STRIPE INTEGRATION

20.1.0

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://dashbo ard.stripe.com. Please contact your Stripe Implementation Consultant for help with taking your Stripe account live.

The integration encompasses the deployment of several cartridges and modification of storefront code

2. Component Overview

Functional Overview

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.

Use Cases

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.

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.

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.
Supported payment methods:

- Cards (Visa, Mastercard, American Express, Discover, Diners Club, JCB) Alipay
- The Payment Request Button Element gives you a single integration for Apple Pay, Google Pay, Microsoft Pay, and the browser standard Payment Request API.

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

For any locale specific restrictions, please refer to https://stripe.com/docs/js.

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 Controller and SFRA SiteGenesis implemenations.

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

Setup of Business Manager

The Stripe LINK Cartridge contains several cartridges that are required for full functionality. Additionally, Controller and SFRA 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.

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. *For Controllers "app_stripe_controllers:app_stripe_core:int_stripe_controllers:int_stripe_core" need to be added to the cartridge path*
5. *Repeat steps 2 – 4 for each Storefront Site where Stripe will be implemented*

Business Manager Cartridge Assignment

1. *Navigate to Administration > Sites > Manage Sites - Click on the Business Manager Site > "Manage the Business Manager site." Link*
2. *Add "int_stripe_core" to the Cartridges: path*

Metadata import

1. *Navigate to the metadata folder of the project and open the stripe_site_template folder.*
2. *Open the sites folder and edit the 'siteIDHere' folder to the site ID of the site you want.*
3. *Add a folder for each site you need stripe on.*
4. *Navigate to Administration > Site Development > Site Import & Export*
5. *Zip the stripe_site_template folder and import it.*

Building Stripe Styling

Copy app_stripe_core/cartridge/scss/default/stripe.scss into your app_storefront_core cartridge. Import the file on the bottom of style.scss and compile the CSS.

Add New Payment Processors

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

If using Stripe credit 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".

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

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

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

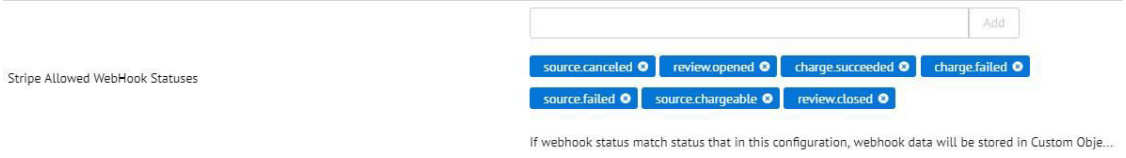
To utilize the Stripe Payment Request Button, enable the "STRIPE_PAYMENT_REQUEST_BTN" payment method. See https://stripe.com/docs/stripe-js/elements/payment-request-button

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. Is this SFRA installation. Set to yes if the current site is using the Storefront Reference Architecture (SFRA)
4. Capture Funds on Stripe Charge a. Default value: true (Yes) b. Set to false (No) to instead Authorize Stripe Charges
5. 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" } }
6. Stripe API URL - https://js.stripe.com/v3/
7. 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-requestbutton#styling-the-element
8. 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.
9. 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-requestbutton# create-payment-request-instance
10. 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.
11. Stripe allowed Webhook Statuses i. Configure the allowed statuses for Webhooks to respond to.

Set to:

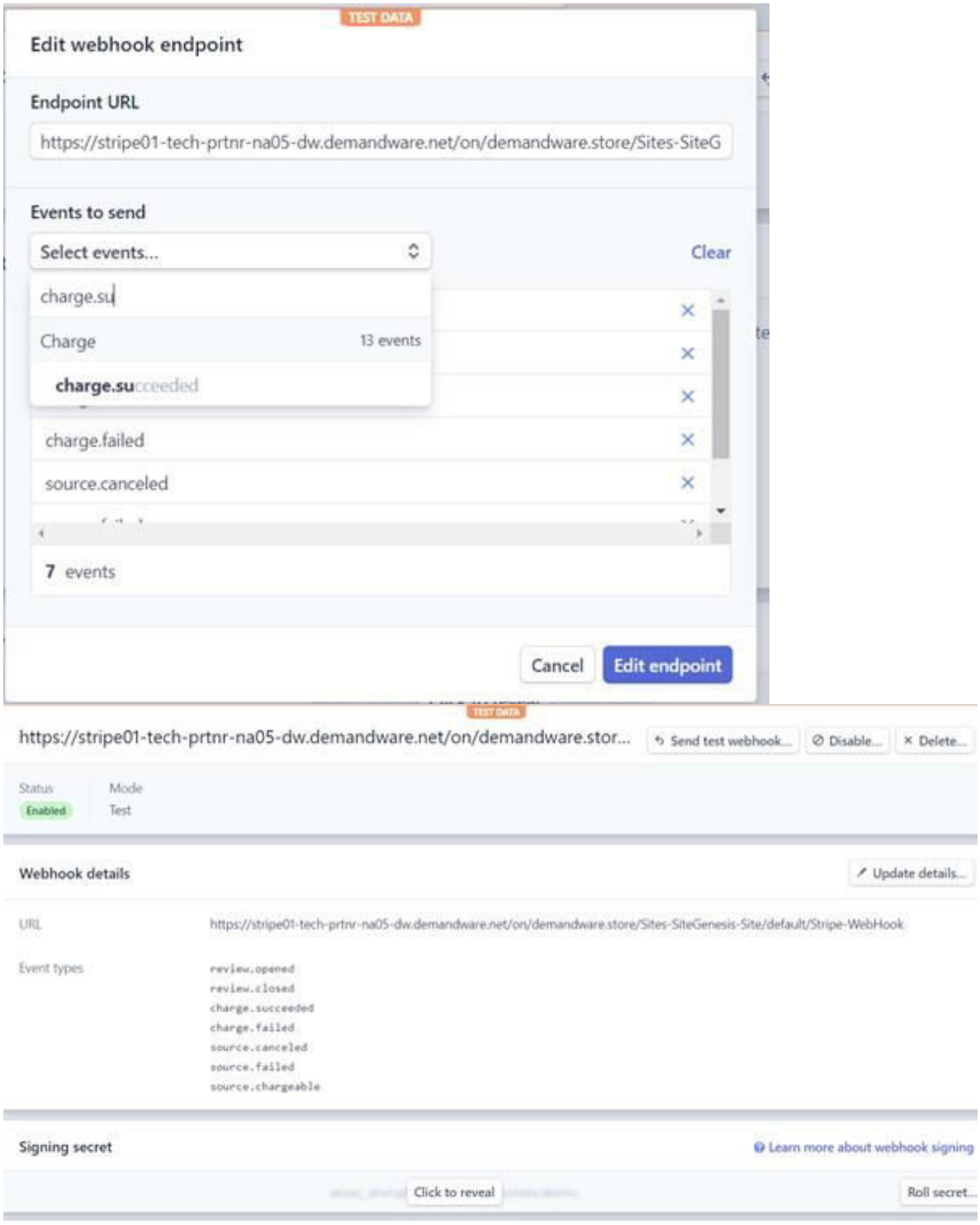
- review.opened
- review.closed
- charge.succeeded
- charge.failed
- source.canceled
- source.failed
- source.chargeable



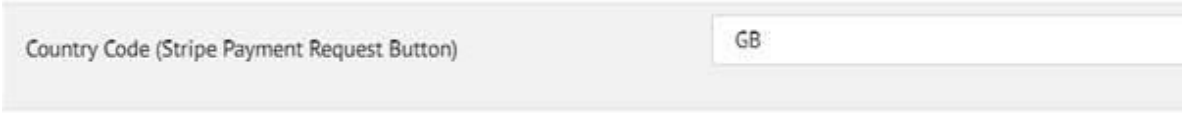
12. Allowed APM Methods a. Update this field, per site locale, to indicate which alternate payment methods are enabled for each locale. Enumeration of allowed Payment Methods from the Stripe API. See more here: https://stripe.com/docs/sources { "default": ["p24", "eps", "sepa_debit", "ideal", "sofort", "bitcoin", "alipay", "bancontact", "giropay"], "en_UK": ["p24", "eps"], "de_AT": ["sofort", "ideal"] }
13. Stripe Enabled – Enables or disables the cartridge

In the Stripe Dashboard (https://dashboard.stripe.com/test/webhooks) enable webhooks, point it to `Stripe-WebHook` controller and subscribe to these events:

- review.opened
- review.closed
- charge.succeeded
- charge.failed
- source.canceled
- source.failed
- source.chargeable



Then copy the signing secret to the 'Stripe Webhook Signing Secret' preference.
Make sure that this value is set to your Stripe account country code:



For ApplePay to work, the file RedirectURL.js must be changed with this code:

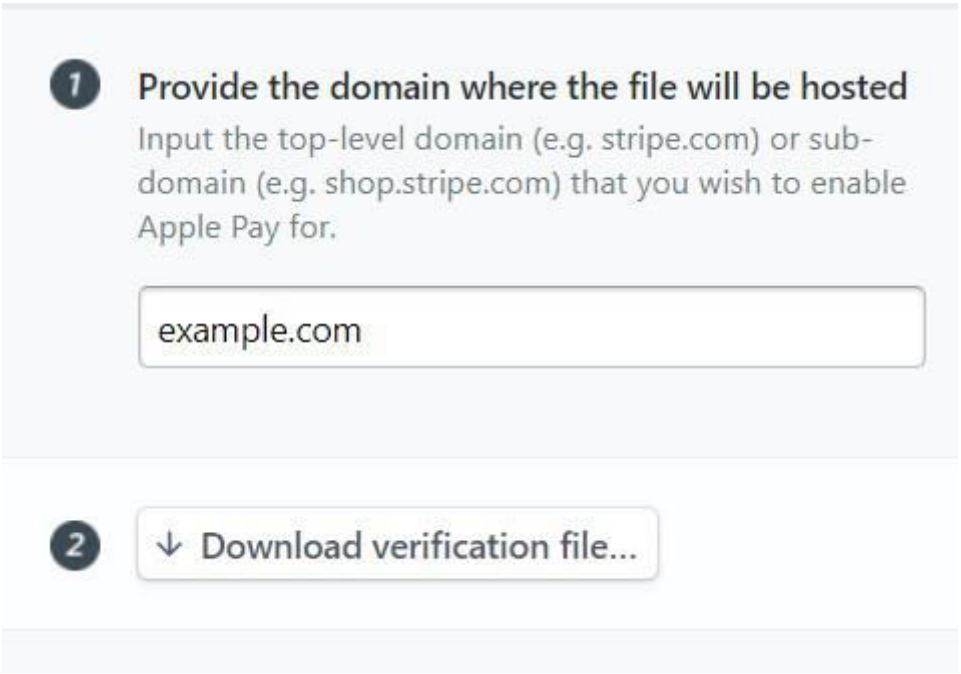
```
    if (URLRedirectMgr.getRedirectOrigin() === '/.well-known/apple-developer-merchantid-domain-association') { // Intercept the incoming path request
        app.getView().render('stripe/util/apple');
        return;
    }
```

```
/**
 * Gets the redirect. Renders the template for a redirect (util/redirectpermanent template). If no redirect can be found,
 * renders an error page (util/redirecterrorutil/redirecterror template).
 */
function start() {

    //Stripe change BEGIN
    if (URLRedirectMgr.getRedirectOrigin() === '/.well-known/apple-developer-merchantid-domain-association') { // Intercept the incoming path request
        app.getView().render('stripe/util/apple');
        return;
    }
    //Stripe change END

    var redirect = URLRedirectMgr.getRedirect(),
        location = redirect ? redirect.getLocation() : null;

    if (!location) {
        response.setStatus(410);
        app.getView().render('util/redirecterror');
    } else {
        app.getView({
            Location: location
        }).render('util/redirectpermanent');
    }
}
```

Then you then need to set an alias to one of the sites on the sandbox temporarily so the stripe dashboard can verify the domain. The alias needs to be something like this:

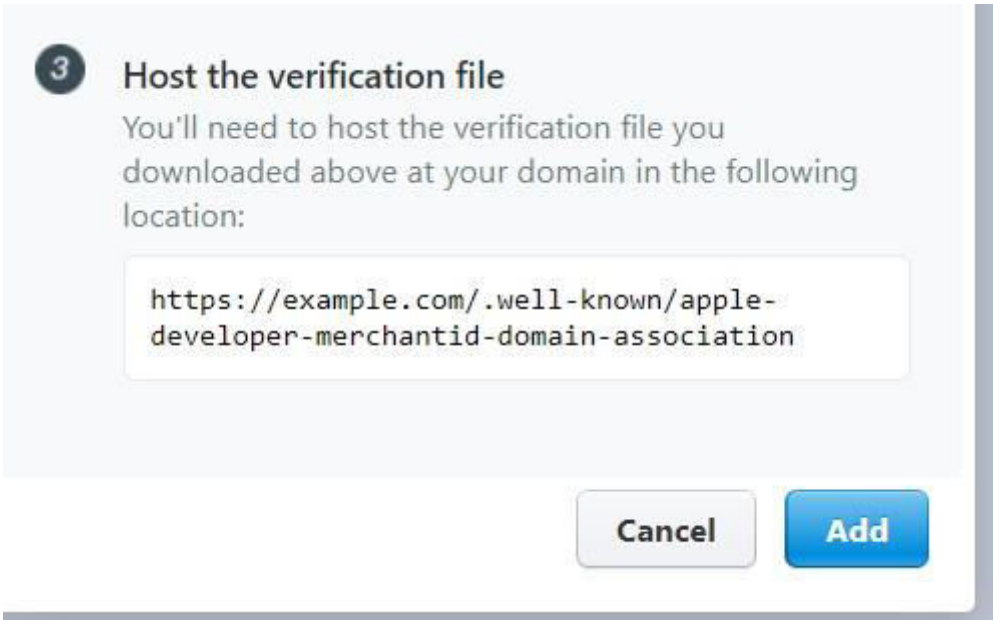
```
{
 "__version": "1",
    "settings": {
        "http-host": "your.sandbox.domain.demandware.net",
        "https-host": "your.sandbox.domain.demandware.net",
        "default":"true",
        "site-path": "/"
    },
    "your.sandbox.domain.demandware.net": [
        {
            "locale": "en_GB",
            "if-site-path": "/"
        }
    ]
}
```

The locale value needs to be a locale that is not disabled.
Then go to https://dashboard.stripe.com/account/apple_pay and click on ‘Add new domain’ button. Enter the domain and download the verification file:



Copy the contents of the file to 'ApplePay Verification String' custom preference:

Then click on the 'Add' button:



Custom Code

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 Tools > Ordering > Payment Methods > Credit/Debit Cards).

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.

Controller: COBilling.js
app_storefront_controllers/cartridge/controllers/COBilling.js

Add the following if statement to function initCreditCardList()

```
//Stripe change BEGIN
    var stripeHelper = require('int_stripe_core').getStripeHelper();
    if (stripeHelper.isStripeEnabled()) {
        applicablePaymentMethods =
stripeHelper.getStripePaymentMethods(applicablePaymentMethods, request.locale);
    }
    //Stripe change END
```

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

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

    applicablePaymentMethods = PaymentMgr.getApplicablePaymentMethods(customer, countryCode, paymentAmount.value);
    applicablePaymentCards = PaymentMgr.getPaymentMethod(PaymentInstrument.METHOD_CREDIT_CARD).getApplicablePaymentCards(customer, countryCode, paymentAmount.value);

    //Stripe change BEGIN
    var stripeHelper = require('int_stripe_core').getStripeHelper();
    if (stripeHelper.isStripeEnabled()) {
        applicablePaymentMethods = stripeHelper.getStripePaymentMethods(applicablePaymentMethods, request.locale);
    }
    //Stripe change END

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

    applicableCreditCards = null;

    if (customer.authenticated) {
        var profile = app.getModel('Profile').get();
        if (profile) {
            applicableCreditCards = profile.validateWalletPaymentInstruments(countryCode, paymentAmount.getValue()).ValidPaymentInstruments;
        }
    }

    return {
        ApplicablePaymentMethods: applicablePaymentMethods,
        ApplicableCreditCards: applicableCreditCards
    };
}
```

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.
    returnToForm(cart);
  } else {
    if (customer.authenticated &&
app.getForm('billing').object.billingAddress.addToAddressBook.value) {

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

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

    // A successful billing page will jump to the next checkout step.
    //Stripe changes BEGIN
    if (request.httpParameterMap.isParameterSubmitted('stripe_redirect_url') &&
!empty(request.httpParameterMap.stripe_redirect_url.stringValue)) {

 response.redirect(request.httpParameterMap.stripe_redirect_url.stringValue);
    } else {
        require('./COSummary').Start();
    }
    //Stripe changes END
    return;
  }
});
```

Controller: COPlaceOrder.js
app_storefront_controllers/cartridge/controllers/COPlaceOrder.js

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

Later down the method after the “else if” block for missing payment info is closed add the following code:

```
    var isAPMOrder = stripeCheckoutHelper.isAPMOrder(order);
    if (!isAPMOrder) {

        var stripePaymentInstrument =
stripeCheckoutHelper.getStripePaymentInstrument(order);

        if (stripePaymentInstrument &&
order.custom.stripeIsPaymentIntentInReview) {
            return {
                Order: order,
                order_created: true
            };
        } else {
            var orderPlacementStatus = Order.submit(order);
            if (!orderPlacementStatus.error) {
                clearForms();
            } else {
                stripeCheckoutHelper.refundCharge(order);
            }
            return orderPlacementStatus;
        }
    } else {
        const Email = app.getModel('Email');
        const Resource = require('dw/web/Resource');
        Email.sendMail({
            template: 'stripe/mail/orderreceived',
            recipient: order.getCustomerEmail(),
            subject: Resource.msg('order.ordercreceived-email.001', 'stripe',
null),
            context: {
                Order: order
            }
        });

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

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

    // Stripe changes BEGIN
    var isAPMOrder = stripeCheckoutHelper.isAPMOrder(order);
    if (!isAPMOrder) {

        var stripePaymentInstrument = stripeCheckoutHelper.getStripePaymentInstrument(order);

        if (stripePaymentInstrument && order.custom.stripeIsPaymentIntentInReview) {
            return {
                Order: order,
                order_created: true
            };
        } else {
            var orderPlacementStatus = Order.submit(order);
            if (!orderPlacementStatus.error) {
                clearForms();
            } else {
                stripeCheckoutHelper.refundCharge(order);
            }
            return orderPlacementStatus;
        }
    } else {
        const Email = app.getModel('Email');
        const Resource = require('dw/web/Resource');
        Email.sendMail({
            template: 'stripe/mail/orderreceived',
            recipient: order.getCustomerEmail(),
            subject: Resource.msg('order.ordercreceived-email.001', 'stripe', null),
            context: {
                Order: order
            }
        });

        return {
            Order: order,
            order_created: true
        };
    }
    // Stripe changes END
}
```

Controller: COShipping.js
app_storefront_controllers/cartridge/controllers/COShipping.js

In the function start() after if (cart.getProductLineItems().size() === 0) {

Add this code:

```
require('./COBilling').Start();
```

```
    session.forms.singleshipping.shippingAddress.shippingMethodID.value = cart.getDefaultShipment().getShippingMethodID();

    // Prepares shipments.
    homeDeliveries = prepareShipments();

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

    // Go to billing step, if we have no product line items, but only gift certificates in the basket, shipping is not required.
    if (cart.getProductLineItems().size() === 0) {
        //Stripe changes BEGIN
        require('./COBilling').Start();
        //Stripe changes END
    } else {
        pageMeta = require('*/cartridge/scripts/meta');
        pageMeta.update({
            pageTitle: Resource.msg('singleshipping.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')
        });
        app.getView({
            ContinueURL: URLUtils.https('COShipping-SingleShipping'),
            Basket: cart.object,
            HomeDeliveries: homeDeliveries
        }).render('checkout/shipping/singleshipping');
    }
}
```

In the function singleShipping() after the line session.forms.singleshipping.fulfilled.value = true;

Add this code:

```
require('./COBilling').Start();
```

```
function singleShipping() {
    var singleShippingForm = app.getForm('singleshipping');
    singleShippingForm.handleAction({
        save: function () {
            var cart = app.getModel('Cart').get();
            if (!cart) {
                // @FIXME redirect
                app.getController('Cart').Show();
                return;
            }

            handleShippingSettings(cart);

            // Attempts to save the used shipping address in the customer address book.
            if (customer.authenticated && session.forms.singleshipping.shippingAddress.addToAddressBook.value) {
                app.getModel('Profile').get(customer.profile).addAddressToAddressBook(cart.getDefaultShipment().getShippingAddress());
            }
            // Binds the store message from the user to the shipment.
            if (Site.getCurrent().getCustomPreferenceValue('enableStorePickUp')) {
                if (!app.getForm(session.forms.singleshipping.inStoreShipments.shipments).copyTo(cart.getShipments())) {
                    require('./Cart').Show();
                    return;
                }
            }

            // Mark step as fulfilled.
            session.forms.singleshipping.fulfilled.value = true;
            // @FIXME redirect
            //Stripe changes BEGIN
            require('./COBilling').Start();
            //Stripe changes END
        },
```

Controller: PaymentInstruments.js
app_storefront_controllers/cartridge/controllers/PaymentInstruments.js

Update the list() function as below:

```
var stripeHelper = require('int_stripe_core').getStripeHelper();
    var paymentInstruments;

    if (stripeHelper.isStripeEnabled()) {
        var wallet = stripeHelper.getStripeWallet(customer);
        var paymentInstruments = wallet.getPaymentInstruments();
    } else {
        var wallet = customer.getProfile().getWallet();
        var paymentInstruments =
wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD_CREDIT_CARD);
    }
```

Which should look like this:

```
23  /**
24   * Displays a list of customer payment instruments.
25   *
26   * Gets customer payment instrument information. Clears the paymentinstruments form and adds the customer
27   * payment information to it. Updates the page metadata.
28   * Renders a list of the saved credit card payment instruments of the current
29   * customer (account/payment/paymentinstrumentlist template).
30   */
31  function list() {
32      //Stripe changes BEGIN
33      var stripeHelper = require('int_stripe_core').getStripeHelper();
34      var paymentInstruments;
35
36      if (stripeHelper.isStripeEnabled()) {
37          var wallet = stripeHelper.getStripeWallet(customer);
38          var paymentInstruments = wallet.getPaymentInstruments();
39      } else {
40          var wallet = customer.getProfile().getWallet();
41          var paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD_CREDIT_CARD);
42      }
43      //Stripe changes END
44      var pageMeta = require('*/cartridge/scripts/meta');
45      var paymentForm = app.getForm('paymentinstruments');
46
47      paymentForm.clear();
48      paymentForm.get('creditcards.storedcards').copyFrom(paymentInstruments);
49
50      pageMeta.update(dw.content.ContentMgr.getContent('myaccount-paymentsettings'));
51
52      app.getView({
53          PaymentInstruments: paymentInstruments
54      }).render('account/payment/paymentinstrumentlist');
55  }
```

Controller: RedirectURL.js
app_storefront_controllers/cartridge/controllers/RedirectURL.js

In the function start add the following code:

```
    if (URLRedirectMgr.getRedirectOrigin() === '/.well-known/apple-developer-
merchantid-domain-association') { // Intercept the incoming path request
        app.getView().render('stripe/util/apple');
        return;
    }
```

```
/**
 * Gets the redirect. Renders the template for a redirect (util/redirectpermanent template). If no redirect can be found,
 * renders an error page (util/redirecterrorutil/redirecterror template).
 */
function start() {

    //Stripe change BEGIN
    if (URLRedirectMgr.getRedirectOrigin() === '/.well-known/apple-developer-merchantid-domain-association') { // Intercept the incoming path request
        app.getView().render('stripe/util/apple');
        return;
    }
    //Stripe change END

    var redirect = URLRedirectMgr.getRedirect(),
        location = redirect ? redirect.getLocation() : null;

    if (!location) {
        response.setStatus(410);
        app.getView().render('util/redirecterror');
    } else {
        app.getView({
            Location: location
        }).render('util/redirectpermanent');
    }
}
```

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.

The main configuration for integration of the Stripe services can be found under **Administration > Operations > Services**

There is a different service for each external call:

- stripe.http.addCard
- stripe.http.authorizePayment
- stripe.http.createCharge
- stripe.http.createCustomer
- stripe.http.deleteCard
- stripe.http.fetchCustomerCards
- stripe.http.fetchCustomerSources
- stripe.http.refundCharge
- stripe.http.retrieveCustomer
- stripe.http.service
- stripe.http.updateCard

All of these services use the same profile and the same credentials. The only thing that may be different is whether or not the communication log is enabled and the log name prefix. Here is the configuration of some of the services:

stripe.http.addCard

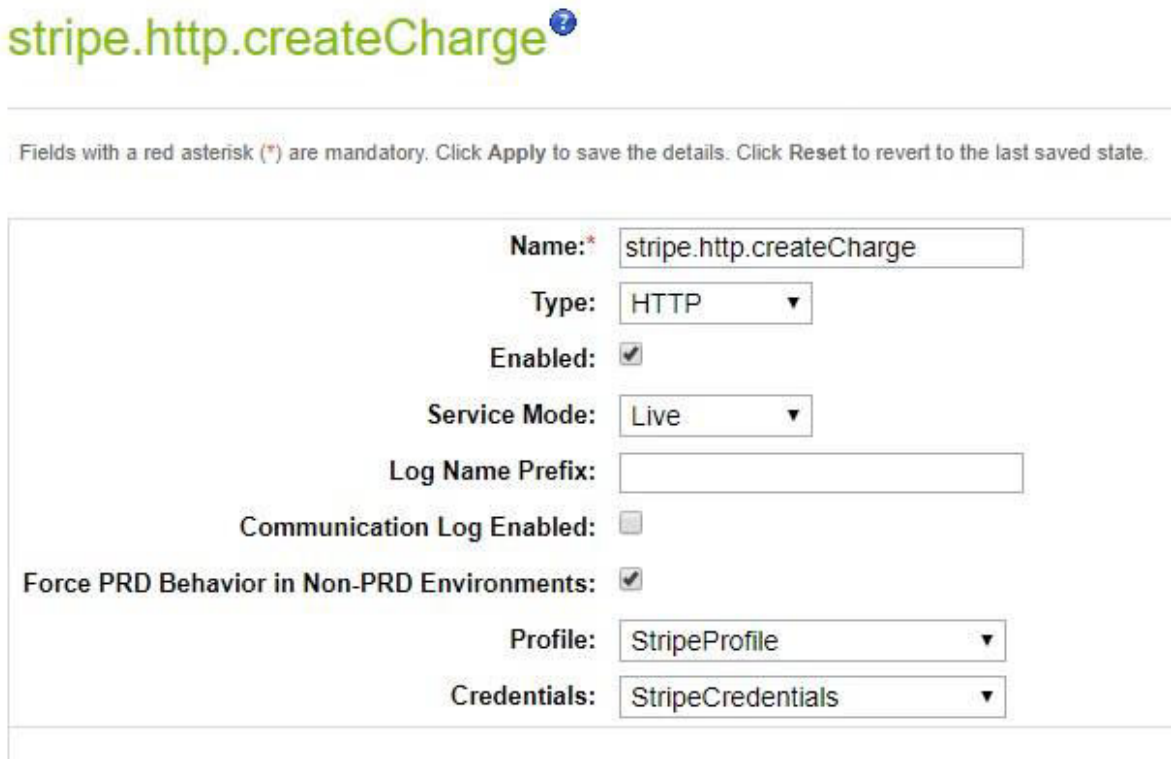
Fields with a red asterisk (*) are mandatory. Click **Apply** to save the details. Click **Reset** to revert to the last saved state.

Name:*	stripe.http.addCard
Type:	HTTP
Enabled:	☑
Service Mode:	Live
Log Name Prefix:	Stripe
Communication Log Enabled:	☑
Force PRD Behavior in Non-PRD Environments:	☐
Profile:	StripeProfile
Credentials:	StripeCredentials

stripe.http.authorizePayment

Fields with a red asterisk (*) are mandatory. Click **Apply** to save the details. Click **Reset** to revert to the last saved state.

Name:*	stripe.http.authorizePayment
Type:	HTTP
Enabled:	☑
Service Mode:	Live
Log Name Prefix:	Stripe
Communication Log Enabled:	☑
Force PRD Behavior in Non-PRD Environments:	☐
Profile:	StripeProfile
Credentials:	StripeCredentials



Firewall Requirements

No requirements

Alternate Payment Methods

Stripe cartridge supports alternate payment methods. Here is a guide how to use some of them:

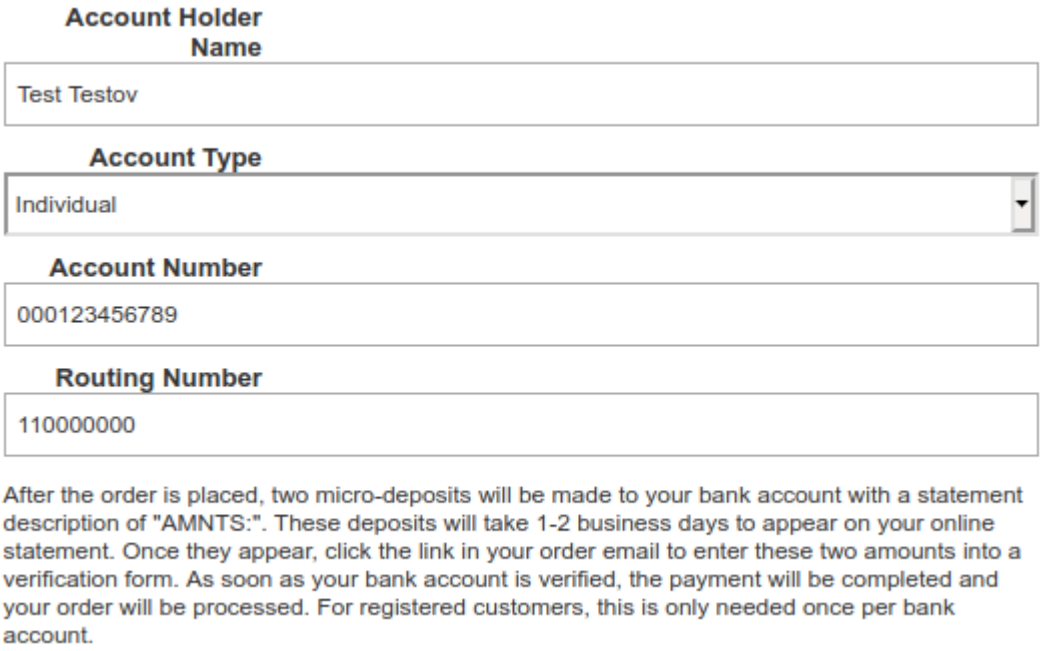
ACH Debit

Stripe supports accepting ACH payments—direct from bank accounts—alongside credit cards. ACH is currently supported only for Stripe businesses based in the U.S.

So, in Business Manager -> Merchant Tools -> Custom Preferences -> Stripe Configs -> Country Code (Stripe Payment Request Button) should be 'US' in order to be able to use ACH Debit.

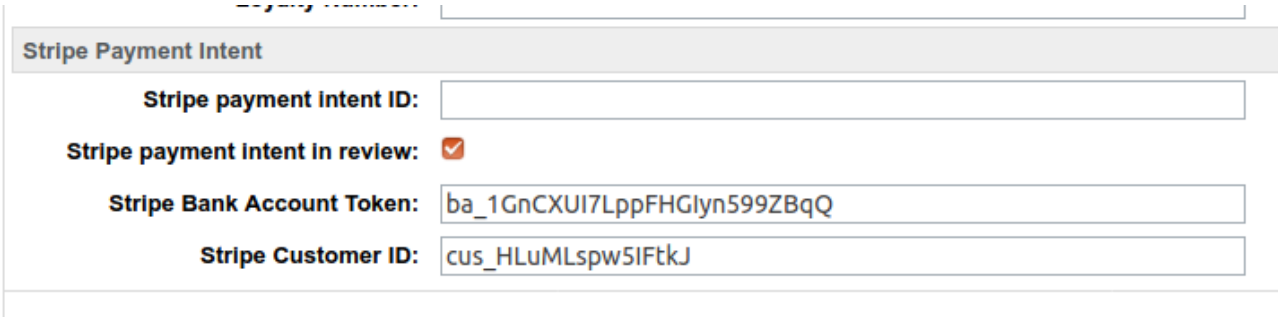
In order to Enable the ACH Debit as payment method, login to your Business Manager, select a site and navigate to Merchant Tools > Ordering > Payment Methods, then Enable payment method with ID = 'STRIPE_ACH_DEBIT'.

Once the ACH Debit payment method is enabled, it will enable the ACH Debit form as part of the checkout:



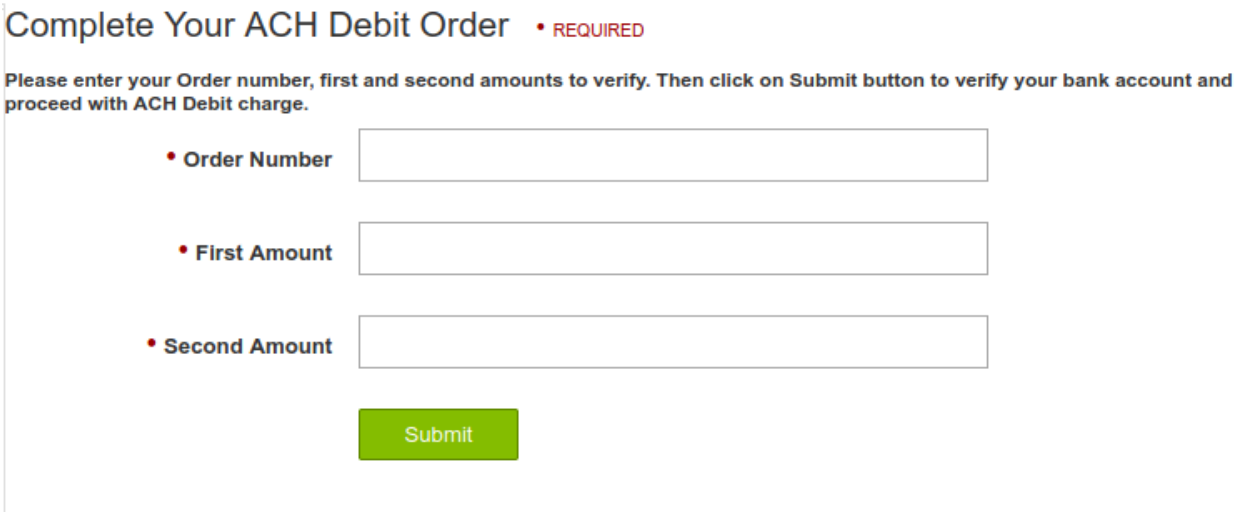
When a customer places an order with ACH Debit, it will initialize the following Order custom attributes:

- 'Stripe payment intent in review' is Checked (until payment is confirmed by the client)
- *'Stripe Bank Account Token'*
- *'Stripe Customer ID'*



Please Note: Before you can create an ACH charge, you must first collect and verify your customer's bank account and routing number.

The cartridge comes with such form where customer can enter ach debit confirmation information that can be modified additionally during Stripe ACH Debit integration:



app_stripe_core/templates/default/ach-debit-complete-form.isml

app_stripe_controllers/cartridge/controllers/AchDebit.js

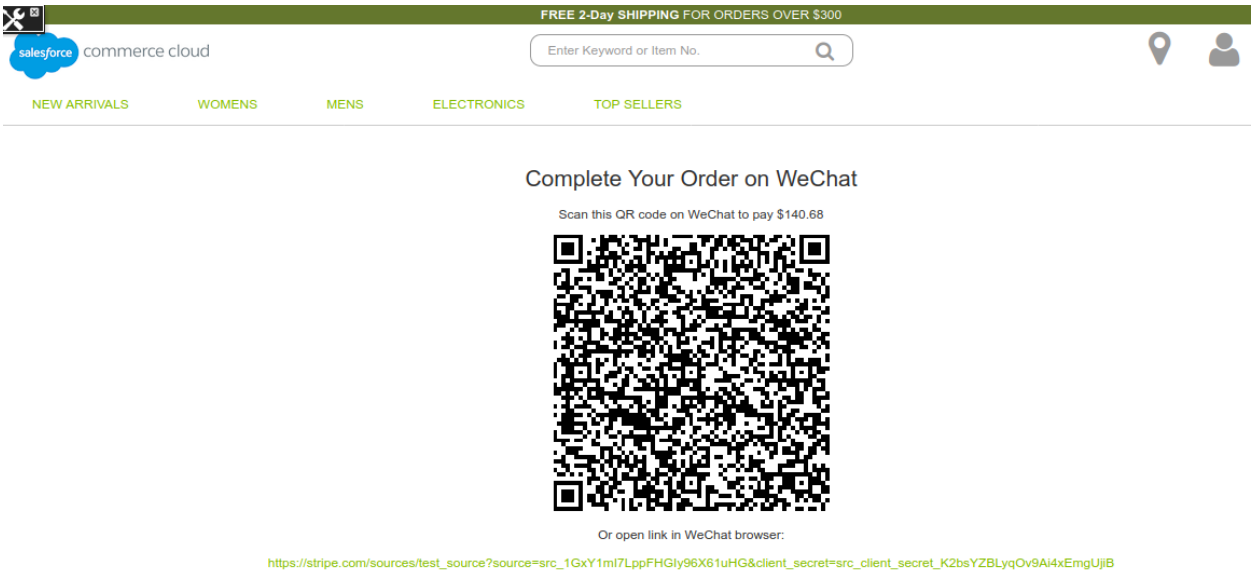
Please Note: The form is public available by default. So, you may need to restrict that access.

WeChat Pay

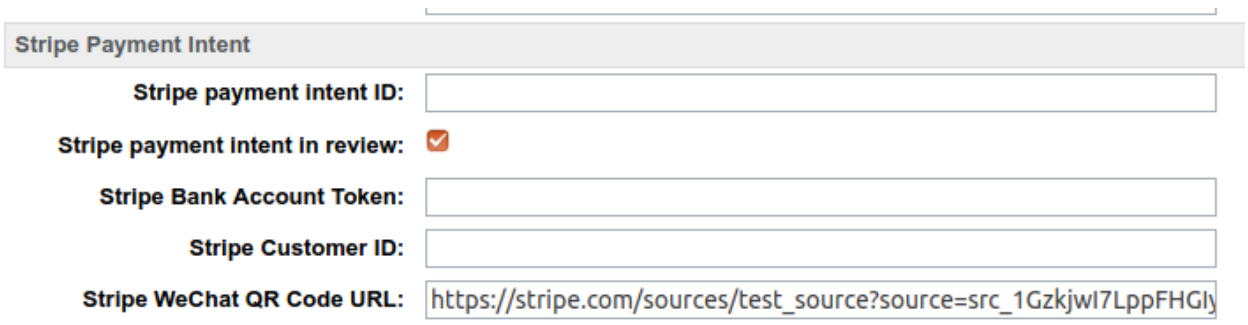
Stripe supports accepting WeChat pay payments, a popular payment method in China.

In order to Enable the WeChat as payment method, login to your Business Manager, select a site and navigate to Merchant Tools > Ordering > Payment Methods, then Enable payment method with ID = 'STRIPE_WECHATPAY'.

Once the WeChat Pay payment method is enabled, it will display the WeChat as a payment method in the checkout and an WeChat QR code at the last step of Order confirmation page:



Please Note: the QR Code URL will be stored as an Order custom attribute (stripeWeChatQRCodeURL):



The WeChat order will be marked as under review ('Stripe payment intent in review' will be checked) until payment is completed on WeChat side and a webhook notification is sent to SFCC which will update the payment.

4. Testing

Please, find more details on the test case document in the same folder.

5. Operations, Maintenance

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.

1. *stripeCustomerID(string) - Store Stripe customer ID*

Order/Basket Custom attributes

1. *stripePaymentIntentID(String) – Store payment intent ID.*
2. *stripeIsPaymentIntentInReview(Boolean) - Store payment intent in review*

Payment Transaction custom attributes

1. *stripeChargeId(string) - Store charge id*
2. *stripeChargeOutcomeData(text) - Store charge outcome data*
3. *stripeClientSecret(string) - Store client secret*
4. *stripeJsonData(text) - Store webhook JSON data*
5. *stripeOrderNumber(number) - Store order number*
6. *stripeSourceCanCharge(boolean) - Store if Stripe source can be charged*
7. *stripeSourceId(string) - Store Stripe source ID*

Payment Transaction custom attributes

1. *stripeChargeId(string) - Store charge ID*
2. *stripeCardID(string) - Store card ID*
3. *stripeCustomerID(string) - Store customer ID*
4. *stripeDefaultCard(boolean) - Store Stripe default card*
5. *stripeClientSecret(string) - Store client secret*
6. *stripePRUsed(boolean) - Store payment request button used*
7. *stripeSavePaymentInstrument(boolean) - Store save payment instrument*
8. *stripeSourceID(string) - Store Stripe source ID*

Custom Objects: In Business Manager, navigate to the Merchant Tools > Custom Objects > Custom Objects. Below custom object is there.

1. *StripeWebhookNotifications*

Custom Site Preferences: noted in detail above (section Configuration).

Availability

Please refer to the Stripe Service Level Agreement https://stripe.com/legal 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.

Failover/Recovery Process

If the Stripe service is unavailable the user will not be able to checkout.

The service availability can be tracked in SFCC using the Service Status.

Support

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

6. 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.

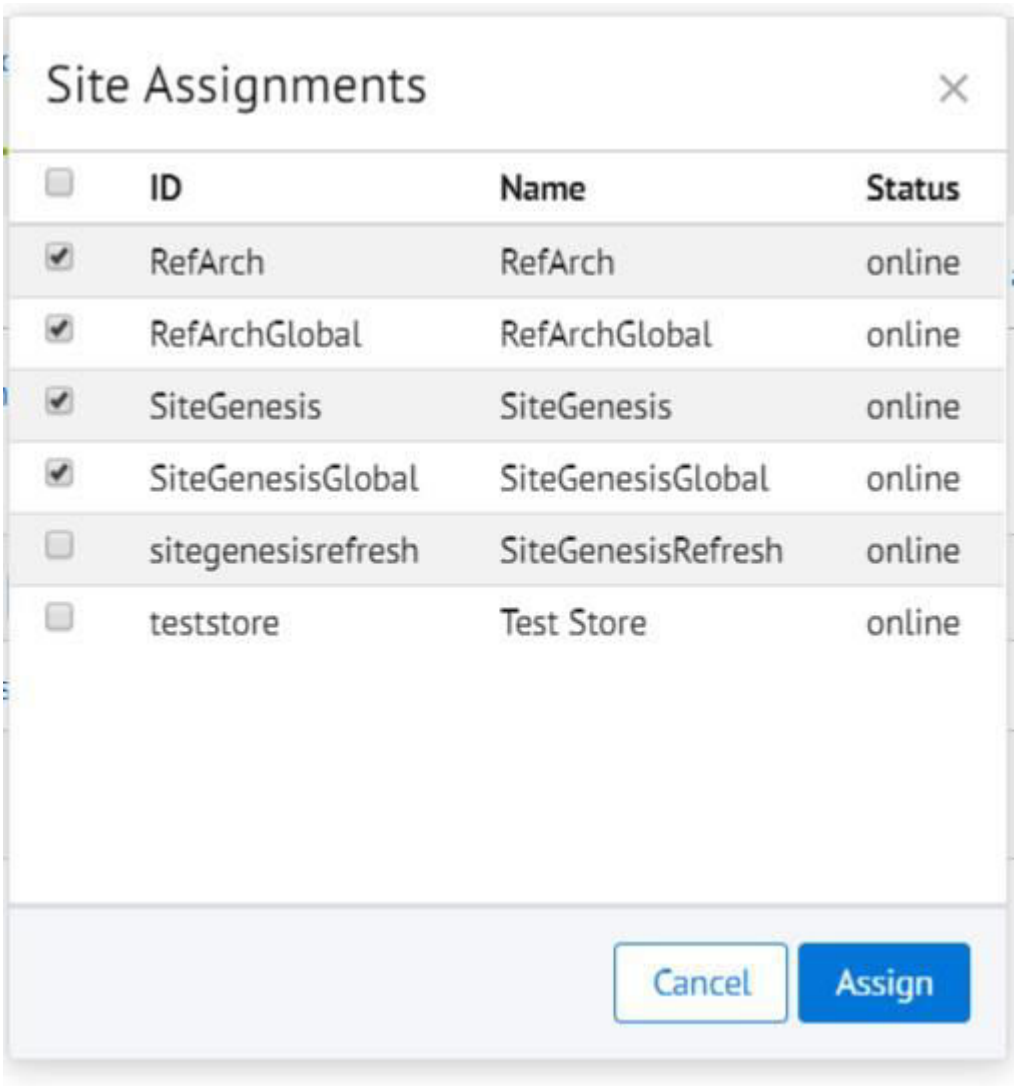
Business Manager

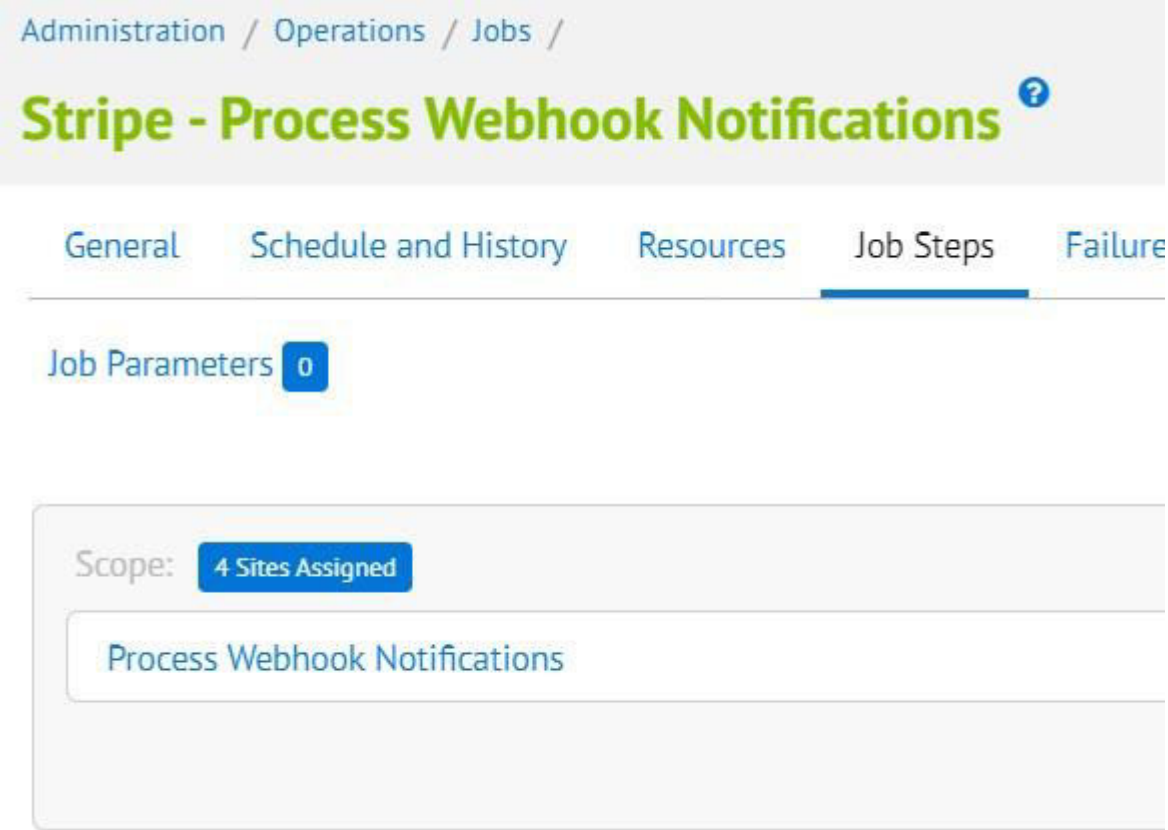
Business Manager settings and configuration notes are described in detail in the Configurations section.

There are 2 jobs coming with the cartridge:

- Stripe - Delete Custom Objects
- Stripe - Process Webhook Notifications

Enable the job “Stripe - Process Webhook Notifications” for the desired site:





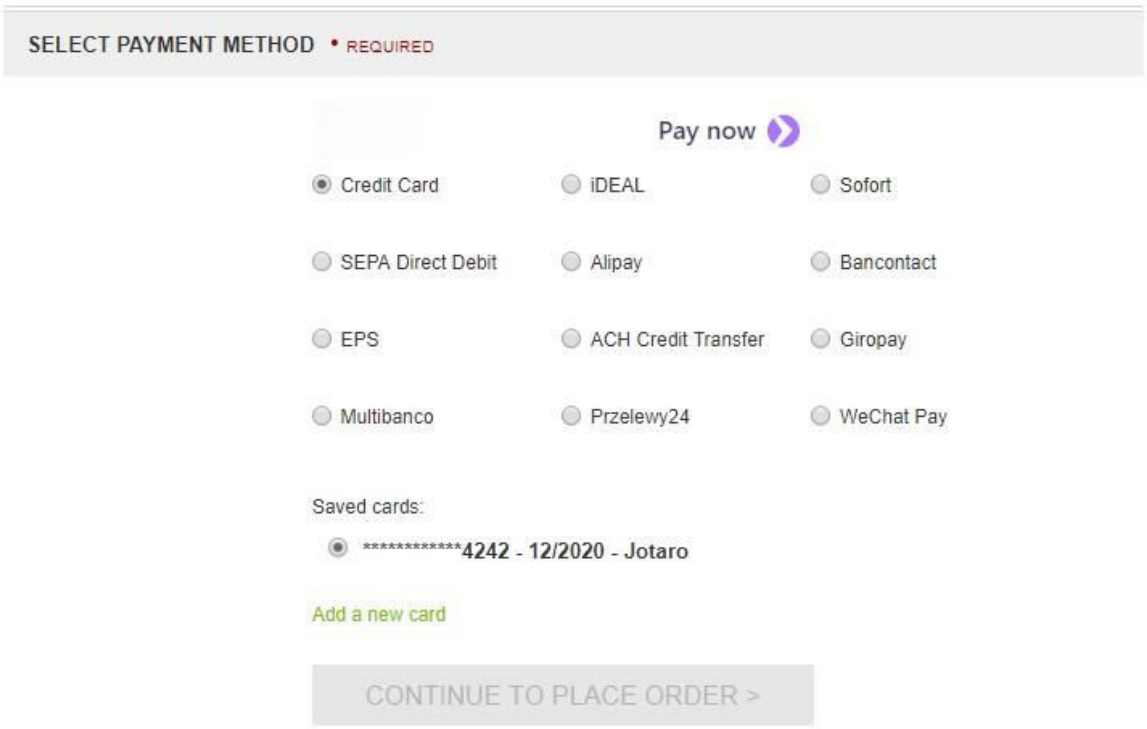
Storefront Functionality

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.

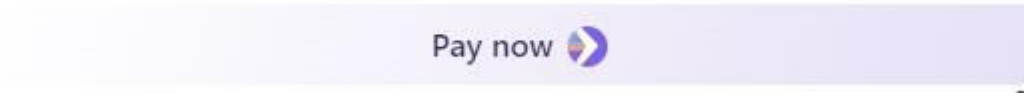
Saved Credit Cards

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.



Payment request button

When a customer has a saved address and credit card information in their browser they see the payment request button (Pay Now). The Payment Request Button Element gives you a single integration for Apple Pay, Google Pay, Microsoft Pay, and the browser standard Payment Request API.



Customers see the button above or an Apple Pay button, depending on what their device and browser combination supports. If neither option is available, they don't see the button. Supporting Apple Pay requires additional steps, but compatible devices automatically support browser-saved cards, Google Pay, and Microsoft Pay.

7. Known Issues

The LINK Cartridge has no known issues.

8. Release History

Version	Date	Changes
20.1.0	2020-02-01	Update documentation to match the new Salesforce template
18.1.0	2019-04-15	Update to use Stripe elements, sources, payment request button, webhooks and asynchronous payments
16.1.0	2019-07-30	Initial release