**Klarna Payments**

Version 18.1.0



Table of Contents

[1. Summary 1-3](#_Toc535944248)

[2. Component Overview 2-4](#_Toc535944249)

[2.1 Functional Overview 2-4](#_Toc535944250)

[2.2 Use Cases 2-5](#_Toc535944251)

[2.3 Limitations, Constraints 2-23](#_Toc535944252)

[2.4 Compatibility 2-24](#_Toc535944253)

[2.5 Privacy, Payment 2-24](#_Toc535944254)

[3. Implementation Guide 3-28](#_Toc535944255)

[3.1 Setup 3-28](#_Toc535944256)

[3.1.1 int\_klarna\_payments 3-28](#_Toc535944257)

[3.1.2 int\_klarna\_payments\_pipelines 3-29](#_Toc535944258)

[3.1.3 int\_klarna\_payments\_controllers 3-29](#_Toc535944259)

[3.1.4 References to Site Genesis Cartridges 3-29](#_Toc535944260)

[3.2 Configuration 3-30](#_Toc535944261)

[3.2.1 Metadata Import 3-30](#_Toc535944262)

[3.2.2 Cartridge Path 3-32](#_Toc535944263)

[3.2.3 Klarna Payments Configurations 3-33](#_Toc535944264)

[3.2.4 Klarna Payments Logo and Payment Option Name 3-34](#_Toc535944265)

[3.2.5 Services Configuration 3-35](#_Toc535944266)

[3.2.6 Custom attributes 3-37](#_Toc535944267)

[3.3 Custom Code 3-37](#_Toc535944268)

[3.3.1 Integration efforts 3-37](#_Toc535944269)

[3.3.2 Templates modifications 3-38](#_Toc535944270)

[3.3.3 Pipeline modifications 3-42](#_Toc535944271)

[3.3.3.1 COBilling pipeline 3-42](#_Toc535944272)

[3.3.3.2 COSummary pipeline. 3-44](#_Toc535944273)

[3.3.3.3 COPlaceOrder pipeline. 3-45](#_Toc535944274)

[3.3.4 Controller modifications 3-46](#_Toc535944275)

[3.3.4.1 COBilling controller. 3-46](#_Toc535944276)

[3.3.4.2 COSummary controller. 3-48](#_Toc535944277)

[3.3.4.3 OrderModel.js 3-49](#_Toc535944278)

[4. Release History 4-51](#_Toc535944279)

# Summary

This cartridge enables a Commerce Cloud store to use the Klarna Payments service.

The developer has to install the cartridge and integrate it into the online store following the instructions from this document. It is mandatory that the merchant has to open an account with Klarna and configure some items in Commerce Cloud Business Manager to make the integration working properly.

Klarna offers a test environment, so the integration can be tested before switching to the Klarna production environment.

The integration consists of an archive, which contains the following contents:

* Cartridge called ‘int\_klarna\_payments’, and optionally either "int\_klarna\_payments\_pipelines" or "int\_klarna\_payments\_controllers" depending on which architecture is to be employed.
* A site-template archive containing new attributes and settings
* This document
* The integration is based on the Site Genesis demo store provided by Commerce Cloud.

# Component Overview

## Functional Overview

Klarna Payments can receive payments in via the Klarna Payments API using JSON REST API and a JavaScript SDK. A result is returned right away. Through Klarna Payments you enable your consumers to try before they buy, finance purchases on your store via Klarna Credit, or make use of other payment methods made available by Klarna. Klarna offers these payment options via a widget, called Klarna Payments widget or just "the widget", which you add inline on your checkout page.

Klarna Payments uses the ‘redirect model’, where the customer is sent to the redirect\_url as a last step of the process before showing the confirmation page. The reason for this redirect is to allow Klarna to recognize the customer's device in future interactions. Klarna needs to interact with the customer's browser as first party (not in an iframe or similar). This is achieved by ‘bouncing’ the browser on a klarna.com page before presenting the confirmation page.

Payment results are sent asynchronously to a 'notification URL' at the Merchant (SCC) site. Payments can change status over the time (Accepted, Rejected and Pending) on Klarna Payments side, and these statuses are updated automatically on the Commerce Cloud side through sending these notifications.

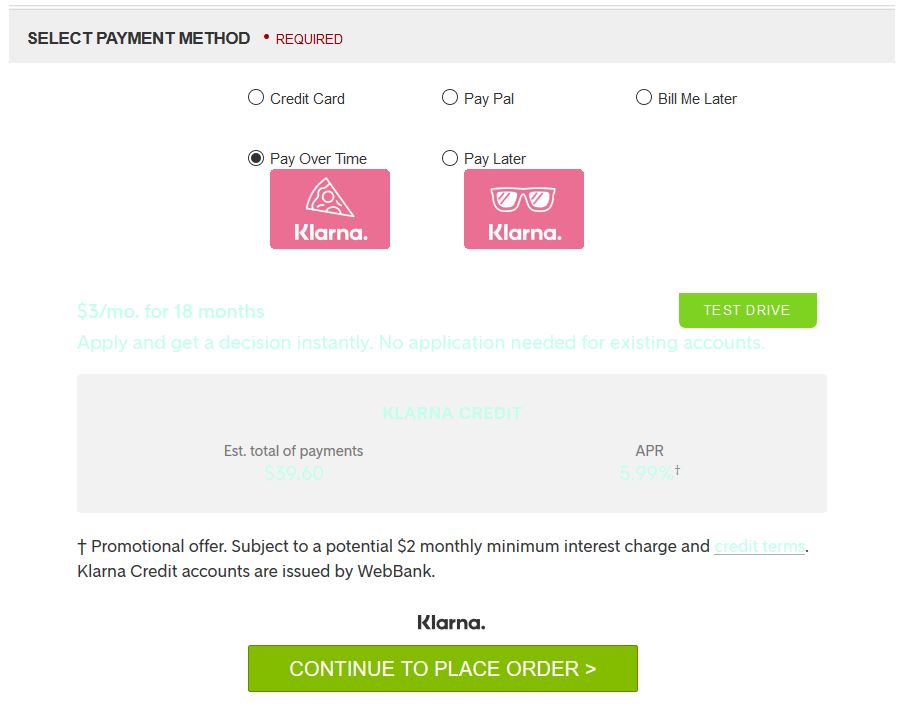
In case order is ACCEPTED order creation in SCC proceeds as usual, in case of REJECTED status, order is failed and in case of PENDING status the 'two phase ordering' approach is used, where the order is first created but only 'placed' after notification for ACCEPTED status is asynchronously received. If neither ACCEPTED, nor REJECTED status is received the order will stay in status ‘created’ until an action is taken by the merchant. Klarna payment status is saved in a custom attribute with id kpFraudStatus in the PaymentTransaction system object, and can be seen in BM on the order details Payment tab as below:



## Use Cases

1. Create an order in SCC using Klarna Payments as a payment method though Klarna Payments widget on the billing page. All the SCC OOTB checkout functionality remains in place, such as but not limited to: cart updates during checkout, checkout with applied coupon(s) code(s), checkout with applied product level promotion, checkout with applied order level promotion, checkout with applied shipping level promotion, checkout with applied order level promotion with bonus product.

Select 'Klarna' as the payment method on Billing page of checkout process and click the 'CONTINUE TO PLACE ORDER' button:



You will see the Order Confirmation page with payment method Klarna and can press the 'PLACE ORDER' button:



The customer is redirected to the redirect\_url. The reason for this redirect is to allow Klarna to recognize the customer's device in future interactions. Klarna therefore needs to interact with the customer's browser as first party (not in an iframe or similar). This is achieved by ‘bouncing’ the browser on a klarna.com page before presenting the SCC Order Summary page:



The newly created order can be the inspected in BM



Klarna Payments order id can be inspected in the Attributes tab of the order



Payment method details can be inspected on the Payment tab of the order, and it should be Klarna



Order can be further inspected in Klarna's Orders System



1. Refuse Klarna Payments as a payment method

Upon selecting 'Klarna' as the payment method on Billing page of checkout process and based on the information provided Klarna Payments can be refused as a payment method.



1. Klarna Payments not available – if Klarna API is not available, Klarna is not presented as a payment option



1. Order accepted after review - Instead of immediately accepting the order, it may happen that Klarna flags this transaction for additional review and accept it later on. The order is then created in the SCC system:



This order is marked with EXPORT\_STATUS\_NOTEXPORTED, confirmation status NOTCONFIRMED and NOTPAID.



Upon receiving approval from Klarna the order is placed:



In Klarna's Orders system the order is marked as accepted after review:



1. Order rejected after review - Instead of immediately accepting the order, it may happen that Klarna flags this transaction for additional review and reject it later on. The order is then created in the SCC system:



This order is marked with EXPORT\_STATUS\_NOTEXPORTED, confirmation status NOTCONFIRMED and NOTPAID.



Upon receiving rejection from Klarna the order is failed:



In Klarna's Orders system the order is marked as rejected after review:



1. Klarna Payment optionnot available for the current purchase – the customer is presented with an appropriate message when attempt to chose Klarna is made.



1. Gift Certificate / Klarna Payments split payment

Use Klarna Payments in combination with OOTB Site Genesis Gift Certificate functionality. When a gift certificate(s) is(are) redeemed in the checkout flow and remaining order amount is paid with Klarna Payments an order line item(s) ‘Gift Certificate’ is added to the Klarna Order reducing the amount owned with the gift certificate Payment Instrument(s).



1. Virtual Card Network as an alternative to standard order management

The VCN payment option can be enabled through the site preference as shown below:



This option is disabled by default. However, if standard order management is not a reasonable option for a Klarna integration, instead Klarna’s VCN, or virtual card network, can be enabled. When a customer places an order, the order is first booked in SFCC. If the order has a fraud\_status of PENDING, action is not taken on the order until receiving Klarna’s push notification that the fraud\_status has changed to ACCEPTED. Once an order has been accepted by Klarna, the merchant platform creates a VCN settlement, per the merchant card services (MCS) API.

Once a settlement has been created, the merchant platform has up to 3 hours to authorize the virtual credit card. Then, once the order has been fulfilled, the card funds should be captured. While Klarna is the original payment method of the order, the order will be settled with a credit card, not directly by Klarna.

In order to use the VCN option the merchant should:

* Enable VCN option in Site Preferences as shown above
* Generate a 4096 bit RSA key pair. Set the custom preference ‘vcnPublicKey’ with the value of the public key without the header and footer lines (begin and end public key) and the custom preference ‘vcnPrivateKey’ with the value of the private key without the header and footer lines (begin and end private key). As shown below:



**How to generate a 4096 bit RSA key pair:**

In order to generate an RSA keypair with a 4096 bit private key you can use the following openssl command:

***openssl genpkey -algorithm RSA -out private\_key.pem -pkeyopt rsa\_keygen\_bits:4096***

In order to extract the public key from an RSA keypair, you can use the following openssl command:

***openssl rsa -pubout -in private\_key.pem -out public\_key.pem***

In the folder where you have executed the above commands two new files will be created - public\_key.pem and private\_key.pem.

The contents of the files should look something like:

**public\_key.pem**

-----BEGIN PUBLIC KEY-----

MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAoNYG7l2G8nZa+22oBYZk

tV228lw3UE9WO4oxfknJtKEdHn84x55ULt8KQTh9NVtdeKC8nTfTgyvMt/GNCa18

xuZV/lGYDftKt85hbV5EjOum+StAIufEXvlBX7nMOMc1KyWm9kp2kbqd88mFIX63

KV94OoNEXcNatRDFYR+qz53+ifadDQtQ1slVNStdroCZDJ1+LxtBy9V+BdmsBK1E

RLsKh/JLXyWE24FJKV+z00s7TQkdWW/5ET12OGQYZsWo1yqgi9HplNvrisve8vWP

xaL4m8iZ3I/9yYdg7yANQbTxSJcbbRCgaaagPo30CNxeqU6qafY5g8vY3E52CoXH

DdO4UslX1qcuYIDhqaDzey6W+b8m755xLi+rqQyM4PBWL0J0dM3FVid8+4YKILex

3AKBFciqRCMHSOGaEeyrXKTjlAsghr9RS8PifvQRrL440cHzqw2vX0DvpjSWcmUJ

tW4wUq5RNSsobrxnVmoV6fj1z67Q/1P+l5Ie+oowdahR5ztVqJlO+2PNoX4I5VDs

/Pkz3f8wWVc3Mp2oNT244o+/NIiyRfPFaJJx7JAgrcvZt2nFAmY4QApXLFJCpgEM

wYucE4AH4gJKsh3KZbxRERrrO72bL2rxvWqBp/0h7DcMsV9sQs4BvxxIl6CF506F

ThzmclaKLBAyd5LALiXiPfkCAwEAAQ==

-----END PUBLIC KEY-----

**private\_key.pem**

-----BEGIN PRIVATE KEY-----

MIIJQQIBADANBgkqhkiG9w0BAQEFAASCCSswggknAgEAAoICAQCg1gbuXYbydlr7

bagFhmS1XbbyXDdQT1Y7ijF+Scm0oR0efzjHnlQu3wpBOH01W114oLydN9ODK8y3

8Y0JrXzG5lX+UZgN+0q3zmFtXkSM66b5K0Ai58Re+UFfucw4xzUrJab2SnaRup3z

yYUhfrcpX3g6g0Rdw1q1EMVhH6rPnf6J9p0NC1DWyVU1K12ugJkMnX4vG0HL1X4F

2awErUREuwqH8ktfJYTbgUkpX7PTSztNCR1Zb/kRPXY4ZBhmxajXKqCL0emU2+uK

y97y9Y/FovibyJncj/3Jh2DvIA1BtPFIlxttEKBppqA+jfQI3F6pTqpp9jmDy9jc

TnYKhccN07hSyVfWpy5ggOGpoPN7Lpb5vybvnnEuL6upDIzg8FYvQnR0zcVWJ3z7

hgogt7HcAoEVyKpEIwdI4ZoR7KtcpOOUCyCGv1FLw+J+9BGsvjjRwfOrDa9fQO+m

NJZyZQm1bjBSrlE1KyhuvGdWahXp+PXPrtD/U/6Xkh76ijB1qFHnO1WomU77Y82h

fgjlUOz8+TPd/zBZVzcynag1Pbjij780iLJF88VoknHskCCty9m3acUCZjhAClcs

UkKmAQzBi5wTgAfiAkqyHcplvFERGus7vZsvavG9aoGn/SHsNwyxX2xCzgG/HEiX

oIXnToVOHOZyVoosEDJ3ksAuJeI9+QIDAQABAoICACRkaUsUNI22RB3yEPu3DiCP

pO6v+QAeA4gTW+GUdqR9dCZLaSCZ7bhxVVOuoX4qPzslO6hjUmOyzG6upFgVPk+P

HNQfyEUZoC148Eib9OziAXUN2URMpv1KbwVm+BO814X8zguai7uru0PHTG1oy677

4Ct1OknxAxxHQDIaxT6XJFo5SA4EinUfNz2Bo3/xry/QjxW/mCK0GwDd4PNp9TGM

FPTv2SgdSDOWzGQlOH5N3owuzMpI8NV6z74wv+i5Ptv41Dzu8WhyXpiYSsk00SRK

HPC68j2bAzTPghp5aSZ9976SGm2SPonJXyboXdiHbI/osdyqDxeIT3iB9GmrHX/i

kHPGJCh7fRZvqj39Hc+IxYjabwW3rDeDIPB7ab9z1KLF4z1D6AZOKCPyTaDRdQ1Q

eDi7LwDmk7NHEPrmF/nIcguQdqbIbmFO2zEs0TOe6y4uBMndRsbQprTNSMUdBkrA

lNaYVSTQ1Z0Y/8DZDpGcyS1OnJv74F15uDjKN6/ov991mZ1JrZ+V2sdS3EDUlmvP

6thQKwI7Ln6h+ApHtWUG1NmvQe5gJE0qAeJ9b45clUzIRUwhVmEp8NoIJh0kAjaN

d4lk7xy9ZRDUY5yekPeYrJPShjsHAyEoktJIjRufI2UUq3uxNjjICoQcOVGfNDIS

YTTPwpu1pmC0C+rh2fgBAoIBAQDRultRArvtc2JKhVOUyZk88zd9kvrI6fNiyKmi

HgiWf7qkTPD9xhOQWDw3iwRFQAD+YkgV5MCBO8wp8oO8GEsOCI+XZWExOcPT0Vfj

PZHiQrTFnlfG/+fAO14xLf3j3ED4YQXdHOKI3xoLknQx/EydLoctxgkkpgWLrsA7

DwdSAg1/0sBvaHY27ogAfdimHdaKZ5OAe4a9k1qP3xVZBuOe8Sd65unBavUJLDuv

ikeNmkSVgW1sm55/729JIr63USHF76It+vE1cdZ+vKg5vYotsQgPzvNBmUO/E8Gj

zMXQRfqfvEDlNXEX0rCupTkw1G6AGTwQc/NPzyr/LTpLe6UBAoIBAQDEUjTiG11V

hf7WjdG3gctRlr+mYapQHgXdVLx2QSaqUYid+0QXK11YfJlsRB6nwa+OED83RfP0

lIFqxpzudSLPmoDuIBT7Dl5c/aleyKs/siUusP8QVDXk6OAR84XSytC35sIRV7pE

VMuBL91jfkQ0Lf/PreslK/kI6Yvwwp4qrHK6/f9TgciHclYtf+/oti4ky6GJgfmP

fmuCqjxmUKbXXFPd5RbL2THGOowilb8zDLjf3RlbjlQFqogAk6H9hp2V0VZLiJHp

UWM3z3zxDWeDaqJ08sHuk/rA9QpsVTu8IGTQsxdj8JwluN1Q+YZiOuPiSENBqPzT

V3exexzo3sD5AoIBAGU3qEyPojz1+9D1SaI8LW2CABzlq4z9g84ABAZOslxX5q7W

x1PinZyDSQSRXg1B13jt29ZdIR79ygnQlg1YOBjcvtgVQHPuafk3RlBQbbCh+vaI

9dn/tUxMGqhnhunKaby1rovJHfdqnPpKwzNAjYUqaGkJ822xhmmke/fEyAanIPa4

stDRvIPEWPTLx5xcOCdx13khpKSnkgRvaLEfpwkVX7Vr7hK/2OSFaYTNmrzXYBQ7

c6D/9d3Oo4nLb/mu+Tq67S19t53Qg/GEgTfkpuRoVPi0KyhUnKKCGWlBMZLTwyIG

S9eTFDKoJ0cSTGipjW7bPua93wZ8eEbRABpf4QECggEANNhQBeEJ0aCdBVHtdrEI

crDaa8X0W1aJi5dol4hYCRajaKsfHAF/QfdgMQVxHwUC5YG4En/Q+DAVWhGWYpXD

RhC3zeFy5FVszyk0sx/fAOlKGvRn5BRW4YRR9GMRzbjsT+RcruBnckdE9ERXGpX9

c/JB3rxZBIt+oIiFM8yfWKtMwsrmNKtFuDftvJeok4KejycFF4eWDqsf828xjPT+

xA/FP4CQD1UqkcpmuFSIwAwXo6LXVY7NTS0nKMiUnTLkLlTIHtLnO9+9jmNapWRP

Tc+hZUuHKlpI8DHFmX2j87LgkFD05eD5lynY4RgZtU1W1C1RdVYwoA72WB7knEaB

uQKCAQAH9s67P/7fFX9dfEans3PHU4nGjD8dJ8eoNQ6DhBMydZpGWI5ZUeEBZDRk

0cBOeRs5BOcS43Em9kETpzawyCwxmnwzl+CzoPzMQcTw9tXomF9HG6RJ9XBdJfGA

ALAwCd4bASxmFM6guSP5GKnZ9aY3tR3tWWDfr7f9z8wOewzzpPclwRh009fPe4TC

NXoEm1MELJVeUieDSLKZgjgCw8WHGqLItONpA0/fwSM2gIcxETVV7qx3aPuJzCVh

LQZoBLQk3UMKsWDdpzeBdiERe66NAgVk92Xe7SY9EY2vymaq761i1x1vlprT27qp

240LDJawqM0IraKmdCvWjofWSaOU

-----END PRIVATE KEY-----

* Finally, you need to send the generated public key to Klarna. It will be used to encrypt the VCN pan and csc on our side. After confirmation from Klarna side that the key has been successfully added to your profile you would be able to use VCN option.
* Authorize the virtual credit card

The virtual credit card details are stored as order level custom attributes, and can be accessed through the Order object as follows:

* Order.custom.kpVCNBrand – virtual credit card brand;
* Order.custom.kpVCNCSC – virtual credit card csc;
* Order.custom.kpVCNExpirationMonth – virtual credit card expiration month;
* Order.custom.kpVCNExpirationYear - virtual credit card expiration month ;
* Order.custom.kpVCNHolder - virtual credit card holder;
* Order.custom.kpVCNPAN - virtual credit card number;
* Order.custom.kpIsVCN – Boolean property indicating whether VCN is used.

These attributes can be inspected in the BM order details in the attributes tab as shown below:



It is up to the merchant to use the credit card details as described above to authorize the virtual credit card. As an example authorization is done with calling BASIC\_CREDIT-Authorize as below (pipeline version):



You need to add your own logic in KLARNA\_PAYMENTS-Authorize in the place of BASIC\_CREDIT-Authorize call node.

You would also need to add your own card authorization login in the place of BASIC\_CREDIT-Authorize call node in the KLARNA\_PAYMENTS-Notification pipeline as shown below:



## Limitations, Constraints

The merchant will need a configured Klarna Payments account.

The merchant can style the Klarna Payments widget (skin), to make it matching the look and feel of their store. The list with the graphic elements that can be customized, through site preferences is as follows:

* "color\_details" (site preference kpColorDetails): "#C0FFEE"
* "color\_button" (site preference kpColorButton): "#C0FFEE"
* "color\_button\_text" (site preference kpColorButtonText): "#C0FFEE"
* "color\_checkbox" (site preference kpColorCheckbox): "#C0FFEE"
* "color\_checkbox\_checkmark" (site preference kpCheckboxCheckmark): "#C0FFEE"
* "color\_header"(site preference kpColorHeader): "#C0FFEE"
* "color\_link"(site preference kpColorLink): "#C0FFEE"
* "color\_border"(site preference kpColorBorder): "#C0FFEE"
* "color\_border\_selected"(site preference kpBorderSelected): "#C0FFEE"
* "color\_text"(site preference kpColorText): "#C0FFEE"
* "color\_text\_secondary"(site preference kpColorTextSecondary): "#C0FFEE"
* "radius\_border"(site preference kpRadiusBorder): "0px"



Successful payment authorizations (Klarna Payments status APPROVED) lead to a 'Paid' order payment status and ‘Ready for Export’ export status

Refused and pending payment authorizations (Klarna Payments statuses REJECTED and PENDING) lead to a 'Not Paid' order payment status, and ‘Not Exported’ export status

Cancels and refunds lead to a 'Not Paid' order payment status (even if the status was 'Paid' before), and ‘Not Exported’ export status

The cartridge makes use of the Klarna Payments JSON REST API and a JavaScript SDK.

## Compatibility

Based on Commerce Cloud 17.1.0 and SiteGenesis 103.1.3

## Privacy, Payment

**NO CREDIT CARD DATA IS BEING PROCESSED OR STORED WITHIN SCC IN THE SCOPE OF THIS INTEGRATION**

The cartridge supports sending additional information on the customer's past purchase history when turned on for a merchant. This automatically happens for non-EU countries (US).

The data send to Klarna is customizable through a hook. An example hook contents are provided in int\_klarna\_payments/scripts/EMD/KlarnaPaymentsBuildEMD\_hooks.js. The merchant can customize the contents of the BuildEMD function. Current Basket or Order object can be accessed thorough args. LineItemCtnr. This script should return a JSON string to be used as a value for the body sub-field of the attachment field as [described here](https://developers.klarna.com/api/#payments-api__create-a-new-orderattachment__body)

If the example KlarnaPaymentsBuildEMD\_hooks.js file is used unchanged the data send to Klarna is by the following schema:

{

"$schema": "http://json-schema.org/draft-03/schema#",

"id": "http://klarna.com/v2/emd#",

"description": "Extended Merchant Data Payload Schema",

"type": "object",

"properties": {

"customer\_account\_info": {

"type": "array",

"items": {

"type": "object",

"properties": {

"unique\_account\_identifier": {

"type": "string",

"maxLength": 24

},

"account\_registration\_date": {

"description": "ISO 8601 e.g. 2012-11-24T15:00",

"type": "string",

"format": "date-time",

"pattern": "^[0-9][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]T[0-2][0-9]:[0-5][0-9](:[0-5][0-9]){0,1}Z{0,1}$"

},

"account\_last\_modified": {

"description": "ISO 8601 e.g. 2012-11-24T15:00",

"type": "string",

"format": "date-time",

"pattern": "^[0-9][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]T[0-2][0-9]:[0-5][0-9](:[0-5][0-9]){0,1}Z{0,1}$"

}

}

}

},

"payment\_history\_full": {

"type": "array",

"items": {

"type": "object",

"additionalProperties": false,

"properties": {

"unique\_account\_identifier": {

"type": "string"

},

"payment\_option": {

"type": "string",

"enum": ["card", "direct banking", "non klarna credit", "sms", "other"]

},

"number\_paid\_purchases": {

"type": "integer"

},

"total\_amount\_paid\_purchases": {

"type": "number"

},

"date\_of\_last\_paid\_purchase": {

"description": "ISO 8601 e.g. 2012-11-24T15:00",

"type": "string",

"format": "date-time",

"pattern": "^[0-9][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]T[0-2][0-9]:[0-5][0-9](:[0-5][0-9]){0,1}Z{0,1}$"

},

"date\_of\_first\_paid\_purchase": {

"description": "ISO 8601 e.g. 2012-11-24T15:00",

"type": "string",

"format": "date-time",

"pattern": "^[0-9][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]T[0-2][0-9]:[0-5][0-9](:[0-5][0-9]){0,1}Z{0,1}$"

}

}

}

}

}

}

Example data:

{

"extra\_merchant\_data": {

"content\_type": "application/vnd.klarna.internal.emd-v2+json",

"body": {

"customer\_account\_info": [{

"unique\_account\_identifier": "5509d9f7c8720c0e4575154b",

"account\_registration\_date": "2015-03-18T20:03:03Z",

"account\_last\_modified": "2015-03-18T20:03:03Z"

}],

"purchase\_history\_full": [{

"unique\_account\_identifier": "5509d9f7c8720c0e4575154b",

"payment\_option": "card",

"number\_paid\_purchases": "23",

"total\_amount\_paid\_purchases": "140023",

"date\_of\_last\_paid\_purchase": "2015-03-18T20:03:03Z",

"date\_of\_first\_paid\_purchase": "2015-03-18T20:03:03Z"

}]

}

}

}

# Implementation Guide

To setup and implement Klarna Payments Salesforce Commerce Cloud Cartridge, the following steps must be followed:

## Setup

Klarna Payments integration has three cartridges:

* int\_klarna\_payments which implements the core storefront functionality;
* int\_klarna\_payments\_pipelines which implements the storefront functionality with pipeline based approach;
* int\_klarna\_payments\_controllers which implements the storefront functionality with controllers based approach;

### int\_klarna\_payments

The int\_klarna\_payments cartridge has the following components:

Cartridge name

* int\_klarna\_payments

Scripts

* /checkout
  + - SetPendingOrderStatus.ds
* /common
  + - KlarnaPaymentsApiContext.js
    - KlarnaPaymentsHttpService.ds
* /EMD
  + - KlarnaPaymentsBuildEMD\_hooks.js
* /locale
  + - KlarnaPaymentsGetCountryFromLocale.ds
    - KlarnaPaymentsGetLocale.ds
* /order
  + - KlarnaPaymentsAcknowledgeOrder.ds
    - KlarnaPaymentsCancelOrder.ds
    - KlarnaPaymentsCreateOrder.ds
    - KlarnaPaymentsOrderModel.js
    - KlarnaPaymentsOrderRequestBuilder.js
* /session
  + - KlarnaPaymentsCreateSession.ds
    - KlarnaPaymentsSessionModel.js
    - KlarnaPaymentsSessionRequestBuilder.js
    - klarnaPaymentsUpdateSession.ds
* /util
  + - Builder.js
    - KlarnaPaymentsConstants.js
* /VCN
  + - klarnaPaymentsVCNSettlement.ds
* createKlarnaPaymentInstrument.ds
* hooks.json
* klarnaRemovePreviousPI.ds

Static

* /default
  + - js
      * klarna-payments.js

Templates

* /default
  + - klarnapayments
      * http\_200.isml
      * modules.isml
      * paymentmethodshelper.isml
      * shippingaddresshelper.isml
      * klarnapaymentblock.isml
    - klarnapaymentscategories.isml
* /resources
  + - klarnapaymentsresources.isml

### int\_klarna\_payments\_pipelines

The int\_klarna\_payments\_pipelines cartridge has the following components:

Cartridge name

* int\_klarna\_payments\_pipelines

Pipelines

* KLARNA\_PAYMENTS.xml

### int\_klarna\_payments\_controllers

The int\_klarna\_payments\_controllers cartridge has the following components:

Cartridge name

* int\_klarna\_payments\_controllers

Controllers

* KLARNA\_PAYMENTS.js

Scripts

* hooks.json

Other

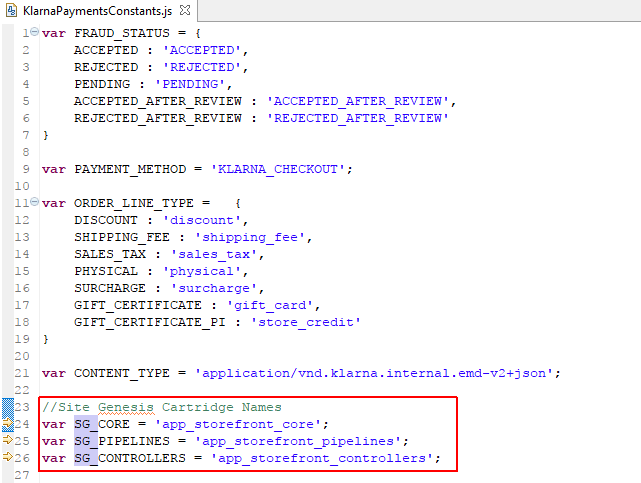
* package.json

### References to Site Genesis Cartridges

Klarna Payments cartridge is developed assuming storefront cartridge naming conventions as:

* app\_storefront\_core
* app\_storefront\_ pipelines
* app\_storefront\_ controllers

Wherever code from Site Genesis cartridges is used it is referenced with the above cartridge names. The names of the Site Genesis cartridges have been referred in int\_klarna\_payments cartridge in int\_klarna\_payments/cartridge/scripts/util/KlarnaPaymentsConstants.js as shown below:



If the base cartridges are named differently the variables shown above need to be renamed to the correct paths based on the individual integration.

## Configuration

### Metadata Import

Find the archive ‘site-template.zip’ in the installation package, it contains metadata files. Please review the archive, do the necessary modifications, and import it through BM Administration > Site Development > Site Import & Export section. The only required modification is renaming the root folder of the archive to the ID of your site.

Check the fields which will be updated after importing file ‘site-template.zip’, and be sure that there won’t be any conflict with the existing fields.

The following configuration items will be added after you import ‘site-template.zip’ archive:



See full list below.

Besides configuration parameters, the following order attributes will be added:



Custom Object Definition:

After importing metadata there will be a site specific custom object to store Klarna countries:



Note: The same custom object is used by Klarna Checkout

KlarnaCountries custom object holds the corresponding locale for each Klarna Payments endpoint. Even if you have locales that are not supported by Klarna Payments we recommend to make a corresponding entry in the custom object for that locale (you can use one of the other endpoints). Thus on the billing page of the unsupported locale you will have the Klarna Payments widget showing an appropriate message like below:



If Klarna Payments is not supported as a payment method and you have not updated the locale entry in the KlarnaCountries custom object as stated above, the customer will be presented with the following more general message on billing page:



### Cartridge Path

If using the controller based integration, cartridge path should be setup as such under Sites / Manage Sites /

app\_storefront\_controllers:app\_storefront\_core:int\_klarna\_payments\_controllers:int\_klarna\_payments



If using pipelines, the cartridge path should be similar to:

app\_storefront\_pipelines:app\_storefront\_core:int\_klarna\_payments\_pipelines:int\_klarna\_payments



### Klarna Payments Configurations

Also, there is a page in Business Manager where you have to configure Klarna Payments cartridge, you can find it in Site Preferences > Custom Preferences > Klarna\_Payments:



Here is the description of the Site Preferences fields which have to be configured:

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Attribute ID** | **Description** |
| Send product\_url and image\_url | sendProductAndImageURLs | If set to true product\_url and image\_url fields in Klarna Payments create session call will be included in API call, otherwise product\_url and image\_url fields will not be populated. |
| Merchant Reference 2 Mapping | merchant\_reference2\_mapping | The field from SCC order (basket) object that is mapped to merchant\_reference2 field from klarna API request.  Has to be one of the class attributes of SCC LineItemCtnr. Note that for complex data structures result may not always be as expected. |
| Border Color Preference | kpColorBorder | CSS hex color to be used in Klarna Payments iFrame |
| Border Selected Color Preference | kpColorBorderSelected | CSS hex color to be used in Klarna Payments iFrame |
| Button Color Preference | kpColorButton | CSS hex color to be used in Klarna Payments iFrame |
| Button Text Color Preference | kpColorButtonText | CSS hex color to be used in Klarna Payments iFrame |
| Checkbox Color Preference | kpColorCheckbox | CSS hex color to be used in Klarna Payments iFrame |
| Checkbox Checkmark Color Preference | kpColorCheckboxCheckmark | CSS hex color to be used in Klarna Payments iFrame |
| Details Color Preference | kpColorDetails | CSS hex color to be used in Klarna Payments iFrame |
| Header Color Preference | kpColorHeader | CSS hex color to be used in Klarna Payments iFrame |
| Link Color Preference | kpColorLink | CSS hex color to be used in Klarna Payments iFrame |
| Text Color Preference | kpColorText | CSS hex color to be used in Klarna Payments iFrame |
| Secondary Text Color Preference | kpColorTextSecondary | CSS hex color to be used in Klarna Payments iFrame |
| Border Radius Preference | kpRadiusBorder | Size (in pixels) of the border radius to be used in Klarna Payments iFrame |
| Attachments | kpAttachments | Flag to swicth on/off the using of attachments when creating a session. Default is OFF. |
| Not available message on billing page | kpNotAvailableMessage | The Klarna Payment not available message on billing page. JSON string holding country code and corresponding message string. For example:  {      **"GB"**:"Klarna Payment not available",    **"default"**:"Klarna Payment not available" } |
| Virtual Card Network Enabled | kpVCNEnabled | If set to true SFCC will create Virtual Card Network settlement from every Klarna order. Default is false |

### Klarna Payments Logo and Payment Option Name

Payment option name can be changed in BM Merchant Tools > Ordering > Payment Methods as shown below.



Payment option name can be localized for the specific country and language through the language dropdown menu on the right, as shown below:



Logo can be changed from BM Merchant Tools > Ordering > Payment Methods. Enter the file name with the logo in the image field on payment method details view (see screenshot below)



### Services Configuration

After the metadata import you should have klarna service ready in Administration > Operations > Services like so:



You need to configure Klarna Payments services credentials. Go to Credentials tab as shown below:



You need to edit the existing credentials or create new ones using the credentials provided to you by Klarna and the API URL for the specific location, see for example below:



The Klarna Payments API is accessible through a few different URLS. There are different URLs for testing and for making live purchases as well as different URLs for depending on if you are based in Europe or in the U.S.

**Live environment**

The API for the European production environment can be found at

* https://api.klarna.com/

The API for the U.S. production environment can be found at

* https://api-na.klarna.com/

**Testing environment**

The API for the European testing environment can be found at

* https://api.playground.klarna.com/

The API for the U.S. testing environment can be found at

* https://api-na.playground.klarna.com/

### Custom attributes

ORDER system object has been extended with the following custom attributes:

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Attribute ID** | **Description** |
| Klarna Payments Order ID | kpOrderID | Holding the Klarna Payments Order ID, if Klarna Payments is used for the specific order |
| VCN Brand | kpVCNBrand | Holding the Klarna Payments Virtual Card Network card Brand |
| VCN CSC | kpVCNCSC | Holding the Klarna Payments Virtual Card Network card csc (encrypted) |
| VCN Expiration Month | kpVCNExpirationMonth | Holding the Klarna Payments Virtual Card Network card expiration month |
| VCN Expiration Year | kpVCNExpirationYear | Holding the Klarna Payments Virtual Card Network card expiration year |
| VCN PAN | kpVCNPAN | Holding the Klarna Payments Virtual Card Network card pan (encrypted) |
| VCN Holder | kpVCNHolder | Holding the Klarna Payments Virtual Card Network card holder name |
| Is VCN Used | kpIsVCN | True if VCN is used for payment of the order, otherwise false |

Payment Transaction system object has been extended with the following custom attributes:

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Attribute ID** | **Description** |
| Fraud Status | kpFraudStatus | Klarna Payments order fraud status |

## Custom Code

Integration may vary based on the customers' storefront. Site Genesis version 2 is used to demonstrate Klarna Payments integration.

### Integration efforts

The following steps are needed to complete the integration:

* Install the cartridges
* Change and import ‘site-template.zip’ (system objects and custom objects)
* Do the required code changes
* Configure your merchant's Klarna Payments Account in the Klarna back office
* Configure Klarna Payments parameters in Commerce Cloud
* Test
* Create styling for Klarna Widget (optional)

These steps should not require more than a few hours.

Note that the name of your storefront site ID should match the name in the ‘site-template.zip’, otherwise you will get import errors.

Also, note that the payment methods ‘Klarna’ and the payment processors 'KLARNA\_PAYMENTS' will be created or updated from the ‘site-template.zip’.

### Templates modifications

The following changes should be made regardless of whether a controller or a pipeline integration approach are being used.

To integrate Klarna Payments the following storefront cartridge templates should be updated:

* default/checkout/billing/billing.isml
* default/checkout/summary/summary.isml
* default/checkout/billing/paymentmethods.isml
* default/checkout/shipping/minishipments.isml

**default/checkout/summary/summary.isml**

Add Code:

<script><isinclude template="/resources/klarnapaymentsresources.isml"/></script>

<script type="text/javascript" src="${URLUtils.staticURL('/js/klarna-payments-finalize.js')}"></script>

<script src=" https://x.klarnacdn.net/kp/lib/v1/api.js" async></script>

Before</isdecorate> closing tag as shown below:

****

**default/checkout/billing/billing.isml**

Add Code:

<script><isinclude template="/resources/klarnapaymentsresources.isml"/></script>

<script type="text/javascript" src="${URLUtils.staticURL('/js/klarna-payments.js')}"></script>

<script src=" https://x.klarnacdn.net/kp/lib/v1/api.js" async></script>

Before</isdecorate> closing tag as shown below:



**default/checkout/billing/paymentmethods.isml**

Add code:

<isif condition="${paymentMethodType.value === 'Klarna'}">hide</isif>

After <div class="form-row label-inline close to line 18 as shown below:



Add Code:

<isinclude template="klarnapayments/klarnapaymentscategories.isml"/>

After </isloop> Close to line 28 as shown below:



Add Code:

<iscomment>Klarna Payments --------------------------------------------------------------</iscomment>

<isinclude template="klarnapayments/klarnapaymentblock.isml"/>

Before</fieldset> closing tag, close to line 150 as shown below:



**default/checkout/shipping/minishipments.isml**

Add the following code:

<isinclude template="klarnapayments/modules.isml"/>

In the end begining of the file as shown below:



Add the following code:

<iskpaddresshelper p\_address="${shipment.shippingAddress}"/>

After <isminicheckout\_address p\_address="${shipment.shippingAddress}"/> as as shown below:



### Pipeline modifications

If using a pipeline based SiteGenesis integration, additionally follow the instructions in this chapter. If integrating via the controller based model see next chapter.

To use Klarna Payments cartridge you will need to make the following changes in pipelines.

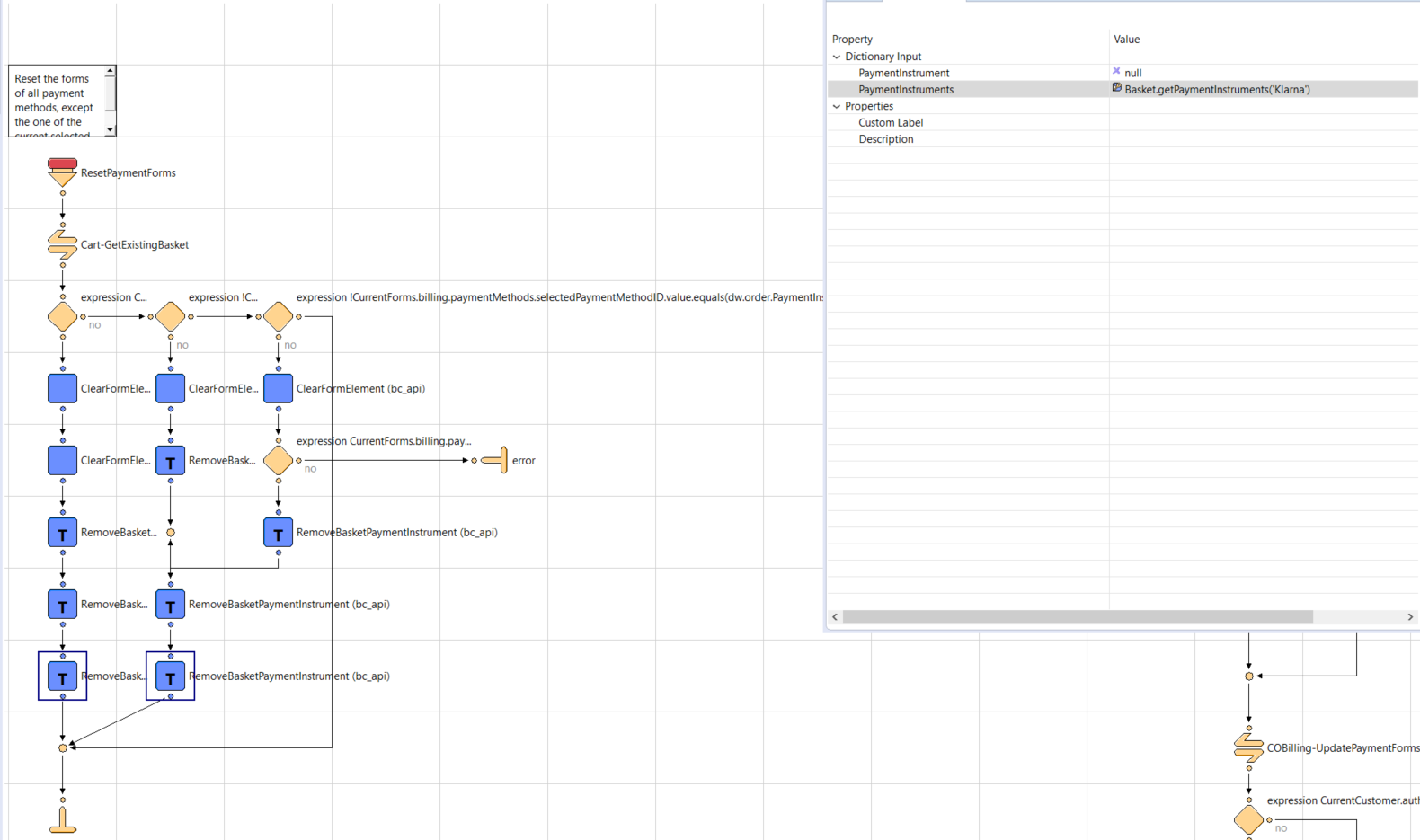
### COBilling pipeline

Go to COBilling pipeline, Start node and add KLARNA\_PAYMENTS-CreateSession call node entry point before checkout/billing/billing -> Billing interaction continue node (see screen shot below)



In pipeline COBilling-ResetPaymentForms, add two pipelets 'com.demandware.pipelet.basket.RemoveBasketPaymentInstrument' at the end of the pipeline.

In Dictionary Input 'PaymentInstruments', set it with value: Basket.getPaymentInstruments('Klarna').



### COSummary pipeline.

Go to COSummary pipeline, Submit node and add KLARNA\_PAYMENTS-Redirect call node entry point before COSummary-ShowConfirmation jump node (see screen shot below)



### COPlaceOrder pipeline.

Go to COPlaceOrder pipeline, PlaceOrder node and add KLARNA\_PAYMENTS-PendingOrder call node entry point before PlaceOrder script node node (see screen shot below). Create ‘pending’ and ‘error’ transitions and end nodes as shown on the screenshot



### Controller modifications

If using a controller based SiteGenesis integration, additionally follow the instructions in this chapter. If integrating via the pipeline method please see Pipeline modifications chapter above.

### COBilling controller.

Go to COBilling controller, returnToForm method and add the following code block:

**try**

{

require('int\_klarna\_payments\_controllers/cartridge/controllers/KLARNA\_PAYMENTS.js').CreateSession();

} **catch**( e )

{

require( 'dw/system/Logger' ).getLogger( 'COBilling.js' ).error( 'Klarna Create Session Error: {0}', e );

}

*after pageMeta.update (see screen shot below)*



In the method resetPaymentForms(), add the command for the three conditions: cart.removePaymentInstruments(cart.getPaymentInstruments('Klarna'));

***function*** resetPaymentForms**()** **{**

***var*** cart **=** app**.**getModel**(**'Cart'**).**get**();**

***var*** status **=** Transaction**.**wrap**(*function*** **()** **{**

***if*** **(**app**.**getForm**(**'billing'**).**object**.**paymentMethods**.**selectedPaymentMethodID**.**value**.**equals**(**'PayPal'**))** **{**

app**.**getForm**(**'billing'**).**object**.**paymentMethods**.**creditCard**.**clearFormElement**();**

app**.**getForm**(**'billing'**).**object**.**paymentMethods**.**bml**.**clearFormElement**();**

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**PaymentInstrument**.**METHOD\_CREDIT\_CARD**));**

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**PaymentInstrument**.**METHOD\_BML**));**

// remove Klarna PaymentInstrument

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**'Klarna'**));**

**}** ***else*** ***if*** **(**app**.**getForm**(**'billing'**).**object**.**paymentMethods**.**selectedPaymentMethodID**.**value**.**equals**(**PaymentInstrument**.**METHOD\_CREDIT\_CARD**))** **{**

app**.**getForm**(**'billing'**).**object**.**paymentMethods**.**bml**.**clearFormElement**();**

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**PaymentInstrument**.**METHOD\_BML**));**

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**'PayPal'**));**

// remove Klarna PaymentInstrument

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**'Klarna'**));**

**}** ***else*** ***if*** **(**app**.**getForm**(**'billing'**).**object**.**paymentMethods**.**selectedPaymentMethodID**.**value**.**equals**(**PaymentInstrument**.**METHOD\_BML**))** **{**

app**.**getForm**(**'billing'**).**object**.**paymentMethods**.**creditCard**.**clearFormElement**();**

***if*** **(!**app**.**getForm**(**'billing'**).**object**.**paymentMethods**.**bml**.**ssn**.**valid**)** **{**

***return*** ***false*;**

**}**

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**PaymentInstrument**.**METHOD\_CREDIT\_CARD**));**

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**'PayPal'**));**

// remove Klarna PaymentInstrument

cart**.**removePaymentInstruments**(**cart**.**getPaymentInstruments**(**'Klarna'**));**

**}**

***return*** ***true*;**

**});**

***return*** status**;**

**}**

### COSummary controller.

Go to COSummary controller, submit() method and add the following code block:

**try**

{

require('int\_klarna\_payments\_controllers/cartridge/controllers/KLARNA\_PAYMENTS.js').Redirect();

} **catch**( e )

{

require( 'dw/system/Logger' ).getLogger( 'COSummary.js' ).error( 'Klarna Redirect Error: {0}', e );

}

***before*** showConfirmation(placeOrderResult.Order)(see screen shot below)



### OrderModel.js

Go to OrderModel.js, placeOrder method and add the following code block:

**if**( session.privacy.KlarnaPaymentsFraudStatus === 'PENDING' )

{

**try**

{

require('int\_klarna\_payments\_controllers/cartridge/controllers/KLARNA\_PAYMENTS.js').PendingOrder();

} **catch**( e )

{

require( 'dw/system/Logger' ).getLogger( 'OrderModel.js' ).error( 'Klarna Payments Pending Order Error: {0}', e );

}

**return**;

}

**before**var placeOrderStatus = OrderMgr.placeOrder(order); (see screen shot below)



# Release History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| 17.1.0 |  | Initial release |
| 18.1.0 |  | Implement payments endpoint |