

# STRIPE INTEGRATION

20.1.0



## [Table of Contents](#)

<b>1. Summary</b>	<b>3</b>
<b>2. Component Overview</b>	<b>4</b>
<i>Functional Overview</i>	4
<i>Use Cases</i>	4
<i>Limitations, Constraints</i>	4
<i>Compatibility</i>	4
<i>Privacy, Payment</i>	4
<b>3. Implementation Guide</b>	<b>6</b>
<i>Setup of Business Manager</i>	6
<i>Configuration</i>	7
<i>Stripe Dashboard</i>	8
<i>Custom Code</i>	11
<i>External Interfaces</i>	16
<i>Firewall Requirements</i>	19
<b>4. Testing</b>	<b>20</b>
<b>5. Operations, Maintenance</b>	<b>21</b>
<i>Data Storage</i>	21
<i>Availability</i>	21
<i>Failover/Recovery Process</i>	21
<i>Support</i>	22
<b>6. User Guide</b>	<b>23</b>
<i>Roles, Responsibilities</i>	23
<i>Business Manager</i>	23
<i>Storefront Functionality</i>	24
Payment request button	25
<b>7. Known Issues</b>	<b>26</b>
<b>8. Release History</b>	<b>27</b>

## 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, SFRA version 4.4*

*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 SFRA "app\_stripe\_sfra:int\_stripe\_sfra:int\_stripe\_core" needs 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

*If necessary, update the path to your base SFRA installation in package.json file from the same root folder.*

*Normally you would have a top-level project folder, into which the repositories of SFRA base cartridge and all required plugins, libraries and any other LINK cartridges will be cloned. In case you have cloned the Stripe cartridge into that folder as well, the below change will not be required. Otherwise, update paths.base property in the package.json to contain a relative path to the local directory containing the Storefront Reference Architecture repository. Its default value will be as follows:*

```
"paths": {  
  "base": "../storefront-reference-architecture/cartridges/app_storefront_base/"  
}
```

*Once you are certain the correct path to SFRA cartridges is configured, run npm run compile:scss command from the root folder of Stripe repository.*

##### 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

{LINK Integration Documentation}

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

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" ] }
13. Stripe Enabled – Enables or disables the cartridge

## Stripe Dashboard

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

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...

charge.su

Charge 13 events

charge.succeeded

charge.failed

source.canceled

7 events

Clear

×

×

×

×

×

Cancel

Edit endpoint



TEST DATA

https://stripe01-tech-prtnr-na05-dw.demandware.net/on/demandware.store/... [Send test webhook...](#) [Disable...](#) [Delete...](#)

Status: **Enabled** Mode: 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)

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
  res.render('stripe/util/apple');
  return next();
}
```

```
link stripe > cartridges > app.stripe.sfra > cartridge > controllers > JS RedirectURL.js > ...
5 server.extend(page);
6
7 server.replace('Start', function (req, res, next) {
8   var URLRedirectMgr = require('dw/web/URLRedirectMgr');
9
10  // Stripe changes BEGIN
11  if (URLRedirectMgr.getRedirectOrigin() === '/.well-known/apple-developer-merchantid-domain-association') { // Intercept the incoming path request
12    res.render('stripe/util/apple');
13    return next();
14  }
15  // Stripe changes END
16
17  var redirect = URLRedirectMgr.redirect;
18  var location = redirect ? redirect.location : null;
19  var redirectStatus = redirect ? redirect.getStatus() : null;
20
21  if (!location) {
22    res.statusCode(404);
23    res.render('error/notFound');
24  } else {
25    if (redirectStatus) {
26      res.setRedirectStatus(redirectStatus);
27    }
28    res.redirect(location);
29  }
30
31  return next();
32  });
```

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](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.

example.com

**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  
38303630393261383634383836663730643031303730313030303061  
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 SFRA version 4.4. Below are the customizations made to SFRA code.

There are a lot of controller endpoints that are appends instead of replaces. Those will not get covered as they should work without doing anything.

Controller updates are only required for replaced endpoints as you may have already replaced that endpoint in your integration. Simply use the changes that are made to the base cartridge and add them to your already replaced controller. If you haven't extended/replaced these endpoints you don't need to do anything.

#### **Controller: CheckoutServices.js**

app\_stripe\_sfra/cartridge/controllers/CheckoutServices.js

Remove the payment instrument validation in the 'SubmitPayment' endpoint

```

161 // if there is no selected payment option and balance is greater than zero
162 if (!paymentMethodID && currentBasket.totalGrossPrice.value > 0) {
163     var noPaymentMethod = {};
164
165     noPaymentMethod[billingData.paymentMethod.htmlName] =
166         Resource.msg('error.no.selected.payment.method', 'payment', null);
167
168     delete billingData.paymentInformation;
169
170     res.json({
171         form: billingForm,
172         fieldErrors: [noPaymentMethod],
173         serverErrors: [],
174         error: true
175     });
176     return;
177 }
178
179 // Stripe changes BEGIN
180 // // Validate payment instrument
181 // var creditCardPaymentMethod = PaymentMgr.getPaymentMethod(PaymentInstrument.METHOD_CREDIT_CARD);
182 // var paymentCard = PaymentMgr.getPaymentCard(billingData.paymentInformation.cardType.value);
183
184 // var applicablePaymentCards = creditCardPaymentMethod.getApplicablePaymentCards(
185 //     req.currentCustomer.raw,
186 //     req.geolocation.countryCode,
187 //     null
188 // );
189
190 // if (!applicablePaymentCards.contains(paymentCard)) {
191 //     // Invalid Payment Instrument
192 //     var invalidPaymentMethod = Resource.msg('error.payment.not.valid', 'checkout', null);
193 //     delete billingData.paymentInformation;
194 //     res.json({
195 //         form: billingForm,
196 //         fieldErrors: [],
197 //         serverErrors: [invalidPaymentMethod],
198 //         error: true
199 //     });
200 //     return;
201 // }
202 // Stripe changes END
203

```

Update 'PlaceOrder' as below:

Replace the order creation block:

```

// Re-calculate the payments.
var calculatedPaymentTransactionTotal=
COHelpers.calculatePaymentTransaction(currentBasket);
if (calculatedPaymentTransactionTotal.error) {
    res.json({
        error: true,
        errorMessage: Resource.msg('error.technical', 'checkout', null)
    });
    return next();
}

```

With this:

```

const stripeCheckoutHelper =
require('int_stripe_core').getCheckoutHelper();
var order = stripeCheckoutHelper.createOrder(currentBasket);

```

```

395 // Re-calculate the payments.
396 var calculatedPaymentTransactionTotal = COHelpers.calculatePaymentTransaction(currentBasket);
397 if (calculatedPaymentTransactionTotal.error) {
398     res.json({
399         error: true,
400         errorMessage: Resource.msg('error.technical', 'checkout', null)
401     });
402     return next();
403 }
404
405 // Stripe changes BEGIN
406 const stripeCheckoutHelper = require('int_stripe_core').getCheckoutHelper();
407 var order = stripeCheckoutHelper.createOrder(currentBasket);
408 // Stripe changes END
409
410 if (!order) {
411     res.json({
412         error: true,
413         errorMessage: Resource.msg('error.technical', 'checkout', null)
414     });
415     return next();
416 }

```

Replace the order placement down below too:

```
var placeOrderResult = COHelpers.placeOrder(order, raudDetectionStatus);
```

Replace everything after this line with:

```

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

    if (stripePaymentInstrument &&
order.custom.stripeIsPaymentIntentInReview) {
        res.json({
            error: false,
            orderID: order.orderNo,
            orderToken: order.orderToken,
            continueUrl: URLUtils.url('Order-Confirm').toString()
        });

        return next();
    }
    // Places the order
    var placeOrderResult = COHelpers.placeOrder(order,
fraudDetectionStatus);
    if (placeOrderResult.error) {
        stripeCheckoutHelper.refundCharge(order);
        res.json({
            error: true,
            errorMessage: Resource.msg('error.technical', 'checkout', null)
        });
        return next();
    }

    COHelpers.sendConfirmationEmail(order, req.locale.id);

    // Reset usingMultiShip after successful Order placement
    req.session.privacyCache.set('usingMultiShipping', false);

```

```

        res.json({
            error: false,
            orderID: order.orderNo,
            orderToken: order.orderToken,
            continueUrl: URLUtils.url('Order-Confirm').toString()
        });

        return next();
    }
    res.json({
        error: false,
        orderID: order.orderNo,
        orderToken: order.orderToken,
        continueUrl: URLUtils.url('Order-Confirm').toString()
    });

    return next();
}

```

```

428 var fraudDetectionStatus = hooksHelper('app.fraud.detection', 'fraudDetection', currentBasket, require('*/cartridge/scripts/hooks/fraudDetection').fraudDetection);
429 if (fraudDetectionStatus.status === 'fail') {
430     Transaction.wrap(function () { OrderMgr.failOrder(order); });
431
432     // fraud detection failed
433     req.session.privacyCache.set('fraudDetectionStatus', true);
434
435     res.json({
436         error: true,
437         cartError: true,
438         redirectUrl: URLUtils.url('Error-ErrorCode', 'err', fraudDetectionStatus.errorCode).toString(),
439         errorMessage: Resource.msg('error.technical', 'checkout', null)
440     });
441     return next();
442 }
443
444 // Stripe changes BEGIN
445 var isAPMOrder = stripeCheckoutHelper.isAPMOrder(order);
446 if (!isAPMOrder) {
447     var stripePaymentInstrument = stripeCheckoutHelper.getStripePaymentInstrument(order);
448
449     if (stripePaymentInstrument && order.custom.stripeIsPaymentIntentInReview) {
450         res.json({
451             error: false,
452             orderID: order.orderNo,
453             orderToken: order.orderToken,
454             continueUrl: URLUtils.url('Order-Confirm').toString()
455         });
456         return next();
457     }
458
459     // Places the order
460     var placeOrderResult = COHelpers.placeOrder(order, fraudDetectionStatus);
461     if (placeOrderResult.error) {
462         stripeCheckoutHelper.refundCharge(order);
463         res.json({
464             error: true,
465             errorMessage: Resource.msg('error.technical', 'checkout', null)
466         });
467         return next();
468     }
469
470     COHelpers.sendConfirmationEmail(order, req.locale.id);
471
472     // Reset usingMultiship after successful Order placement
473     req.session.privacyCache.set('usingMultishipping', false);
474
475     // TODO: Exposing a direct route to an Order, without at least encoding the orderID
476     // is a serious PII violation. It enables looking up every customers orders, one at a
477     // time.
478     res.json({
479         error: false,
480         orderID: order.orderNo,
481         orderToken: order.orderToken,
482         continueUrl: URLUtils.url('Order-Confirm').toString()
483     });
484     return next();
485 }
486 }
487

```

### Controller: PaymentInstruments.js

app\_stripe\_sfra/cartridge/controllers/PaymentInstruments.js

Replace the DeletePayment endpoint with this code



```

server.replace('DeletePayment', function (req, res, next) {
  var stripeHelper = require('int_stripe_core').getStripeHelper();
  var wallet = stripeHelper.getStripeWallet(customer);
  var UUID = req.querystring.UUID;
  wallet.removePaymentInstrument({ custom: { stripeId: UUID } });

  res.json({ UUID: UUID });
  next();
});

```

```

15  server.replace('DeletePayment', function (req, res, next) {
16      var stripeHelper = require('int_stripe_core').getStripeHelper();
17      var wallet = stripeHelper.getStripeWallet(customer);
18      var UUID = req.querystring.UUID;
19      wallet.removePaymentInstrument({ custom: { stripeId: UUID } });
20
21      res.json({ UUID: UUID });
22      next();
23  });

```

### Controller: RedirectURL.js

app\_stripe\_sfra/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
        res.render('stripe/util/apple');
        return next();
    }

```

```

7  server.replace('Start', function (req, res, next) {
8      var URLRedirectMgr = require('dw/web/URLRedirectMgr');
9
10     // Stripe changes BEGIN
11     if (URLRedirectMgr.getRedirectOrigin() === '/.well-known/apple-developer-merchantid-domain-association') { // Intercept the incoming path request
12         res.render('stripe/util/apple');
13         return next();
14     }
15     // Stripe changes END
16
17     var redirect = URLRedirectMgr.redirect;
18     var location = redirect ? redirect.location : null;
19     var redirectStatus = redirect ? redirect.getStatus() : null;
20
21     if (!location) {
22         res.statusCode(404);
23         res.render('error/notFound');
24     } else {
25         if (redirectStatus) {
26             res.setRedirectStatus(redirectStatus);
27         }
28         res.redirect(location);
29     }
30
31     return next();
32 });

```

## 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.

{LINK Integration Documentation}



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<sup>?</sup>

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:		<input checked="" type="checkbox"/>
Service Mode:		Live ▼
Log Name Prefix:		Stripe
Communication Log Enabled:		<input checked="" type="checkbox"/>
Force PRD Behavior in Non-PRD Environments:		<input type="checkbox"/>
Profile:		StripeProfile ▼
Credentials:		StripeCredentials ▼

## stripe.http.authorizePayment<sup>?</sup>

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:		<input checked="" type="checkbox"/>
Service Mode:		Live ▼
Log Name Prefix:		Stripe
Communication Log Enabled:		<input checked="" type="checkbox"/>
Force PRD Behavior in Non-PRD Environments:		<input type="checkbox"/>
Profile:		StripeProfile ▼
Credentials:		StripeCredentials ▼

## stripe.http.createCharge

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

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

### Firewall Requirements

*No requirements*

## 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://support.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:

### 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"/>	sitogenesisrefresh	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.



Payment

Billing Address

qweqwe qatgtwqatw 2 Wylcotts Worcester, AL 12355

Update Address
Add New

\*Email

\*Phone Number

3333333333

Pay now

Credit Card
Alipay
ACH Credit Transfer

Saved cards:

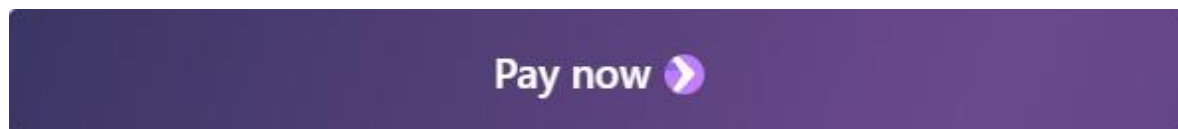
\*\*\*\*\*4242 - 10/2020 - share

\*\*\*\*\*3220 - 2/2029 - QWe qwe

Add a new card

#### 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