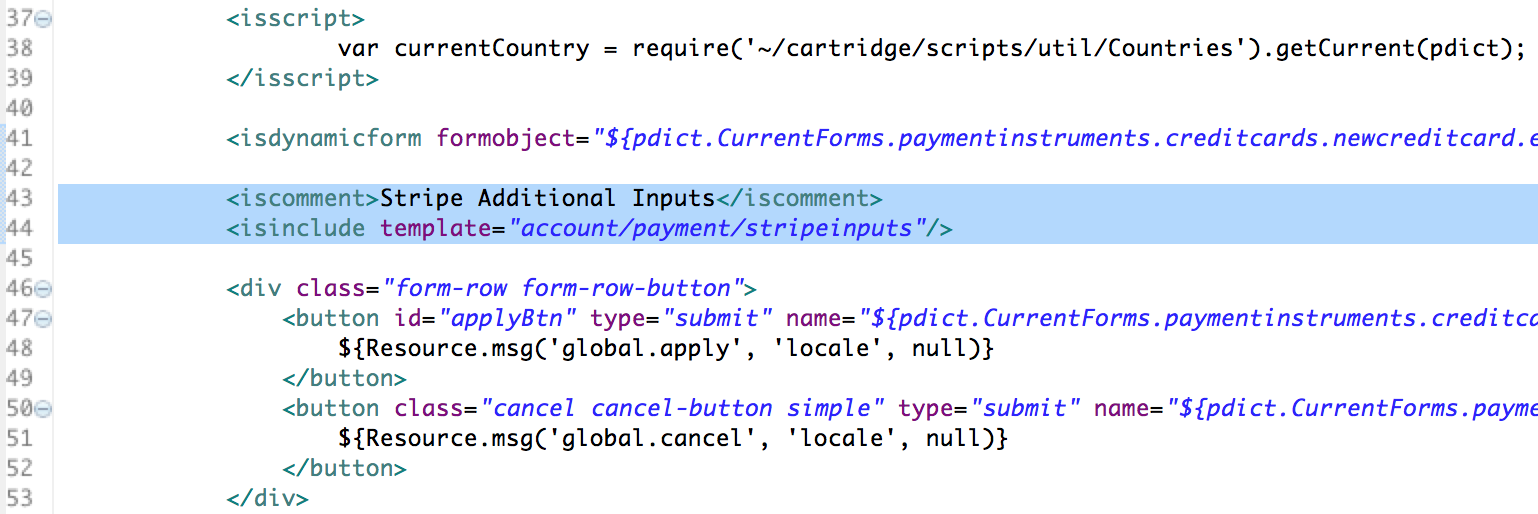
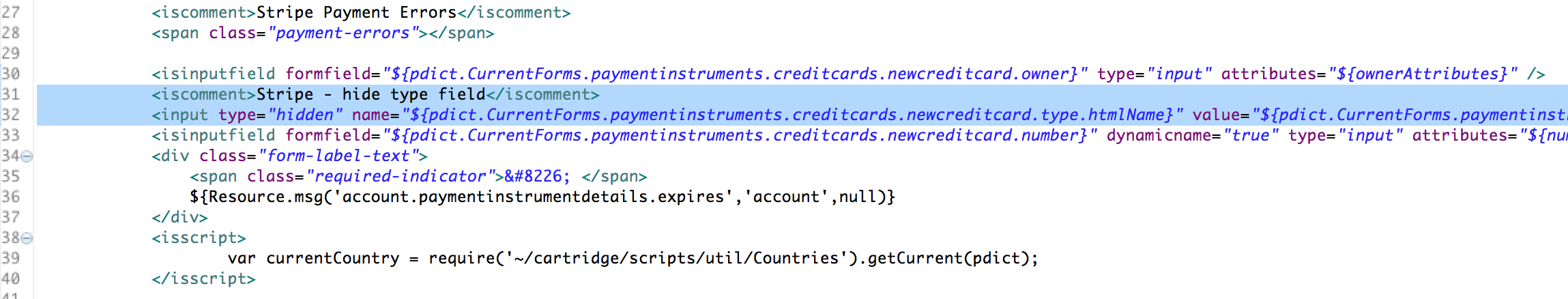
**Update *account/payment/paymentinstrumentlist.isml* to add code for setting a card as a default card. Add just before the closing </li> tag within the <isloop/> element**

<iscomment>Stripe - Make Default Link</iscomment>

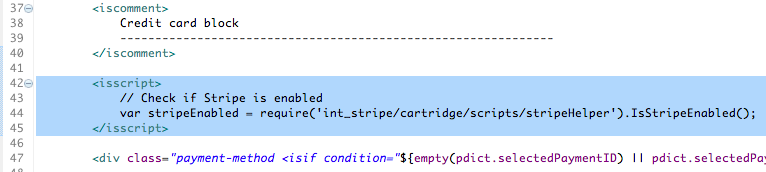
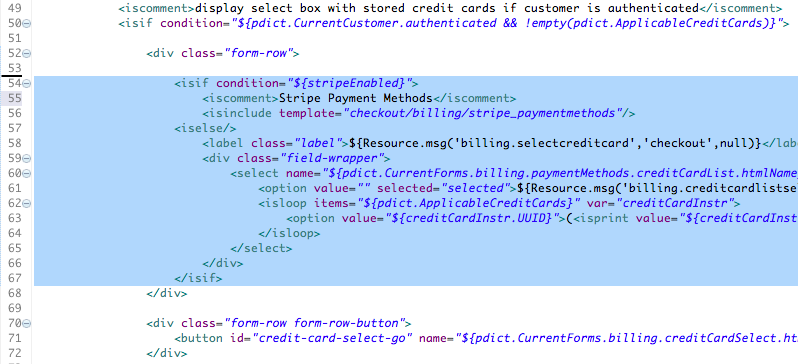
<isinclude template=*"account/payment/makedefaultlink"*/>



**Make the following updates to *account/payment/paymentinstrumentdetails.isml***

1. Add <span/> tag at the top of the Credit Card form  
     
   <iscomment>Stripe Payment Errors</iscomment>  
   <span class=*"payment-errors"*></span>  
     
   Example below shows placement above the owner field, but it can be located anywhere according to design   
     
   
2. Add additional input fields just before the Apply and Cancel buttons  
     
   <iscomment>Stripe Additional Inputs</iscomment>  
   <isinclude template=*"account/payment/stripeinputs"*/>  
     
   
3. Replace the the newcreditcard.type input field with the following code  
     
   <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}"*/>  
     
   

**Make the following updates to *checkout/billing/paymentmethods.isml***

1. Add a check at the start of the Credit Card block to see if Stripe is enabled  
     
   <isscript>  
    // Check if Stripe is enabled  
    var stripeEnabled = require('int\_stripe/cartridge/scripts/stripeHelper').IsStripeEnabled();  
   </isscript>  
     
   
2. Add a check to see if Stripe is enabled and, if so, reference the checkout/billing/stripe\_paymentmethods.isml include  
     
   <isif condition=*"${stripeEnabled}"*>  
    <iscomment>Stripe Payment Methods</iscomment>  
    <isinclude template=*"checkout/billing/stripe\_paymentmethods"*/>  
   <iselse/>  
    // Default Site Genesis / CC Saved Cards code  
   </isif>  
     
   
3. Add a check for Stripe enabled and, if so, reference the checkout/billing/stripe\_credit\_cardfields.isml include. This will replace the CC form fields with fields from the cartridge.  
     
   <isif condition=*"${stripeEnabled}"*>  
    <iscomment>Stripe replacement CC fields</iscomment>  
    <isinclude template=*"checkout/billing/stripe\_creditcard\_fields"*/>  
   <iselse/>  
    // Default Site Genesis / CC form fields code  
   </isif>  
     
   

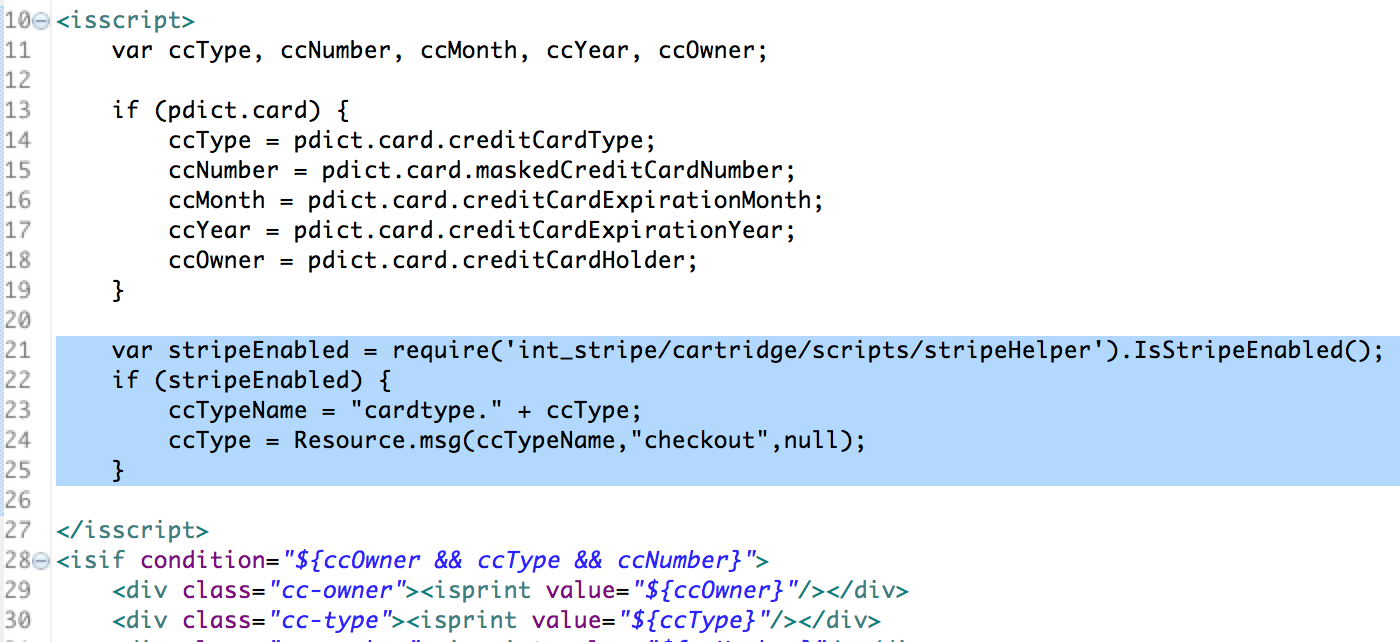
**Update *account/payment/minicreditcard.isml* to read Stripe Names for Credit Card Types in the top <isscript/> tag**

var stripeEnabled = require('int\_stripe/cartridge/scripts/stripeHelper').IsStripeEnabled();

if (stripeEnabled) {

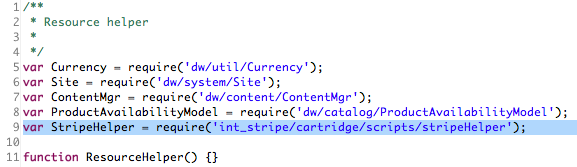
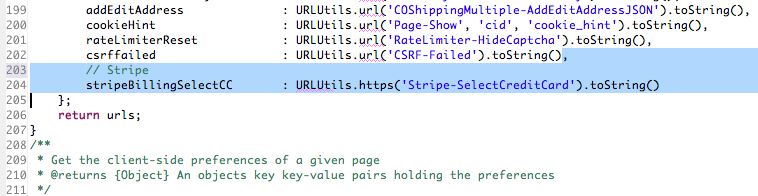
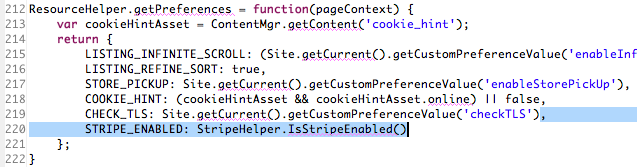
ccTypeName = "cardtype." + ccType;

ccType = Resource.msg(ccTypeName,"checkout",null);

}  
  


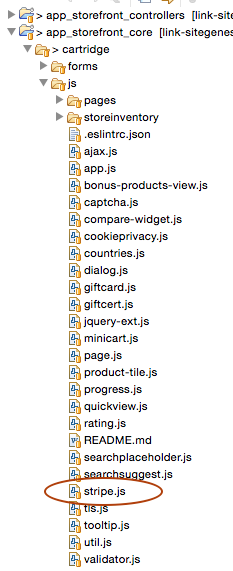
### Scripts

**Update *util/Resource.ds* with the following code**

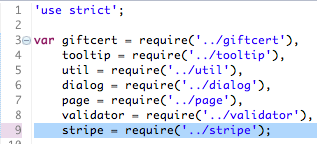
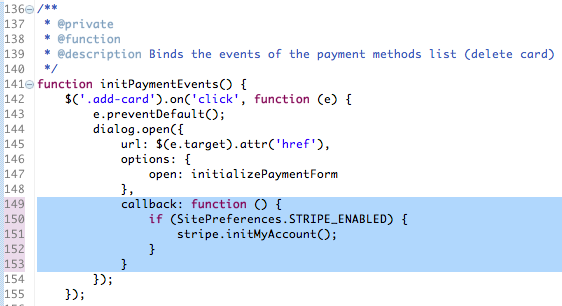
1. Add StripeHelper to the top require section  
     
   **var** StripeHelper = require('int\_stripe/cartridge/scripts/stripeHelper');  
     
   
2. Add the following code to add the stripeBillingSelectCC URL to the end of the urls section  
     
   // Stripe  
   stripeBillingSelectCC : URLUtils.https('Stripe-SelectCreditCard').toString()  
     
   
3. Add the following code to the end of the preferences section  
     
   STRIPE\_ENABLED: StripeHelper.IsStripeEnabled()  
     
   

### JavaScript

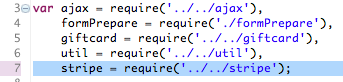
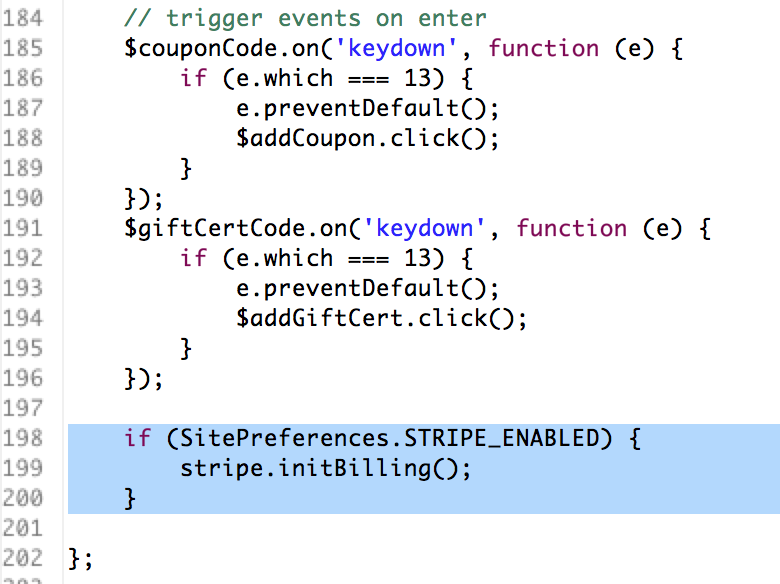
**Add custom Stripe JavaScript to the Storefront by copying js/stripe.js from the int\_stripe cartridge to the js/ folder of the Storefront Cartridge**



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

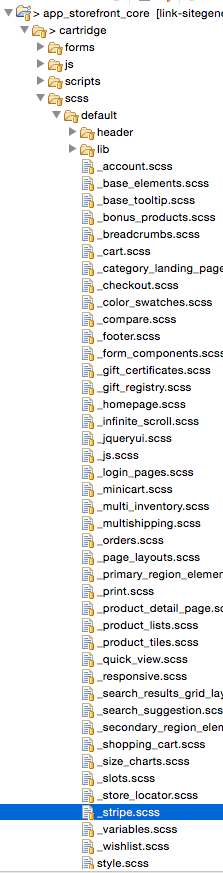
1. Add stripe object reference  
     
   stripe = require('../stripe');  
     
   
2. Add the following code to the initPaymentEvents() function as an additional property in the $(‘.add-card’) event  
     
   callback: **function** () {  
    **if** (SitePreferences.STRIPE\_ENABLED) {  
    stripe.initMyAccount();  
    }  
   }  
     
   

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

1. Add the stripe object reference to the require section  
     
   stripe = require('../../stripe');  
     
   
2. Add the following code to the end of the exports.init function  
     
   **if** (SitePreferences.STRIPE\_ENABLED) {  
    stripe.initBilling();  
   }  
     
   

### CSS

Add \_stripe.scss styles to the storefront

1. Copy/paste int\_stripe/cartridge/scss/\_stripe.css to [storefront\_cartridge]/cartridge/scss/\_stripe.css  
     
   
2. Update [storefront\_cartridge]/cartridge/scss/style.scss to reference \_stripe.scss  
     
   // stripe  
   @import "stripe";  
     
   
3. At this point the styles can be customized to match any required site theming

## 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.

## 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.

# Operations, Maintenance

## Data Storage

The Stripe LINK cartridge extends Demandware 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.

## 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.

## Support

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

# User Guide

## Roles, Responsibilities

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

## Business Manager

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

## Storefront Functionality

### 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.

### 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.

# Known Issues

The LINK Cartridge has no known issues.

# Release History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| 16.1.0 | <DATE> | Initial release |