

Stripe Integration

Version 20.1.0



Stripe Integration

1. Summary

2. Component Overview

Functional Overview

Use Cases

Limitations, Constraints

Compatibility

Privacy, Payment

3. Implementation Guide

Setup of Business Manager

Configuration

Stripe Dashboard

Custom Code

Controllers

External Interfaces

Firewall Requirements

4. Operations, Maintenance

Data Storage

Availability

Failover/Recovery Process

Support

5. User Guide

Roles, Responsibilities

Business Manager

Storefront Functionality

6. Known Issues

7. Release History

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.

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.

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 implementations.

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" 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.

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>

Configuration

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

Stripe Allowed WebHook Statuses

Add

source.canceled

review.opened

charge.succeeded

charge.failed

source.failed

source.chargeable

review.closed

If webhook status match status that in this configuration, webhook data will be stored in Custom Obje...

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"] }

Stripe Dashboard

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

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

TEST DATA

Edit webhook endpoint

Endpoint URL

`https://stripe01-tech-prtnr-na05-dw.demandware.net/on/demandware.store/Sites-SiteG`

Events to send

Select events... Clear

- charge.succeeded
- Charge 13 events
- charge.failed
- source.canceled

7 events

Cancel Edit endpoint

TEST DATA

[https://stripe01-tech-prtnr-na05-dw.demandware.net/on/demandware.stor...](#)

Send test webhook...
Disable...
Delete...

Status

Mode

Enabled

Test

Webhook details

Update details...

URL

https://stripe01-tech-prtnr-na05-dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/Stripe-WebHook

Event types

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

Signing secret

Learn more about webhook signing

Click to reveal

Roll secret...

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:

Country Code (Stripe Payment Request Button)

GB

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;
}

```

```

26 /**
27  * Gets the redirect. Renders the template for a redirect (util/redirectpermanent template). If no redirect can be found,
28  * renders an error page (util/redirecterrorutil/redirecterror template).
29  */
30 function start() {
31
32     //Stripe change BEGIN
33     if (URLRedirectMgr.getRedirectOrigin() === '/.well-known/apple-developer-merchantid-domain-association') { // Intercept the incoming path request
34         app.getView().render('stripe/util/apple');
35         return;
36     }
37     //Stripe change END
38
39     var redirect = URLRedirectMgr.getRedirect(),
40         location = redirect ? redirect.getLocation() : null;
41
42     if (!location) {
43         response.setStatus(410);
44         app.getView().render('util/redirecterror');
45     } else {
46         app.getView({
47             location: location
48         }).render('util/redirectpermanent');
49     }
50 }

```

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:

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.

2

↓ Download verification file...

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

ApplePay Verification String

```

7B227073704964223A2239373943394538343346343131343044463144
3138343432323932323137343130343530443143394644463944373843
37313531303944334643463542433731222C2276657273696F6E223A31
2C22637265617465644F6E223A313437313435343137313137362C2273
69676E6174757265223A22333038303036303932613836343838366637
3064303130373032613038303330383030323031303133313066333030
6430363039363038363438303136353033303430323031303530303330
3830303630393261383634383836663730643031303730313030303061
3038303330383230336536333038323033386261303033303230313032
3032303836383630663639396439636361373066333030613036303832

```

The Stripe console will proxy the Apple Pay for Web verification String upon set...

Then click on the 'Add' button:

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
```

CancelAdd

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.

Controllers

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
```

```

133 function initCreditCardList(cart) {
134     var paymentAmount = cart.getNonGiftCertificateAmount();
135     var countryCode;
136     var applicablePaymentMethods;
137     var applicablePaymentCards;
138     var applicableCreditCards;
139
140     countryCode = Countries.getCurrent({
141         currentRequest: {
142             locale: request.locale
143         }
144     }).countryCode;
145
146     applicablePaymentMethods = PaymentMgr.getApplicablePaymentMethods(customer, countryCode, paymentAmount.value);
147     applicablePaymentCards = PaymentMgr.getPaymentMethod(PaymentInstrument.METHOD_CREDIT_CARD).getApplicablePaymentCards(customer, countryCode, paymentAmount.value);
148
149     //Stripe change BEGIN
150     var stripeHelper = require('int_stripe_core').getStripeHelper();
151     if (stripeHelper.isStripeEnabled()) {
152         applicablePaymentMethods = stripeHelper.getStripePaymentMethods(applicablePaymentMethods, request.locale);
153     }
154     //Stripe change END
155
156     app.getForm('billing').object.paymentMethods.creditCard.type.setOptions(applicablePaymentCards.iterator());
157
158     applicableCreditCards = null;
159
160     if (customer.authenticated) {
161         var profile = app.getModel('Profile').get();
162         if (profile) {
163             applicableCreditCards = profile.validateWalletPaymentInstruments(countryCode, paymentAmount.getValue()).ValidPaymentInstruments;
164         }
165     }
166
167     return {
168         ApplicablePaymentMethods: applicablePaymentMethods,
169         ApplicableCreditCards: applicableCreditCards
170     };
171 }

```

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;
    }
});

```

```

542     save: function () {
543         Transaction.wrap(function () {
544             var cart = app.getModel('Cart').get();
545
546             if (!resetPaymentForms() || !validateBilling() || !handleBillingAddress(cart) || // Performs validation steps, based upon the entered billing address
547                 // and address options.
548             handlePaymentSelection(cart).error) { // Performs payment method specific checks, such as credit card verification.
549                 returnToForm(cart);
550             } else {
551                 if (customer.authenticated && app.getForm('billing').object.billingAddress.addToAddressBook.value) {
552                     app.getModel('Profile').get(customer.profile).addAddressToAddressBook(cart.getBillingAddress());
553                 }
554
555                 // Mark step as fulfilled
556                 app.getForm('billing').object.fulfilled.value = true;
557
558                 // A successful billing page will jump to the next checkout step.
559                 // Stripe changes BEGIN
560                 if (request.httpParameterMap.isParameterSubmitted('stripe_redirect_url') && !empty(request.httpParameterMap.stripe_redirect_url.stringValue)) {
561                     response.redirect(request.httpParameterMap.stripe_redirect_url.stringValue);
562                 } else {
563                     require('./COSummary').Start();
564                 }
565                 // Stripe changes END
566                 return;
567             }
568         });
569     },
570 }

```

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.

```

125     // Recalculate the payments. If there is only gift certificates, make sure it covers the order total, if not
126     // back to billing page.
127     Transaction.wrap(function () {
128         if (!cart.calculatePaymentTransactionTotal()) {
129             COBilling.Start();
130             return {};
131         }
132     });
133
134     // Handle used addresses and credit cards.
135     var saveCCResult = COBilling.SaveCreditCard();
136
137     if (!saveCCResult) {
138         return {
139             error: true,
140             PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
141         };
142     }
143
144     // Creates a new order. This will internally ReserveInventoryForOrder and will create a new Order with status
145     // 'Created'.
146     // Stripe changes BEGIN
147     const stripeCheckoutHelper = require('int_stripe_core').getCheckoutHelper();
148     var order = stripeCheckoutHelper.createOrder(cart.object);
149     // Stripe changes END

```

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
    };
}
```

```

171         return {
172             error: true,
173             PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
174         };
175     });
176 }
177
178 // Stripe changes BEGIN
179 var isAPMOrder = stripeCheckoutHelper.isAPMOrder(order);
180 if (!isAPMOrder) {
181     var stripePaymentInstrument = stripeCheckoutHelper.getStripePaymentInstrument(order);
182
183     if (stripePaymentInstrument && order.custom.stripeIsPaymentIntentInReview) {
184         return {
185             Order: order,
186             order_created: true
187         };
188     } else {
189         var orderPlacementStatus = Order.submit(order);
190         if (!orderPlacementStatus.error) {
191             clearForms();
192         } else {
193             stripeCheckoutHelper.refundCharge(order);
194         }
195         return orderPlacementStatus;
196     }
197 } else {
198     const Email = app.getModel('Email');
199     const Resource = require('dw/web/Resource');
200     Email.sendMail({
201         template: 'stripe/mail/orderreceived',
202         recipient: order.getCustomerEmail(),
203         subject: Resource.msg('order.ordercreceived-email.001', 'stripe', null),
204         context: {
205             Order: order
206         }
207     });
208 }
209
210 return {
211     Order: order,
212     order_created: true
213 };
214 }
215 // Stripe changes END
216 }

```

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();
```

```

92 session.forms.singleshipping.shippingAddress.shippingMethodID.value = cart.getDefaultShipment().getShippingMethodID();
93
94 // Prepares shipments.
95 homeDeliveries = prepareShipments();
96
97 Transaction.wrap(function () {
98     cart.calculate();
99 });
100
101 // Go to billing step, if we have no product line items, but only gift certificates in the basket, shipping is not required.
102 if (cart.getProductLineItems().size() === 0) {
103     //Stripe changes BEGIN
104     require('./COBilling').Start();
105     //Stripe changes END
106 } else {
107     pageMeta = require('*/cartridge/scripts/meta');
108     pageMeta.update({
109         pageTitle: Resource.msg('singleshipping.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')
110     });
111     app.getView({
112         ContinueURL: URLUtils.https('COShipping-SingleShipping'),
113         Basket: cart.object,
114         HomeDeliveries: homeDeliveries
115     }).render('checkout/shipping/singleshipping');
116 }
117 }

```

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;
    }

```



```

26  /**
27   * Gets the redirect. Renders the template for a redirect (util/redirectpermanent template). If no redirect can be found,
28   * renders an error page (util/redirecterrorutil/redirecterror template).
29   */
30  function start() {
31
32      //Stripe change BEGIN
33      if (URLRedirectMgr.getRedirectOrigin() === '/.well-known/apple-developer-merchantid-domain-association') { // Intercept the incoming path request
34          app.getView().render('stripe/util/apple');
35          return;
36      }
37      //Stripe change END
38
39      var redirect = URLRedirectMgr.getRedirect(),
40          location = redirect ? redirect.getLocation() : null;
41
42      if (!location) {
43          response.setStatus(410);
44          app.getView().render('util/redirecterror');
45      } else {
46          app.getView({
47              Location: location
48          }).render('util/redirectpermanent');
49      }
50  }

```

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:*	<input type="text" value="stripe.http.addCard"/>
Type:	<input type="text" value="HTTP"/>
Enabled:	<input checked="" type="checkbox"/>
Service Mode:	<input type="text" value="Live"/>
Log Name Prefix:	<input type="text" value="Stripe"/>
Communication Log Enabled:	<input checked="" type="checkbox"/>
Force PRD Behavior in Non-PRD Environments:	<input type="checkbox"/>
Profile:	<input type="text" value="StripeProfile"/>
Credentials:	<input type="text" value="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:*	<input type="text" value="stripe.http.authorizePayment"/>
Type:	<input type="text" value="HTTP"/>
Enabled:	<input checked="" type="checkbox"/>
Service Mode:	<input type="text" value="Live"/>
Log Name Prefix:	<input type="text" value="Stripe"/>
Communication Log Enabled:	<input checked="" type="checkbox"/>
Force PRD Behavior in Non-PRD Environments:	<input type="checkbox"/>
Profile:	<input type="text" value="StripeProfile"/>
Credentials:	<input type="text" value="StripeCredentials"/>

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

Name:*	<input type="text" value="stripe.http.createCharge"/>
Type:	<input type="text" value="HTTP"/>
Enabled:	<input checked="" type="checkbox"/>
Service Mode:	<input type="text" value="Live"/>
Log Name Prefix:	<input type="text"/>
Communication Log Enabled:	<input type="checkbox"/>
Force PRD Behavior in Non-PRD Environments:	<input checked="" type="checkbox"/>
Profile:	<input type="text" value="StripeProfile"/>
Credentials:	<input type="text" value="StripeCredentials"/>

Firewall Requirements

No requirements

4. 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://support.stripe.com>).

5. 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:

Site Assignments

<input type="checkbox"/>	ID	Name	Status
<input checked="" type="checkbox"/>	RefArch	RefArch	online
<input checked="" type="checkbox"/>	RefArchGlobal	RefArchGlobal	online
<input checked="" type="checkbox"/>	SiteGenesis	SiteGenesis	online
<input checked="" type="checkbox"/>	SiteGenesisGlobal	SiteGenesisGlobal	online
<input type="checkbox"/>	sitegenesisrefresh	SiteGenesisRefresh	online
<input type="checkbox"/>	teststore	Test Store	online

CancelAssign

Stripe - Process Webhook Notifications ?

[General](#)

[Schedule and History](#)

[Resources](#)

[Job Steps](#)

[Failure](#)

[Job Parameters](#) 0

Scope: 4 Sites Assigned

[Process Webhook Notifications](#)

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.

SELECT PAYMENT METHOD • REQUIRED

Pay now 

☒ Credit Card

☐ iDEAL

☐ Sofort

☐ SEPA Direct Debit

☐ Alipay

☐ Bancontact

☐ EPS

☐ ACH Credit Transfer

☐ Giropay

☐ Multibanco

☐ Przelewy24

☐ WeChat Pay

Saved cards:

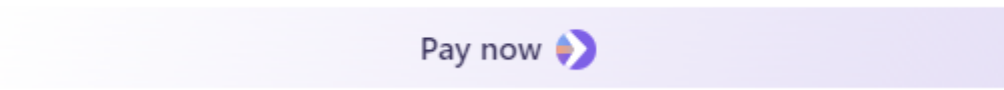
☒ *****4242 - 12/2020 - Jotaro

Add a new card

CONTINUE TO PLACE ORDER >

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.

6. Known Issues

The LINK Cartridge has no known issues.

7. 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