Klarna Payments for SG

Version 23.2.0

Table of Contents

1. Summary

The **Klarna Payments SG cartridge** enables integration of Klarna Payment solution on Commerce Cloud Storefront. The integration provides merchants the flexibility to offer choice of multiple Klarna Payment products on the Commerce Cloud checkout.

This document contains the instructions for a developer to install the cartridge and integrate it on the Salesforce Commerce Cloud site. The cartridge is fully compatible with the SiteGenesis JavaScript Controllers (SGJS).

Merchant teams are required also to configure the cartridge with the valid merchant credentials and site configurations in Commerce Cloud Business Manager to enable Klarna payments methods in the checkout.

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

- Cartridge called “int_klarna_payments” and “int_klarna_payments_controllers” to be imported.
- A site-template archive containing new attributes and settings.
- This document for Site Genesis (Klarna Payments Integration Guide).
- The integration is based on the SiteGenesis demo store provided by Commerce Cloud.

It is a requirement that Merchant sign a contract for integration support and production go-live with Klarna.

Klarna offers a playground (test) environment, so the integration can be tested before switching to the Klarna production environment. Based on the contract, Klarna shall provide assistance with integration and testing prior to sign-off for go-live.

2. Component Overview

2.1. Functional Overview

Key Features:

- Integrate Klarna Payments using best practices on international sites based (markets in North America, Europe, Oceania)
- Enable multiple payment products for customer in Pay Now, Pay Later and Pay Over Time categories
- Fast integration/go-live with virtual card-based integration approach for settlement
- Handle Notification: pending status updates (reject/accept) for suspected orders post review
- Site managers can customize the Klarna Payments widget styling displayed in checkout, to match the style guide of merchant website(s)
- GDPR (EU) compliant checkout flow
- Multi Shipping Address support
- Supports Klarna authorize with finalize for Bank Transfer methods (Pay Now)
- Enable Onsite Messaging placements on PDP, Cart, Header, Footer, and dedicated Info page
- Enable Klarna's Express Button on Cart page (*US) for faster checkout
- BOPIS (Buy Online, Pickup in Store) support including extra merchant data
- Support for Klarna Payment Method based promotions
- Support for adjusted price promotions with Gross Tax Policy
- Support disabling Payment method for authorization rejection
- Support Auto-Capture
- Support for custom service rate limits configuration

Klarna Payment cartridge makes use of the Klarna Payments JSON REST API and a JavaScript SDK to integrate on the storefront. Klarna Payment enables consumers to choose from the different

payment method products offered by Klarna. Multiple Klarna products are available within the categories “Pay Now”, “Pay Later” and “Pay Over Time”. The cartridge integration displays payment options via a widget (iframe) added inline on the billing page, referred to as Klarna widget or just “the widget”. The widget with information about the payment method is displayed to the customer when the individual clicks on the Klarna payment method.

Customers can authorize the payment after reviewing the payment method terms and clicking Place Order button. The Klarna order is created, and customer is re-directed to the confirmation page.

Orders successfully placed with Klarna return a Fraud Status: ACCEPTED and displayed in Business manager (BM).

With ACCEPTED status, order creation in SCC proceeds as usual. 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:

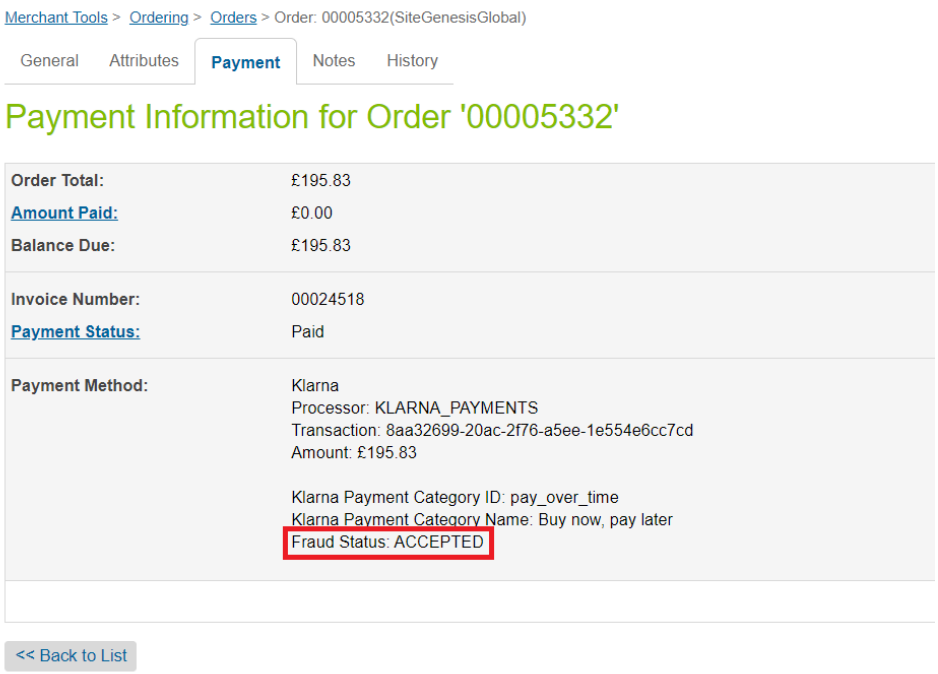


Figure 1 Klarna Payment Details in BM

An alternate flow when PENDING status is returned for Klarna order creation, the SCC order creation proceeds with modified statuses. If later a Klarna notification with updated fraud status

FRAUD_RISK_ACCEPTED is returned, SCC order status is updated and returns to the usual flow. 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:

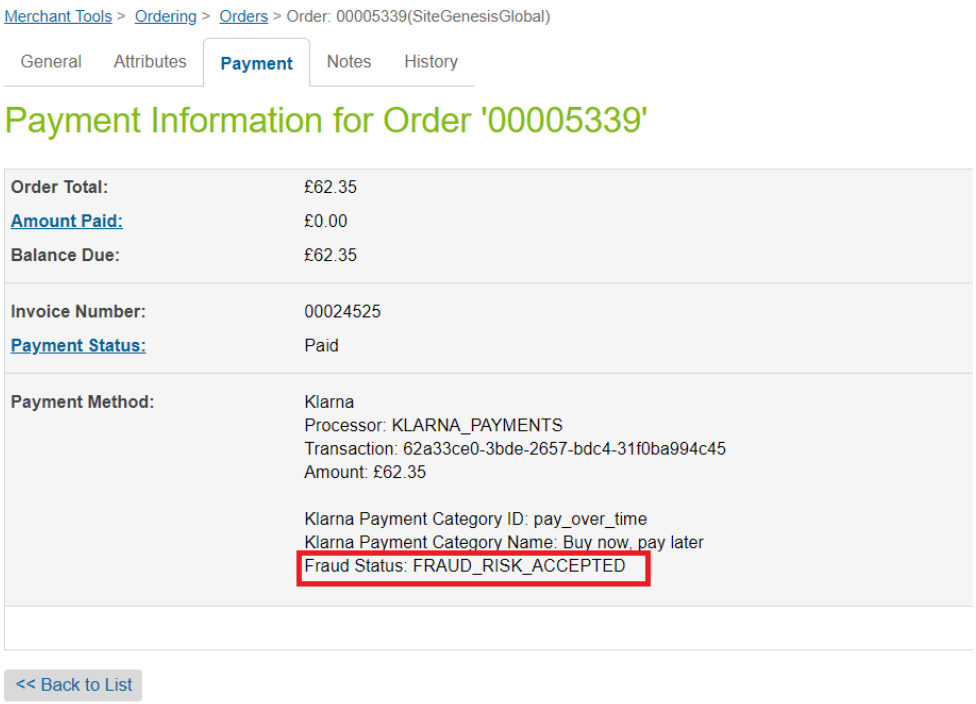


Figure 2 Klarna Payment Details in BM

2.2. Locales

The cartridge supports most locales including:

- English
- German
- Danish
- Spanish
- Finnish
- French
- Italian
- Dutch
- Polish
- Mexican

For a list of more supported locales, contact Klarna.

2.3. Use Cases

2.3.1. Enable Klarna Payments Across International Sites (NA, EU, OC)

Klarna Payments on SG can be configured independently on each site by locale.

2.3.2. Multitude of Payment Options for Customers

On the checkout billing step, configured Klarna payment options are dynamically loaded based on customer's cart information and the payment method categories returned for the current Klarna session.

The screen below shows an example of the payment options displayed in the Checkout billing page:

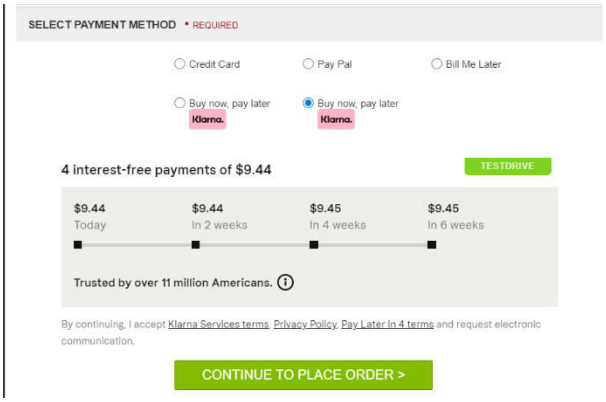


Figure 3 Payment Option

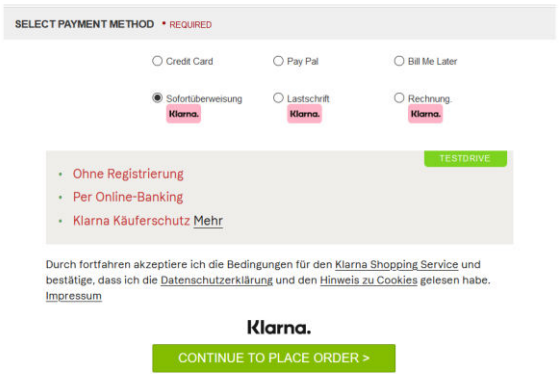


Figure 4 Payment Option

When customer selects (clicks) payment method, a widget with additional information of the Klarna product is displayed as shown above.

***Note**: The payment methods displayed are based on market availability and/or contractual agreement with Klarna.*

2.3.3. Authorize and Place Klarna Order in Checkout

Cartridge provides best practice implementation and options to include Extra Merchant Data (EMD) to optimize acceptance rate for Klarna Payment (methods) products. This includes customer info and (Buy Online, Pickup in Store) BOPIS details included as merchant data when custom site preference “attachments” is enabled for the site. The EMD data sent can be extended but should be reviewed case by case and optimized and validated based on merchant data & privacy requirements prior to go-live.

The customer’s information (personally identifiable information) is sent once customer chooses a Klarna payment method and authorizes (clicks: “Continue to Place Order”). The required customer information facilitates assessment and verification of customer data to display payments method options available for the customer.

When customer clicks “Continue to Place Order” button, the authorization is initiated, and a successful authorization takes customer to summary page. Note that customer may be prompted for additional information prior to completion of authorization based on the payment method selected and market. If the authorization is not successful (“approved = false”), then the customer will stay on the billing page, the “Continue to Place Order” button will be disabled and depending on the Business Manager settings the payment option may be hidden or grayed out (refer to section **2.3.4 Refusal of Klarna Payments on Payment Method – Authorization**).

In some cases, the Pay Now payment category requires additional “finalize” call to be triggered when the customer clicks on the “Place Order” button in the review page. This is needed to ensure that the funds will be transferred (e.g.: Bank Transfer payment methods which initiate fund transfer) only when the customer has decided to place the order on the merchant site.

Please, note that in such cases cookie called “**selectedKlarnaPaymentCategory**” is set in the customer’s browser. This cookie contains the selected payment method from the previous step and is used to initialize the “finalize” call to Klarna.

For more information regarding the authorize & finalize calls, refer to the developer documentation here

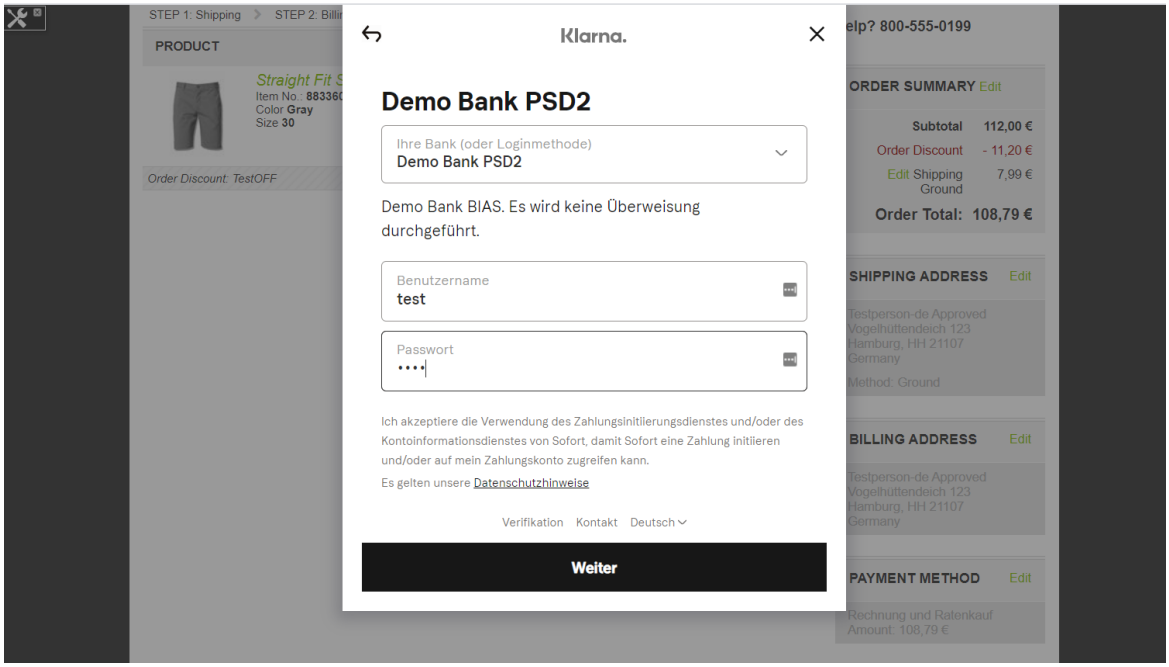


Figure 5 Finalize call screen

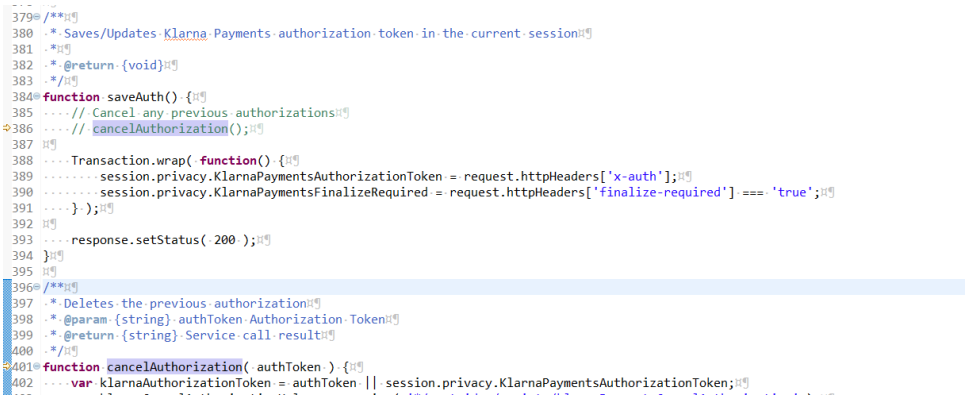
The Klarna Order is only placed prior to final step before SFCC order is created.

Successful payment authorizations (Klarna Payments authorization status: APPROVED) followed by placement of Klarna and SCC orders leads to a “Paid” order payment status and “Ready for Export” export status. Refused and pending Klarna Payment orders (Klarna Payments order statuses REJECTED and PENDING) lead to a “Not Paid” order payment status, and “Not Exported” export status.

In case of VCN settlement errors, Klarna orders are cancelled and SFCC orders are failed and set to a “Not Exported” export status.

When customers have selected Klarna payment option as a payment method for the order, successfully authorized the amount on order and later return to the billing page to choose a different payment method that is non-Klarna payment method. In such scenarios, when customer authorizes (non-Klarna payment method) and reaches the review page – automatic “**cancelAuthorization**” call will be triggered to release the authorized funds (Klarna related) & free up the available purchase amount for this customer.

Merchants have the option to utilize this function to cancel prior authorization when required for specific use-cases apart from the above scenario. If enabled, the checkout flow should be tested thoroughly as part of integration, considering the valid checkout scenarios (e.g., relevant Klarna session flow, Klarna payment method switching, order amount updates, checkout with external payment method, etc.).



```
/**
 * Saves/Updates Klarna Payments authorization token in the current session
 *
 * @return {void}
 */
function saveAuth() {
    // Cancel any previous authorizations
    // cancelAuthorization();

    Transaction.wrap( function() {
        session.privacy.KlarnaPaymentsAuthorizationToken = request.httpHeaders['x-auth'];
        session.privacy.KlarnaPaymentsFinalizeRequired = request.httpHeaders['finalize-required'] === 'true';
    } );

    response.setStatus( 200 );
}

/**
 * Deletes the previous authorization
 * @param {string} authToken Authorization Token
 * @return {string} Service call result
 */
function cancelAuthorization( authToken ) {
    var klarnaAuthorizationToken = authToken || session.privacy.KlarnaPaymentsAuthorizationToken;
```

Figure 6 Cancel Authorization Call

2.3.4. Refusal of Klarna Payments on Payment Method – Authorization

Upon selecting one of Klarna's options as the payment method on billing step of checkout and based on the customer information provided, Klarna Payments can be refused as a payment method.

If the payment method was rejected with "**show_form=false**" & "**approved=false**" (i.e., hard reject), the merchant has the option to choose what happens with the payment option display in Billing page using BM preference "Hide Payment Methods on Deny" (**kpRejectedMethodDisplay**):

No – Leave the payment visible to the customers

Hide – The payment option will be hidden from customer

Grey Out – The payment option will be greyed out and not clickable

Please note that reloading the page will show the denied Klarna payment method again

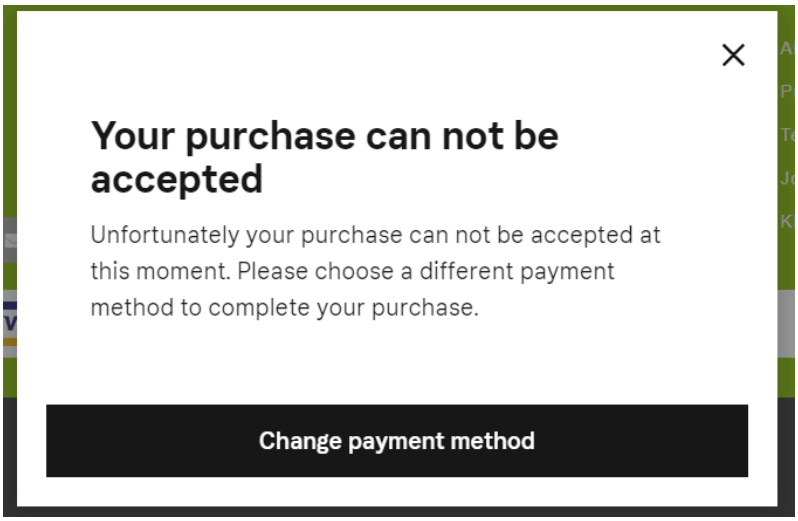


Figure 7 Denied Order Popup

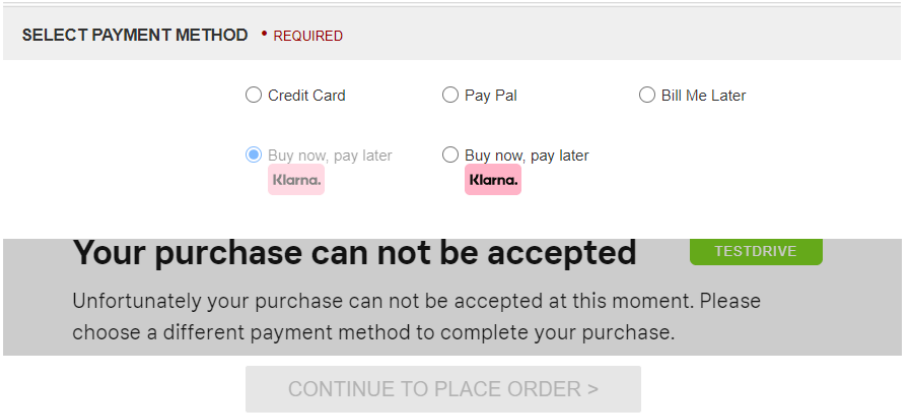


Figure 8 Greyed Out Payment Option

2.3.5. Klarna Payment Option Not Available for Current Purchase - Billing Page

The customer is presented with an appropriate message in the Klarna widget when customer attempts to choose Klarna in the billing page.

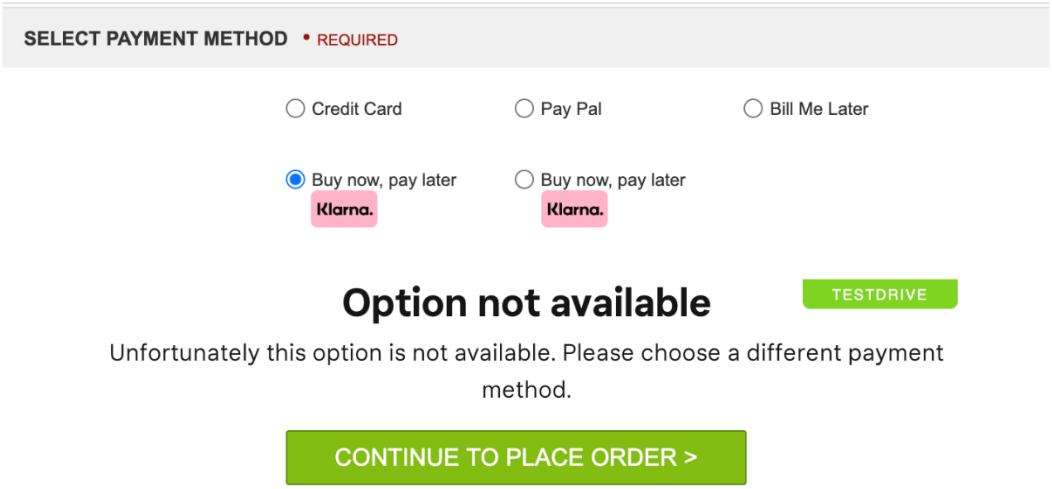


Figure 9 Option Not Available

2.3.6. Klarna Payments Not Available - Checkout

If Klarna API is not available or the site/storefront is not applicable, Klarna is not presented as a payment option in the billing page. It is recommended that Klarna session must not be created when customer chooses a non-Klarna market or merchant store, as well as in cases when merchants have multicurrency storefront with basket currency that is not supported by Karna.

2.3.7. Handling Notifications

In scenarios were the Klarna Order has been created but instead of immediately accepting the order, the Klarna order is flagged for additional review. This results in Commerce Cloud order staying in "Created" status with Fraud Status: PENDING. This order is marked with EXPORT_STATUS_NOTEXPORTED, confirmation status NOTCONFIRMED and NOTPAID.

For Klarna orders with Fraud Status PENDING, once review is complete, updates are sent to the pre-configured notification_url on the merchant Commerce Cloud site. The updated Fraud status depends on the fraud screening, the returned Fraud Status (e.g., FRAUD_RISK_ACCEPTED) is displayed in BM. The push notification is repeatedly sent (up-to 24 hours, every 10 mins) until the POST request is acknowledged with a 200 response.

Klarna sends one of the following event types in the notification to SFCC to update risk status: FRAUD_RISK_ACCEPTED, FRAUD_RISK_REJECTED, FRAUD_RISK_STOPPED.

The notification updates are generally received within 4-24 hours. The order's payment transaction is updated (see **kpFraudStatus**). This can be seen in BM on the order details Payment tab as below:

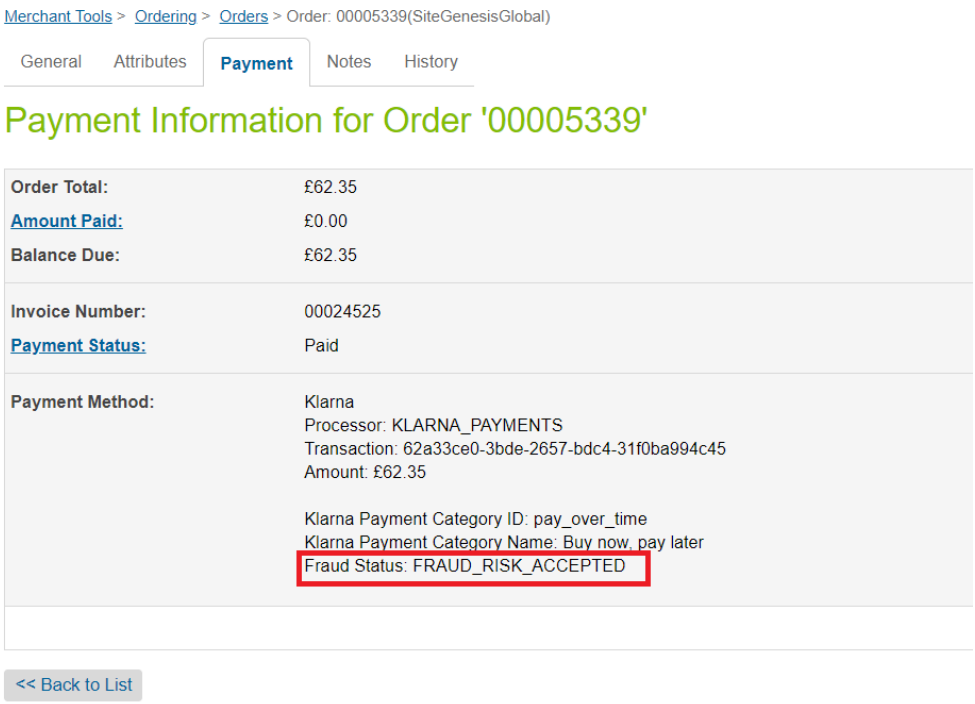


Figure 10 Fraud Status in Payment Details

If the order is "FRAUD_RISK_ACCEPTED" upon notification, the order will be placed in SFCC (order status changes to OPEN). The order is marked with status CONFIRMATION_STATUS_CONFIRMED and export status EXPORT_STATUS_READY. If the order was placed with auto-capture, the payment status will be set to PAYMENT_STATUS_PAID and the full order amount will be captured.

If the order is "FRAUD_RISK_REJECTED" or "FRAUD_RISK_STOPPED" upon notification, the order is failed (FAIL) in SFCC.

***Note**: The Klarna pending functionality availability is dependent on markets and enabled based on the contractual agreement with Klarna.*

2.3.8. Virtual Cards Settlements

This option is disabled by default. However, if standard order management is not a reasonable option for a Klarna integration, then Klarna's Merchant Card Service API based virtual card solution may be utilized via site preference "**kpVCNEnabled**":

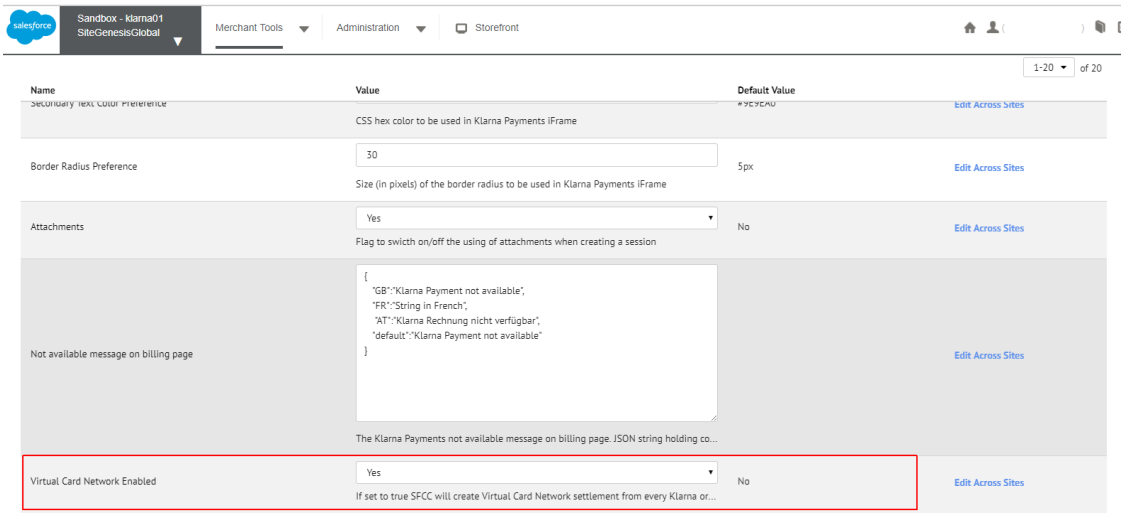


Figure 11 VCN Enablement Setting

The virtual card (see note) is issued against an Klarna order, for the purpose of capturing the Klarna authorized order amount using the standard card rails.

When a customer places an order, the order is first booked in SFCC. Once an order has been accepted by Klarna, the cartridge integration creates a virtual card based settlement, utilizing the merchant card services (MCSv3) API.

Once a settlement has been created (virtual card returned), the encrypted card details are saved in SFCC and can be later utilized by the merchant OMS platform or custom PSP integration to authorize the virtual card until the Klarna order is valid. Then, once the order has been fulfilled, the card funds should be captured. (For delays in capture, or other special use cases, please speak with the Klarna Key Account Manager in advance). While Klarna is the original payment method of the order, the order amount will be settled with the merchant using the issued virtual credit card instead of direct bank account transfer.

Please review the procedures with Klarna team for details of settlement process using virtual cards.

Refer to the below code in "**controllers/KlarnaPayments.js**" and update accordingly to the integrated cards processor.

***Note**: If the Klarna 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 "FRAUD_RISK_ACCEPTED".*

The virtual card issued is limited to 1 single successful authorization per order for a given MID.

For decrypting the credit card details refer to section **9.3 Decrypt VCN Card Details.**

```
Transaction.wrap( function() {
    paymentInstrument.paymentTransaction.transactionID = session.privacy.KlarnaPaymentsOrderID;
    paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;
    session.privacy.OrderNo = orderNo;
    args.Order.custom.kpOrderID = session.privacy.KlarnaPaymentsOrderID;
    args.Order.custom.kpIsVCN = empty( vcnEnabled ) ? false : vcnEnabled;
} );

if ( session.privacy.KlarnaPaymentsFraudStatus === 'PENDING' ) {
    return { authorized: true };
}

if ( vcnEnabled ) {
    var isSettlementCreated = _handleVCNSettlement( args.Order, session.privacy.KlarnaPaymentsOrderID, localeObject ); // esl
    if ( isSettlementCreated ) {
        //Plug here your Credit Card Processor
        return require( '*/cartridge/scripts/payment/processor/BASIC_CREDIT' ).Authorize( {'OrderNo':args.Order.getOrderNo(),
    }
    var klarnaPaymentsCancelOrderHelper = require( '*/cartridge/scripts/order/klarnaPaymentsCancelOrder' );
    klarnaPaymentsCancelOrderHelper.cancelOrder( localeObject, args.Order );
    return { error: true };
}

return { authorized: true };
}
```

Figure 12 Credit Card Authorization Call

To utilize virtual card integration option the merchant should:

- Enable VCN option in Site Preferences as shown above
- Enter the VCN Public Key ID. Unique UUIDv4 value, which should be different for playground testing and Production (live site)

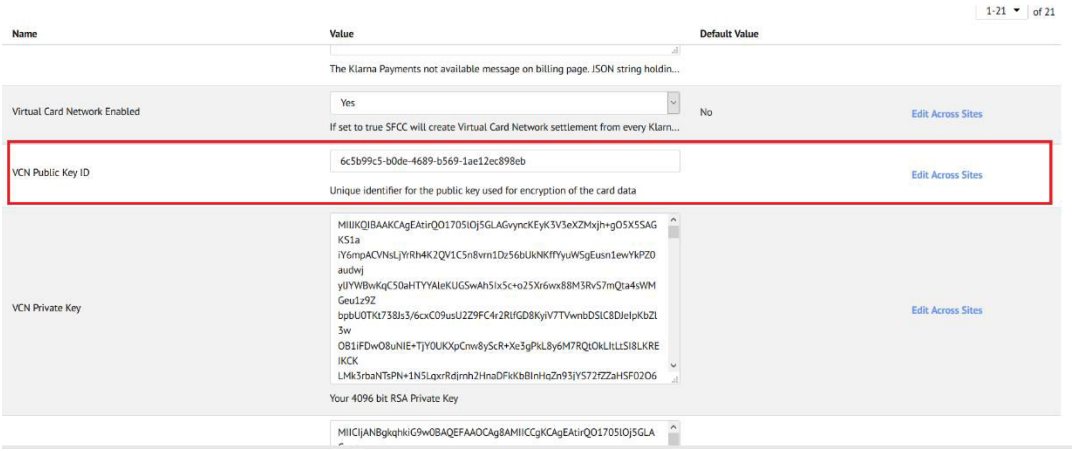


Figure 13 VCN Public Key ID

- Generate a 4096-bit RSA key pair (Refer to section **Generate Key Pair and Key Id for Virtual Card Settlements**). 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)

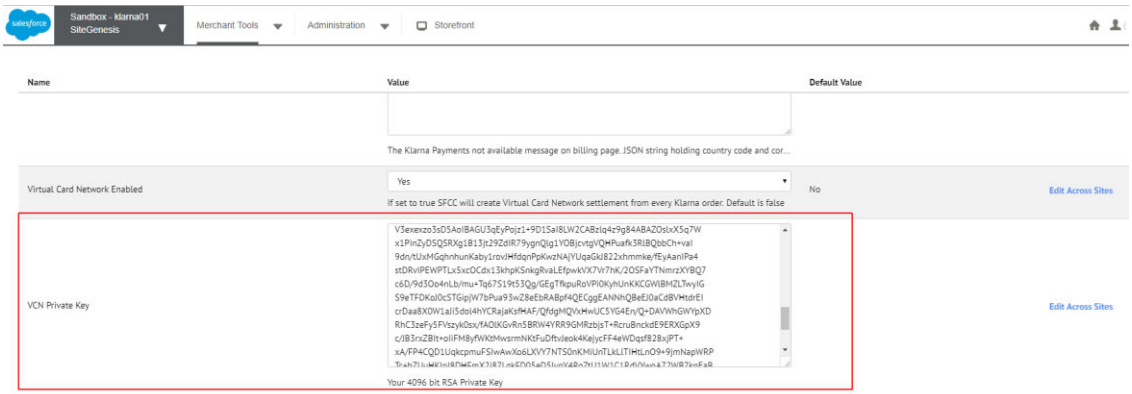


Figure 14 VCN Public & Private Keys

- Update the VCN settlement retry setting "**kpVCNRetryEnabled**". By default, this is disabled. However, if enabled and in cases when a VCN creation error is returned, the application will retry the settlement request once again with order_id as the idempotency key.

- Finally, you need to send the generated unique key_id + public key combination in JWK format to Klarna prior to testing and go-live. It will be used to encrypt the aes key which is used for encrypting the pci data on Klarna side when settlement request is made. After confirmation from Klarna that the key has been successfully added to your merchant profile you would be able to use virtual card-based settlement option for Klarna payment methods

If enabled and fully configured, virtual card settlement request is made successfully. For orders placed with the VCN settlement option, the related custom attributes are shown below:

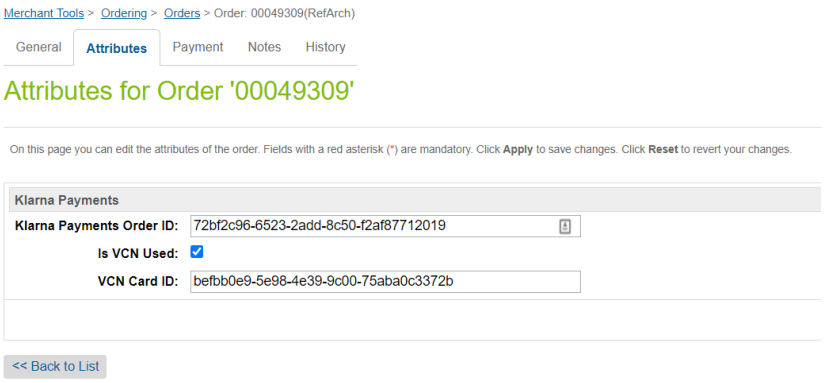


Figure 15 VCN Details in Order

If required, the additional virtual card details can be assigned to this group in Administration > Site Development > System Object Types > select “Order”. In the Attribute Grouping tab select Klarna_Payments and click on “edit”. Assign the new attributes and save the data.

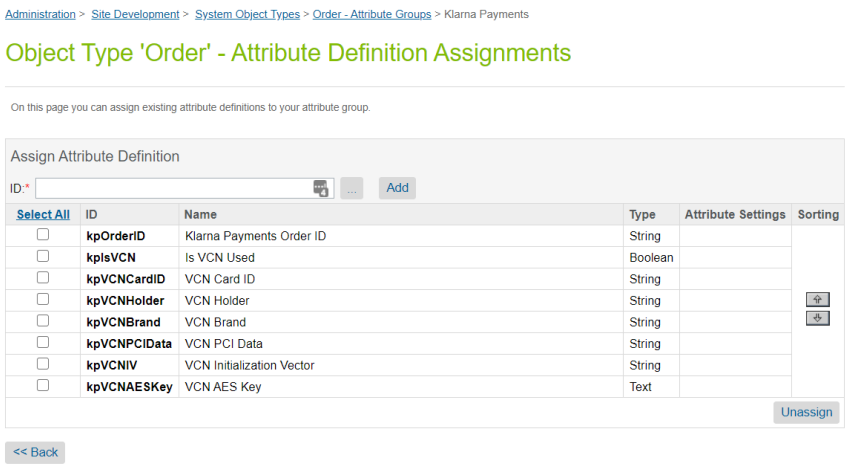


Figure 16 Full List of VCN Attributes

Please work with Klarna Account Manager and Delivery contact in advance to select the appropriate virtual card product based on your business requirements and use-cases. You can find information here around other use cases supported.

Important Note!

*DO NOT SAVE DECRYPTED PCI DATA ON THE SERVER. It is the responsibility of the merchant to ensure PCI-DSS compliance and to ensure the card data is handled securely in co-ordination with required partners/Payment Service Provider/Acquirer. Please review in advance the order export details required for virtual card-based Klarna orders. Any historical decrypted PCI data should also be expunged, regardless of the validity date (see section **3.4.1 Job "OrderCleanUp"**).*

2.3.9. Auto-Capture

Auto-capture is enabled via a site preference "**kpAutoCapture**" located in "**Klarna_Payments**" preference group.

When the preference is enabled (disabled by default), a full amount capture is attempted. If the capture is successful, the SFCC order's payment transaction is marked as Paid, and viewable in the Business Manager.

The order will be marked as "*Captured*" in the Klarna's Merchant Portal.

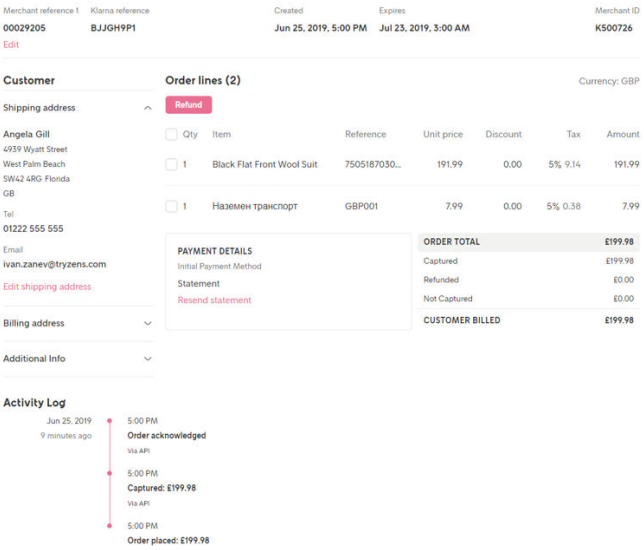


Figure 17 Order Details in Klarna Portal

If the capture is unsuccessful, an error will be logged in the custom error log. The setting must be reviewed with Klarna delivery team before testing and go-live.

***Note**: Auto-capture is possible for orders when VCN is not enabled!*

2.3.10. Widget Customizations

Note: The merchant will need a configured Klarna Payments account.

The merchant can style the Klarna Payments widget (skin), to match the marketing and branding needs of their store. The list with the graphic elements that can be customized out of the box through site preferences are listed below:

"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"

2.3.11. Customizing Payment Method Name

The payment method name "*Klarna Payments*" may be customized via the "*Merchant Tools > Ordering > Payment Methods*" section in Business Manager.

The screenshot below shows the "*Klarna*" method selected and the administrator choosing a language from the drop-down.

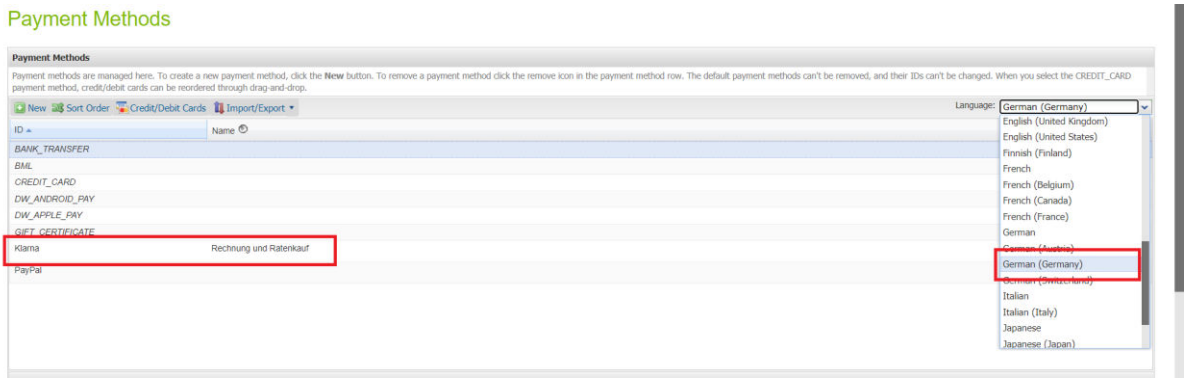


Figure 18 Customize Payment Name

The payment method name would then be visible in the mini summary & confirmation screens, the confirmation emails and My Account Order Details section.

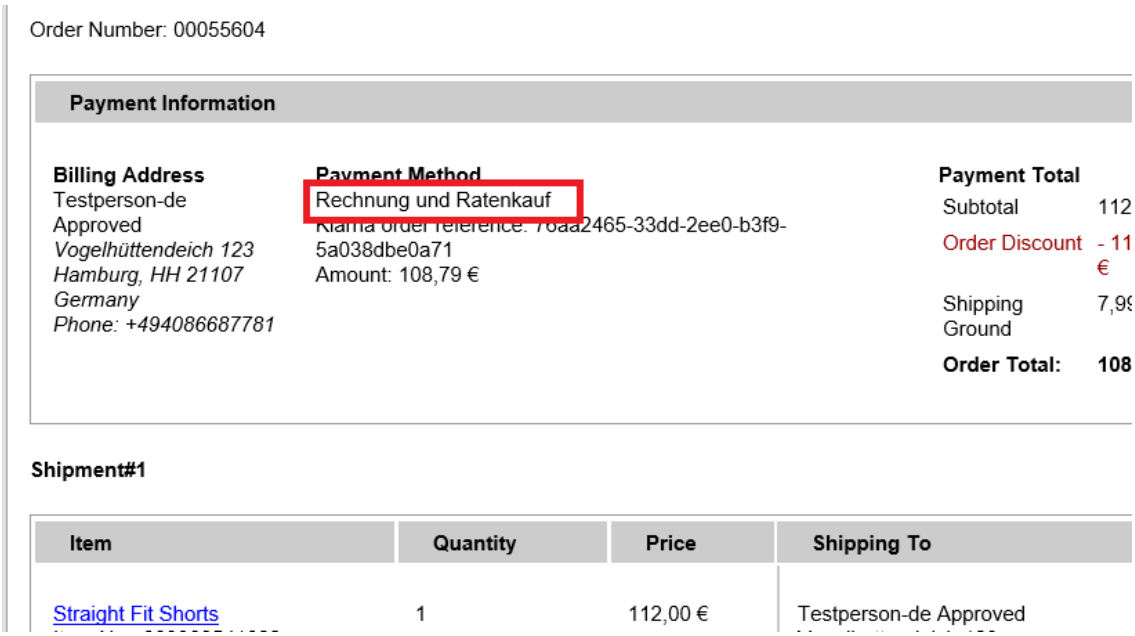


Figure 19 Payment Method Name in Email

2.3.12. Klarna On-Site Messaging

On-site messaging is a platform that enables you to add tailored messaging to your website. With On-site messaging you can let shoppers know about the different payment options you have available as they browse your site. By using Klarna, customers have access to our flexible payment options in the checkout; On-site messaging is a great way to let them know even before they decide to buy.

The Klarna Payment cartridge provides multiple options in the standard implementation based on the reference architecture:

- Product Page and Cart based promotions
- Sitewide
    - Top Banner strip
    - Footer logo
- Custom Info-page

Klarna On-Site Messaging (OSM) is configured by site and by locale via the **KlarnaCountries** custom object.

To configure the OSM settings for a locale, you must visit "***Merchant Tools – Custom Object Editor***" and search for **KlarnaCountries** custom object. Select the country Key, e.g. "US"

In the custom country specific configuration, provide a valid locale for the OSM tag based on the country being configured.

The OSM Data Client ID and Data keys required are available in Klarna Merchant Portal within the On-site Messaging App:

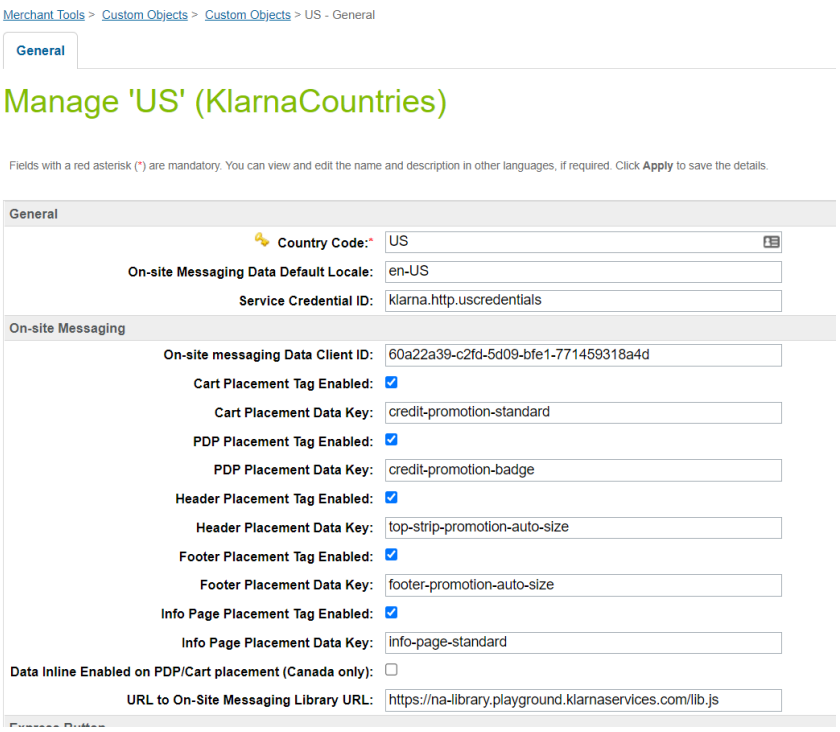


Figure 20 OSM Settings in Business Manager

To enable Placement tag for the Cart Page, the “Cart Placement Data Key” must be filled with Data Key value and the “Cart Placement Tag Enabled” must be checked.

To enable Placement tag for the PDP Page, the “PDP *Placement Data Key*” must be filled with Data Key value and the “*PDP Placement Tag Enabled*” must be checked.

To enable Placement tag for header, the “Header Placement Data Key” must be filled with Data Key value and the “Header Placement Tag Enabled” must be checked.

To enable Placement tag for footer, the “Footer Placement Data Key” must be filled with Data Key value and the “Footer Placement Tag Enabled” must be checked.

To enable Placement tag for the Info Page, the “Info Page Placement Data Key” must be filled with Data Key value and the “Info Page Placement Tag Enabled” must be checked.

In Library URL, please input the full URL to the On-Site Messaging JavaScript Library.

In On-site messaging Data Client ID, please input the On-Site Messaging Data client ID value.

On-site messaging Data locale, please input the valid On-Site Messaging Data locale.

For Canada only, please update “Data Inline Enabled on PDP/Cart placement (Canada only)” value as follows:

- For PayBright enabled payment methods – set to true.
- For Klarna enabled payment methods – set to false.

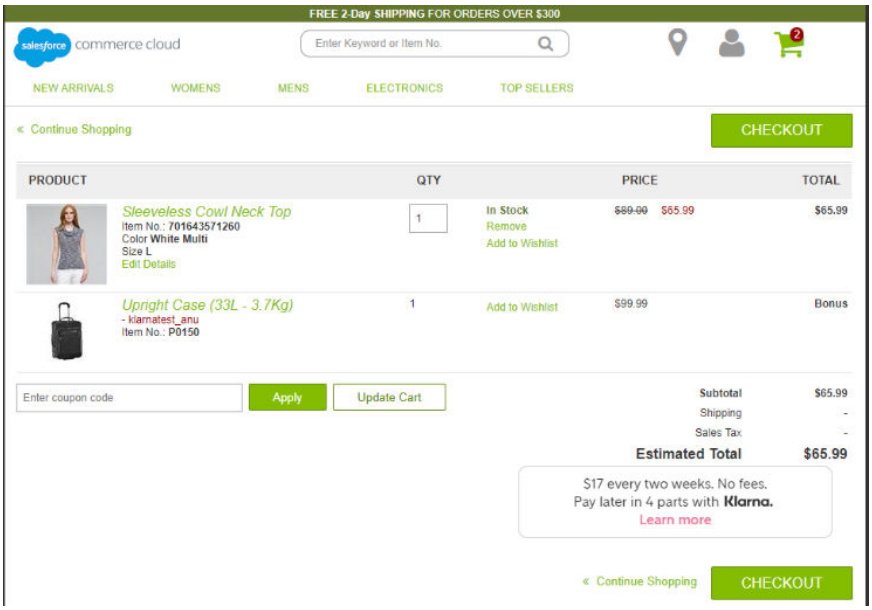


Figure 21 On-Site Messaging Enabled on Cart Page

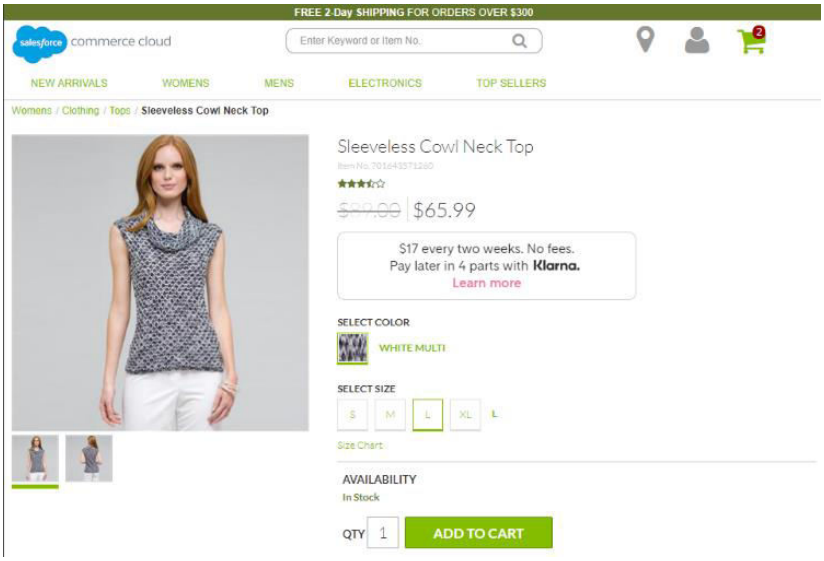


Figure 22 On-Site Messaging Enabled on PDP Page

In addition to the above, if you wish to display the dedicated (custom) Klarna info OSM page you can use the following controller endpoint "**KlarnaPayments-InfoPage**". For example, you should update the "**footer-about**" content asset to include this line of code as shown on the screenshot.

```
<li><a href="$url('KlarnaPayments-InfoPage')$" title="Go to Klarna
Info">Klarna Info</a></li>
```

Figure 23 Footer Asset Update

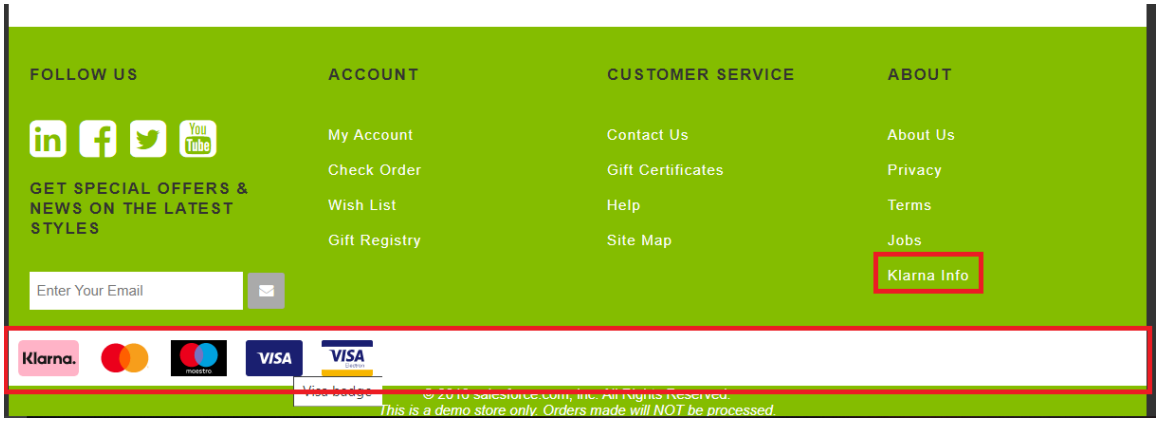


Figure 24 On-Site Messaging Enabled on Footer and link to Klarna Info Page

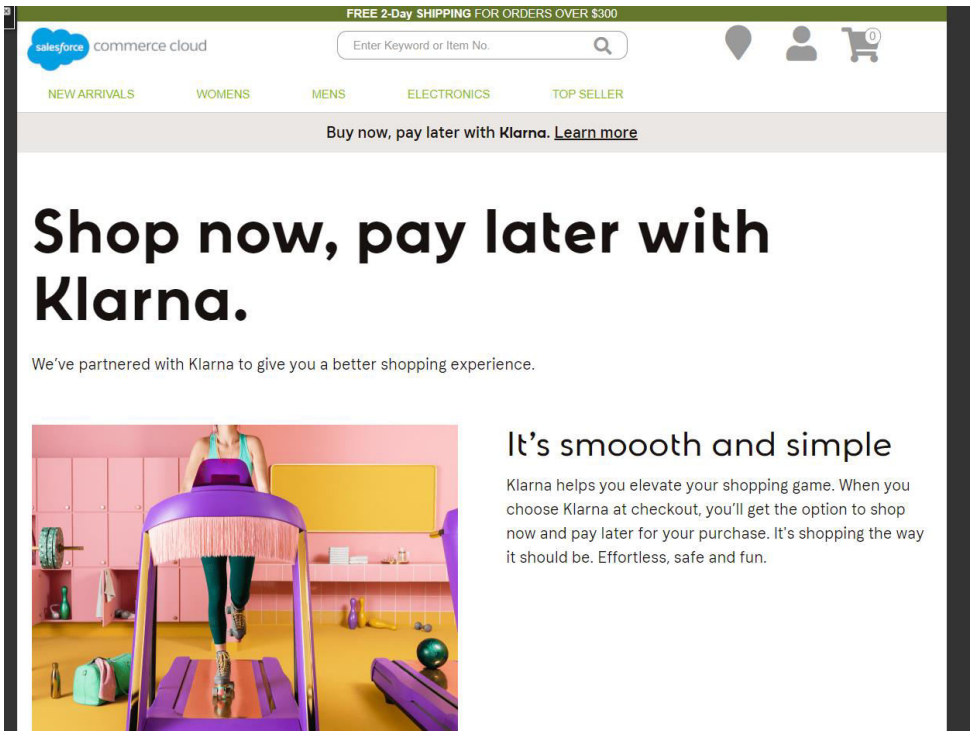


Figure 25 On-Site Messaging Enabled on Header & dedicated Klarna Info Page

For more information regarding OSM customizations and best practices, please refer to the Klarna Developer / Docs

Integration Best practice + information about Klarna Branding and Co-marketing options here.

***Note**: It is the merchant's responsibility to ensure that user content is collected when required for OSM placements for your local market to abide by the legal requirement. e.g., EU cookie-guide in Klarna Merchant Portal*

2.3.13. Klarna Express Button

The cartridge supports the **Klarna Express button (KEB)** on the standard Cart page and mini-Cart.

By enabling Klarna's Express button on the cart page of your website, shoppers can choose to log into their (*or sign up for a*) Klarna account and have their personal details pre-filled in the checkout. Klarna's payment method will be pre-selected for the shopper. In supported markets, Klarna's network of shoppers benefit from a pre-filled checkout experience, which also includes first time shoppers on a merchant storefront.

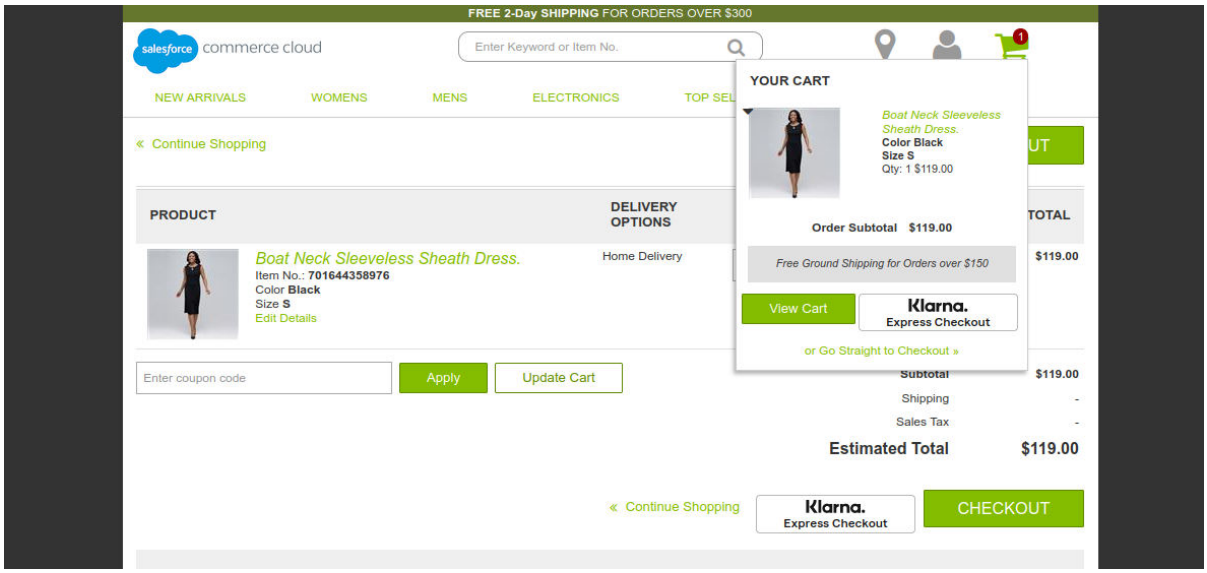


Figure 26 Klarna Express Button on Cart Page

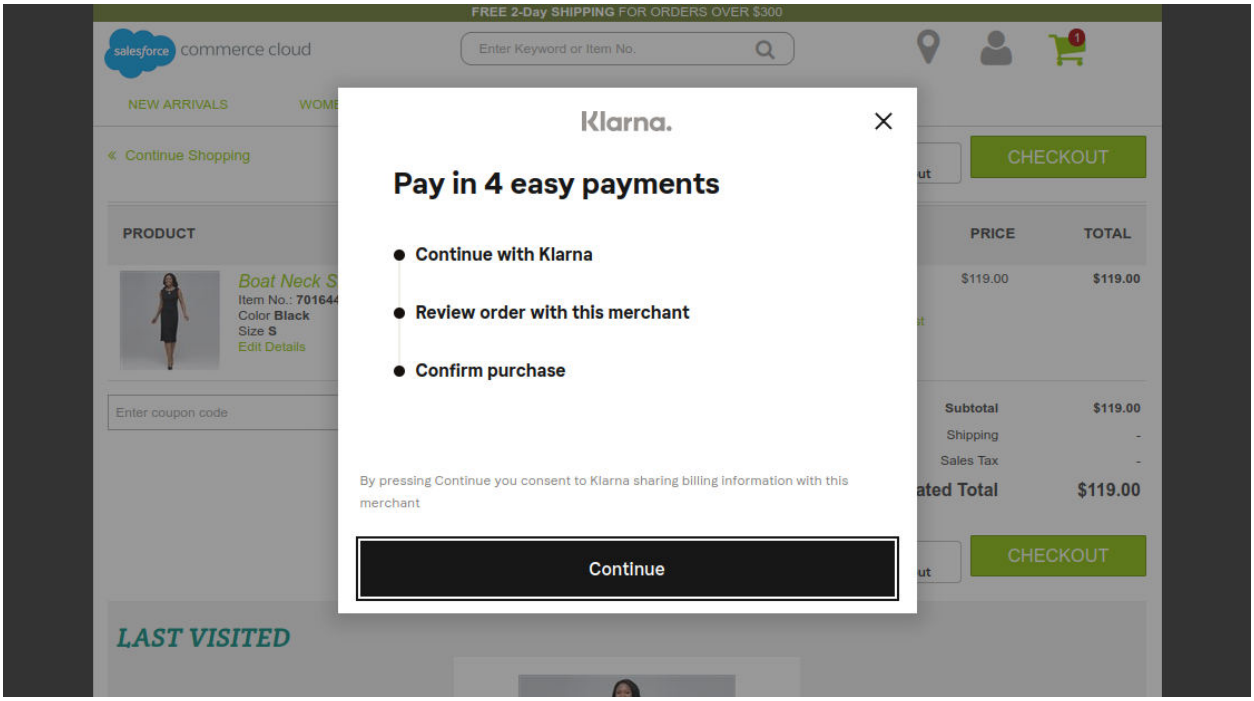


Figure 27 Klarna Express Button clicked Cart Page

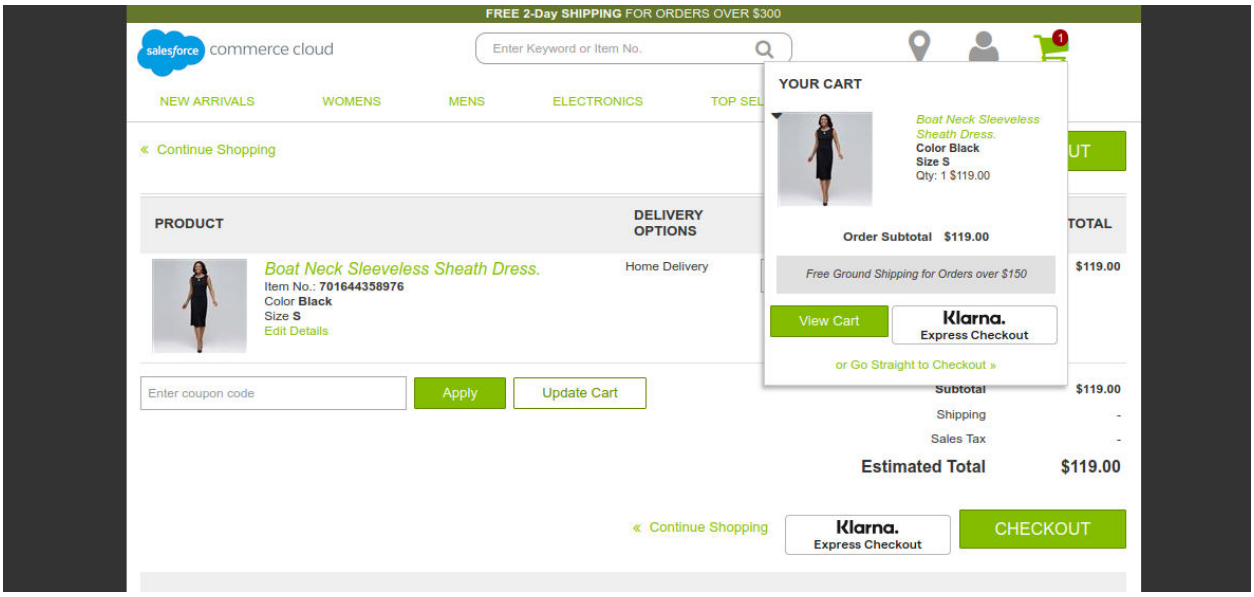


Figure 28 Klarna Express Button on Cart Page

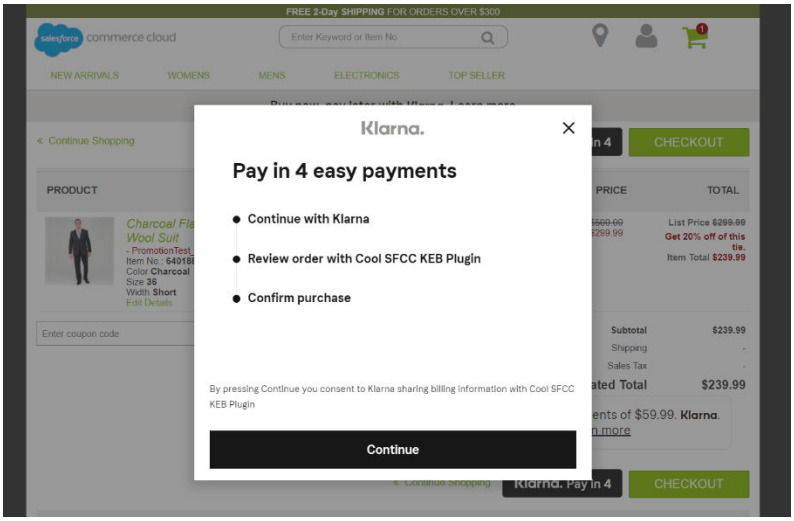


Figure 29 Klarna Express Button clicked Cart Page

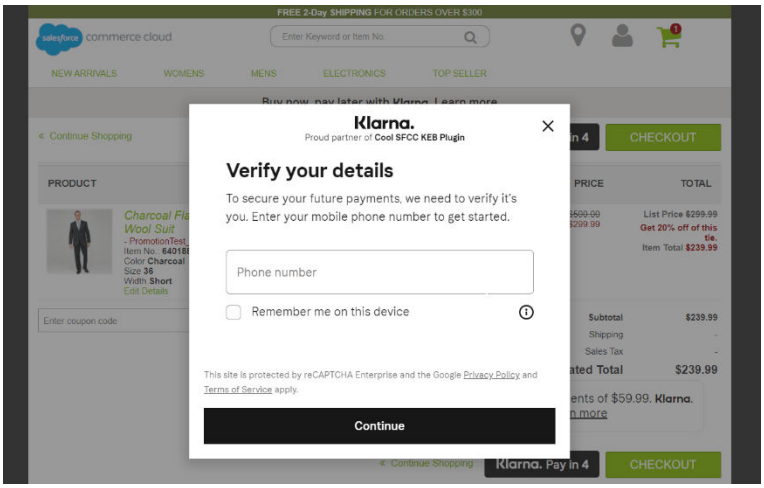


Figure 30 Klarna Express Button user credentials input actions

Once the credentials have been provided and the shopper has successfully authenticated with their Klarna account, they are redirected to the checkout page with relevant details pre-populated and Klarna Payment (e.g.: “Pay in 4”) method pre-selected on the billing page.

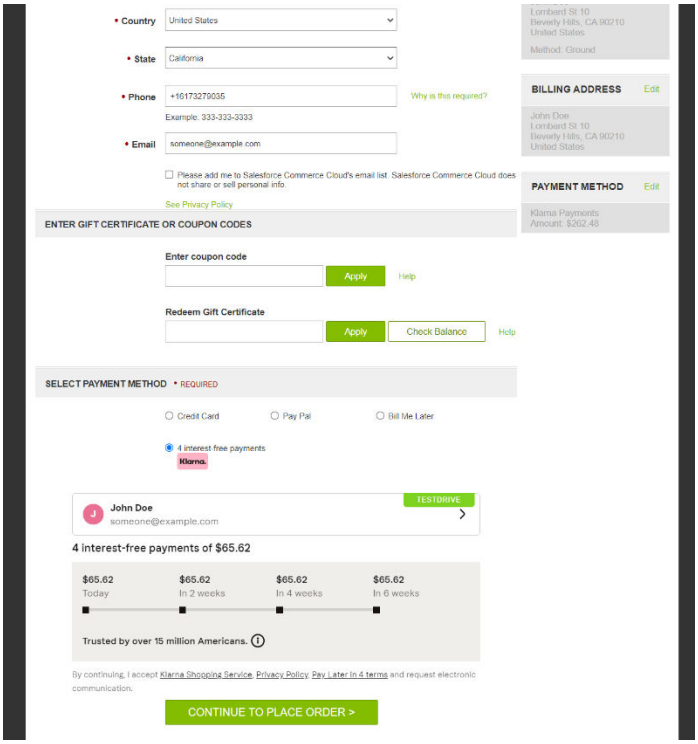


Figure 31 Checkout details prefilled and Klarna Pay in 4 selected

For custom checkout design and when address field validations are in place, please review the changes to ensure the shopper details are populated on the checkout forms as expected and to ensure the Klarna Payment method authorization is completed successfully.

***Note**: On re-direct, in the standard checkout, any existing email ID and phone number are updated with the latest provided by the shopper*

2.3.14. Klarna Payment Method Based Promotions

As of B2C 20.7 release, merchants can include payment methods as qualifier for product, order & shipping promotions.

OOTB when such promotion is set up to use a payment method as qualifier, the total order amount will be visible to the customer once they reach the review page. This means that the Klarna authorization call will be made for larger amount than the final one.

To address this issue, once the customer clicks on any of the payment options in the billing section – a call will be made to the backend. This will re-calculate the basket totals if any promotions are applicable and will update the Klarna session details. As a result, the Klarna iframe widgets and the mini summary section on the storefront will update to show the final order details.

***Note**: When the selected payment method is non-Klarna one, this logic should be customized by the merchant to handle any 3rd party payment integrations.*

2.3.15. Price Adjustment Taxation Handling

OOTB the Klarna API calls will send the product / shipping method details and the relevant discounts as separate lines items as shown below:

```
"order_lines": [
    {
        "type": "discount",
        "name": "5 Off Ties Promotion",
        "reference": "682875540326M_$5_off_ties_promotion",
        "quantity": 1,
        "merchant_data": "5ties",
        "unit_price": -500,
        "tax_rate": 500,
        "total_amount": -500,
        "total_tax_amount": 0,
        "total_discount_amount": 0,
        "product_url": null,
        "image_url": null
    },
    {
        "type": "physical",
        "name": "Checked Silk Tie",
        "reference": "682875540326M",
        "quantity": 1,
        "merchant_data": "",
        "unit_price": 1919,
        "tax_rate": 500,
        "total_amount": 1919,
        "total_tax_amount": 68,
        "total_discount_amount": 0,
```


Figure 32 Line Items with Default Taxation Setting

In some cases, merchants using gross taxation might enable the “Tax Products and Shipping Only Based on Adjusted Price” preference under “Merchant Tools > Site Preferences > Pricing and Promotion” where the price adjustments are not taxed.

The setting, “**kpPromoTaxation**” has been introduced, where merchants should update this to match the promotion setting below:

- price (Based on Price) – The product, shipping and their discounts will be sent as separate lines. This is the default setting.
- adjustment (Based on Adjusted Price) – When this is selected, the product or shipping method line item will be sent with attribute “total_amount” matching the prorated price and attribute “total_discount_amount” – matching the total sum of all discounts for this item.

```
"order_lines": [
    {
        "type": "physical",
        "name": "Checked Silk Tie",
        "reference": "682875540326M",
        "quantity": 1,
        "merchant_data": "",
        "unit_price": 1919,
        "tax_rate": 2200,
        "total_amount": 1419,
        "total_tax_amount": 256,
        "total_discount_amount": 500,
```


Figure 33 Line Items with "Based on Adj." Taxation Setting

***Note**: Enabling this setting is not required for storefronts with net taxation as the tax is not included in the products base price. The total order sales tax is sent as a separate line item to Klarna and not on product/shipping line-item level.*

2.3.16. Buy Online, Pickup in Store (BOPIS)

When store pickup has been enabled on the storefront, the integration will send stores details to Klarna in the authorization request and when placing the Klarna order. Store information is not sent prior the interaction of the customer with the Klarna payment method widgets.

The store address(es) is always included in the EMD attachment “other_delivery_address” when applicable.

The address included on the shipping address in the Klarna order with store pick-up, is as below:

- Orders that have 1 or more store pickup shipments (no home delivery address), the first store shipment details will be set as the shipping address
- Order with store pick-up(s) and home delivery shipment, home delivery address will be used as the shipping address in the Klarna calls
- If the order contains no store pickups, no information is sent in “**other_delivery_addresses**” attribute

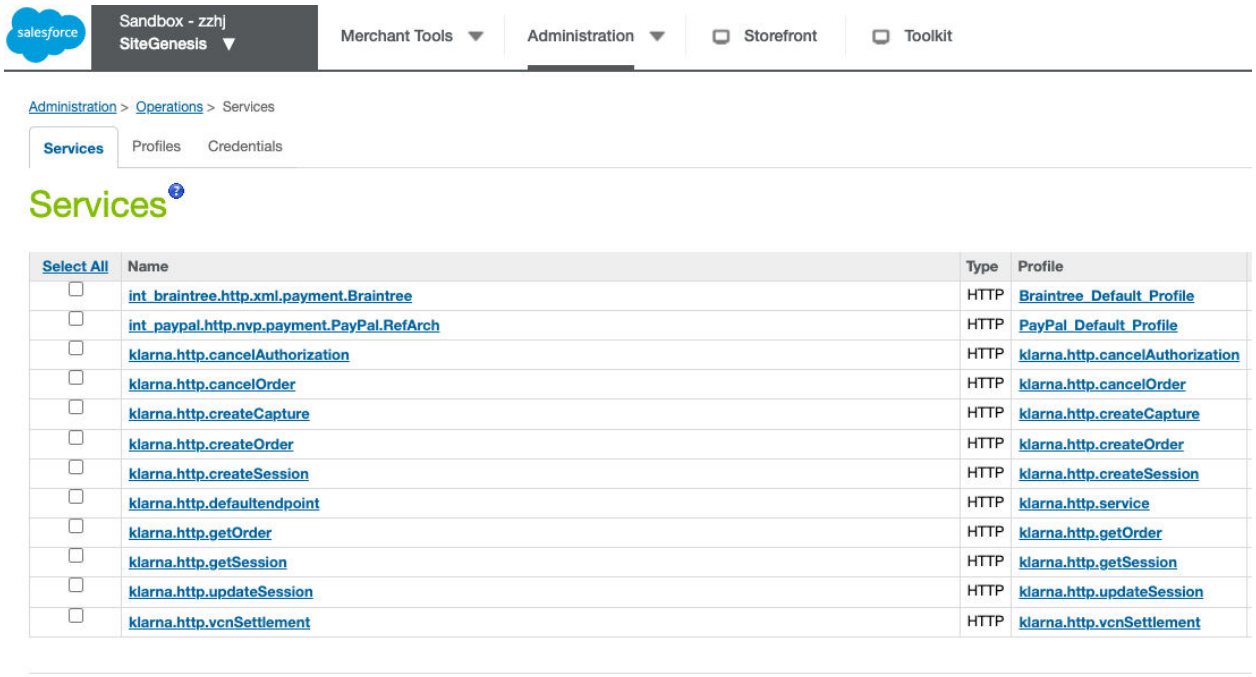
```
{
    "attachment": {
        "content_type": "application/vnd.klarna.internal.emd-v2+json",
        "body": {
            "other_delivery_address":[{
                "shipping_method": "store pick-up",
                "shipping_type": "normal",
                "first_name": "Test",
                "last_name": "Customer",
                "street_address": "1487 Bay St",
                "street_number": "",
                "postal_code": "01109",
                "city": "Springfield",
                "country": "US"
            }]
        }
    }
}
```

For more information on the options refer here

2.3.17. Configuration Support for Service Rate Limits

Klarna Payment API sets rate limits by operation (session creation, order creation, etc) to maintain a high quality of service for all its customers. A merchant has the flexibility to request higher rate limits for a specific duration. The duration of such events may last the period of high traffic events (e.g.: Flash sales or Holiday shopping). To enable these agreed rate limits, the service rate limits feature can be utilized.

The Service ID (e.g.: klarna.http.createSession) and Service Profile (e.g.: klarna.http.createSession) for default operations are included in the meta file.



Default Klarna Service Operations:

- Create Session
- Update Session
- Create Order
- Cancel Order
- Capture Order (Full Capture)
- New Order Settlement (Virtual Card)

For more information on the options refer here

2.3.18. Klarna Subscriptions

The cartridge supports subscription handling.

2.3.18.1. Configuration

Subscription details are configured on product level. Products can be subscription, standard or both. The trial period should be an integer value.

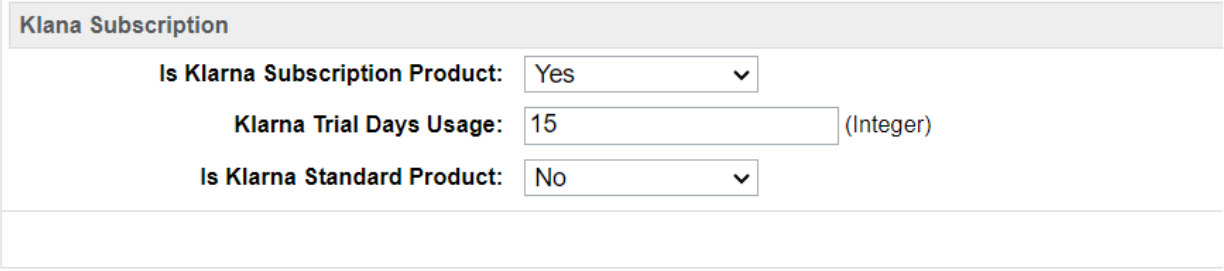


Figure 34 Product subscription configuration

2.3.18.2. Cart page

- Subscription products

Subscription-only products are automatically added to the shopping cart as subscription line items. For products that can be both subscription and standard, users can select their preference on the cart page. It's important to note that the checkout process won't proceed if the cart contains a mix of standard and subscription products with different trial periods, or if some products have a trial period while others do not.

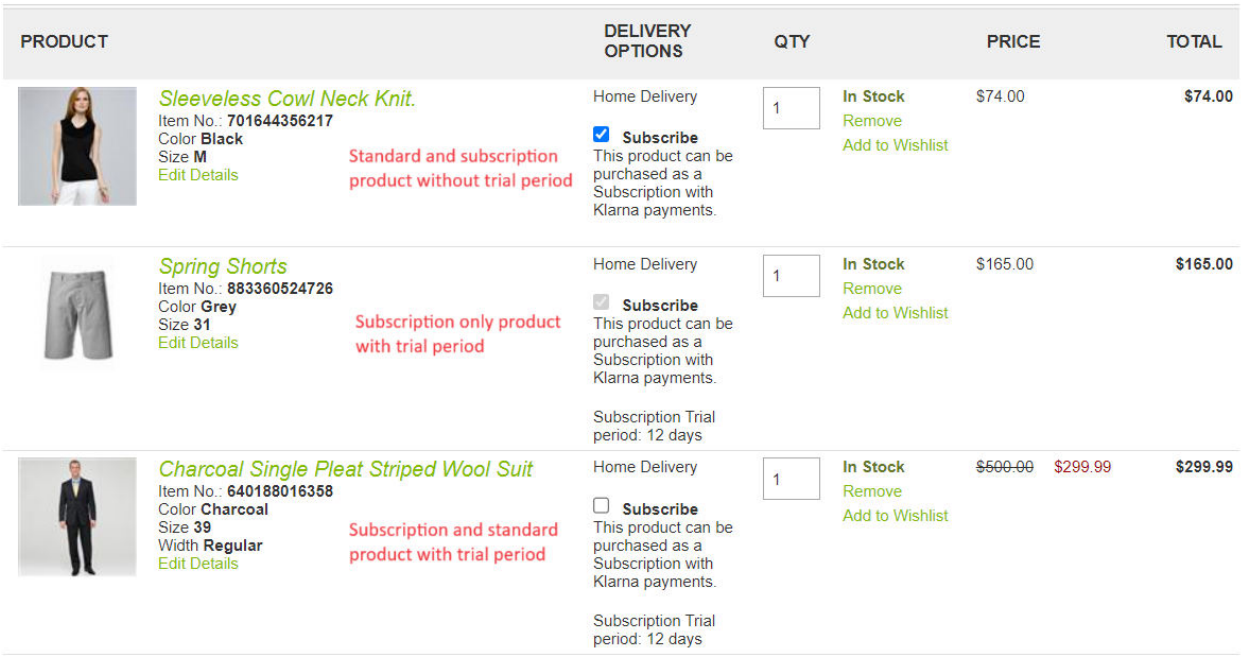


Figure 35 Subscription products on cart page

- Subscription details

Dropdown menus with predefined values appear on the cart page when there's at least one subscription product in the cart. The configuration of these values can be managed in the Administration panel under "Site Development" > "System Object Types" > "Basket - Attribute Definitions." The attributes available for configuration are:

- kpSubscriptionFrequency
- kpSubscriptionPeriod

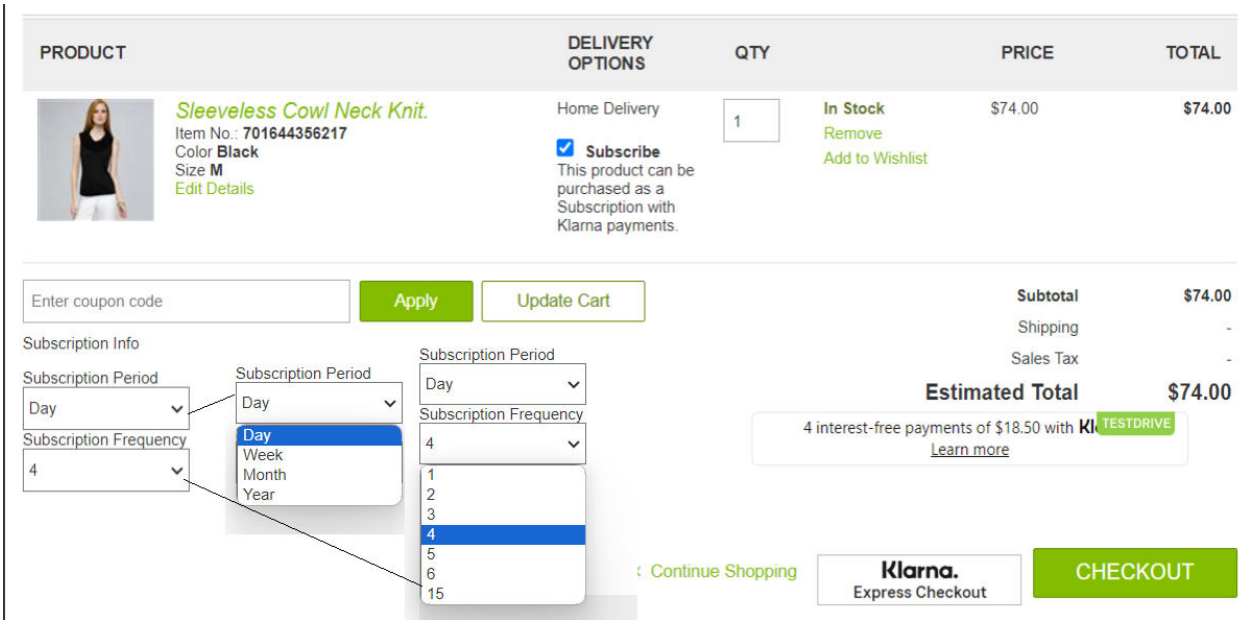


Figure 37 Subscriptions details in cart page

2.3.18.3. Checkout

Only Logged in user is allowed to do a subscription check out. Session intent is defined based on the basket content:

- "tokenize" – basket contains products with trial period, user is not charged on order creation
- "buy_and_tokenize"– subscription products without trial period
- "buy"– only standard products, no subscription products

For intents "tokenize" and "buy_and_tokenize" Klarna customer token for recurring payments is created. This is stored in customer profile for future usage.

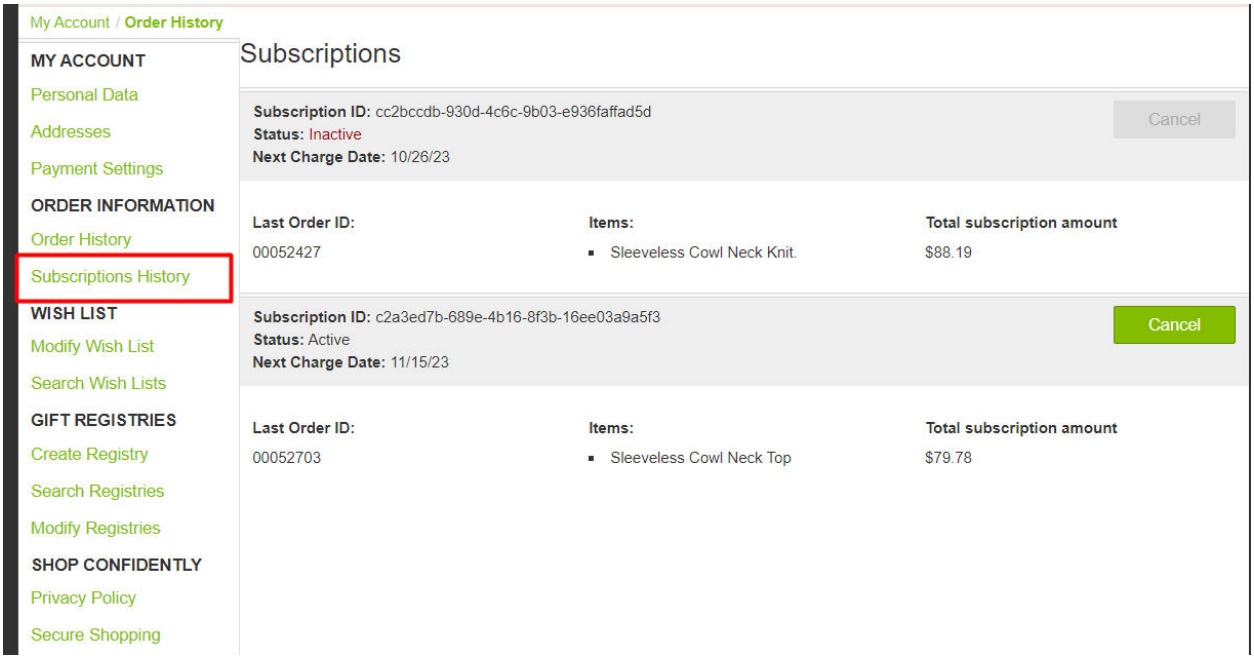
Once the order is created the user profile is updated with subscription data:

- token
- enabled: Status of subscription
- nextChargeDate: A calculated date for the next subscription charge
- subscriptionPeriod: An enumerated value representing the subscription period ('week', 'month', or 'year').

- subscriptionFrequency: The frequency of subscription with will be numeric 1,2,3,4,5,6,15)
- subscriptionProductID: The corresponding subscription product ID.
- lastOrderID: The ID of the last order for the subscription.

2.3.18.4. Account Subscription dashboard

There is a full list of subscriptions per user in my account section. The user will be able to cancel subscriptions. Cancelled subscriptions are displayed with Inactive status. Cancel subscription button will deactivate the customer token and no further charges will be make with it.



2.3.18.5. Recurring Subscription Order Creation

A back-end job to process recurring subscription against each user in SFCC if subscription exists. The job iterates through customers and checks for subscriptions due for payment on the same day.

- Create a new order (with data from lastOrderID) and make a charge call to Klarna using the token and price from the lastOrderID.
- On success - update nextChargeDatebased on subscriptionPeriod and set lastOrderID to the ID of the newly placed order. Update the same in the Subscription Dashboard against the subscription.
- On failure:
    - Retry

      Merchants can configure a retry mechanism where they can set a retry after specific number of days. If the retry fails, then the subscription can be Deactivated. This is generic configuration for retry:

        - Retry: Boolean field with value Yes/No

        - Number of Retry: 1,2 If Retry field is set as Yes, then merchant can configure this field.
        - Retry Frequency: 1, 2 If Retry field is set as Yes, then merchant can configure this field.
    - Cancel subscription – if retry is disabled the subscription is cancelled.

Orders with trial period subscriptions are paid after the trial period is over. On the next charge date, a new order will be created.

New orders are with channel type = SUBSCRIPTIONS.

2.4. Compatibility

This cartridge has been tested against API Version 22.6 (Compatibility Mode: 21.10) and SG version 105.0.0.

2.5. Privacy, Payment

2.5.1. GDPR Compliance

The cartridge is compliant with GDPR recommendation and follows the best practice mentioned here and implementation transmits only required (PII) data to authorize payment method.

2.5.2. EMD (Extra Merchant Data)

The cartridge supports sending additional information on the customer's past purchase history, as well as “Buy Online, Pickup in Store” (BOPIS) store addresses when turned on in custom preferences: “Attachements” (**kpAttachments**). The type of data that can be send as an attachment is mentioned here. EMD is required for certain types of merchant orders and the inclusion of EMD is (e.g.customer_account_info: past interaction with merchant store) generally beneficial to improve acceptance rates.

EMD is included as part of Authorization step in Commerce Cloud checkout. The data send to Klarna is customizable and can be seen in “**int_klarna_payments/scripts/payments/additional CustomerInfo.js**”. 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 .

If the example additionalCustomerInfo.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}$"
                    }
                }
            }
        },
        "other_delivery_address": {
            "type": "array",
            "items": {
                "type": "object",
                "additionalProperties": false,
                "properties": {
                    "shipping_method": {
                        "type": "string",
                        "enum": ["store pick-up", "pick-up point", "registered
box", "unregistered box"]
                    },
                    "shipping_type": {
                        "type": "string",
                        "enum": ["normal", "express"]
                    },
                    "first_name": {
                        "type": "string"
                    },
                    "last_name": {
                        "type": "string"
                    },
                    "street_address": {
                        "type": "string"
                    },
                    "street_number": {
                        "type": "string"
                    },
                    "postal_code": {
                        "type": "string"
                    },
                    "city": {
                        "type": "string"
                    },
                    "country": {
                        "type": "string"
                    }
                }
            }
        }
    }
}
```

Example data:

```
{
    "attachment": {
        "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"
            }],
            "payment_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"
            }],
            "other_delivery_address":[{
                "shipping_method": "store pick-up",
                "shipping_type": "normal",
                "first_name": "Test",
                "last_name": "Customer",
                "street_address": "1487 Bay St",
                "street_number": "",
                "postal_code": "01109",
                "city": "Springfield",
                "country": "US"
            }]
        }
    }
}
```


*Please, note that in cases when the customer uses Guest Checkout, the EMD sent includes **payment_history_full[0].unique_account_identifier** (cqcid value set by SFCC), and all other fields are empty!!*

2.5.3. PCI-DSS Compliance

Important Note: DO NOT SAVE UNENCRYPTED PCI DATA ON THE SERVER!

The virtual card (MCSv3) solution enables settlements using individual virtual card issued against a Klarna order. To be compliant with PCI-DSS requirements, merchant must ensure the data is securely maintained and transmitted as part of their operation in their live store environment. The required steps to ensure this, must be done in consultation with your payment service provider/acquirer and completed prior to go-live. Please review in advance the order export details required for virtual card-based Klarna orders. Any historical decrypted PCI data should also be expunged, regardless of the VCN validity date.

3. Implementation Guide

3.1. Setup of Business Manager

The Klarna Payments LINK Cartridge contains 2 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.

int_klarna_payments – Implements the core storefront functionality.

int_klarna_payments_controllers – Implements the storefront functionality with SG code.

3.1.1. Cartridge Upload & Assignment

Import the two cartridges into UX studio and associate them with a Server Connection.

- Import the "**int_klarna_payments**" cartridge into the SCC Studio Workspace:
    - Open SCC Studio
    - Click File -> Import -> General -> Existing Projects into Workspace
    - Browse to the directory where you saved the "**int_klarna_payments**" cartridge.
    - Click Finish.
    - Click OK when prompted to link the cartridge to the sandbox.

- Import the "**int_klarna_payments_controllers**" cartridge into the SCC Studio Workspace:
    - Open SCC Studio
    - Click File -> Import -> General -> Existing Projects into Workspace
    - Browse to the directory where you saved the "**int_klarna_payments_controllers**" cartridge.
    - Click Finish.
    - Click OK when prompted to link the cartridge to the sandbox.

- Prepend the Klarna cartridges to the effective site cartridge path:
    - Log into the SCC Business Manager.

- Click Administration -> Sites -> Manage Sites.

- Select the desired site.
  - Click on the Settings tab.
  - Prepend "**int_klarna_payments_controllers:int_klarna_payments**" to the "**Cartridges**" field.
  - Click Apply

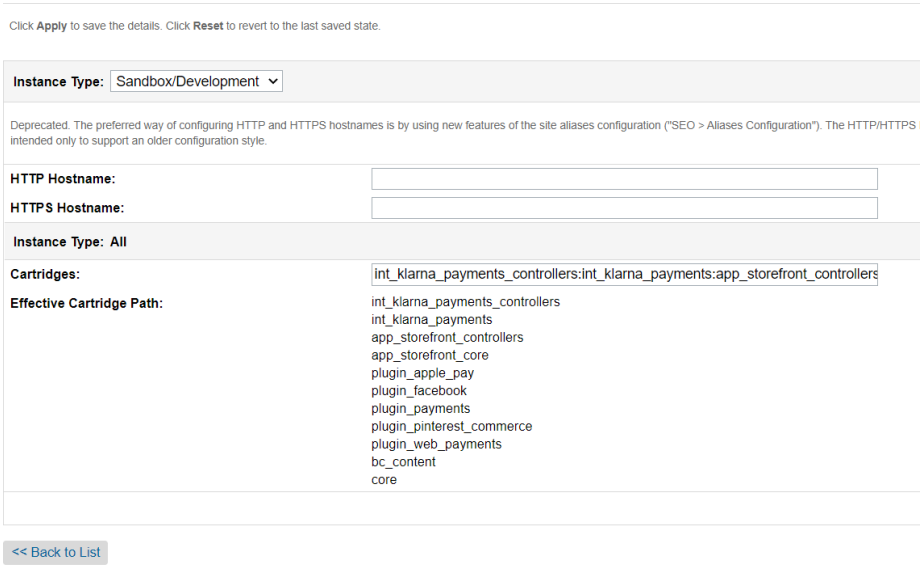


Figure 38 Effective Cartridge Path

3.1.2. Metadata Import

- Go to main directory "**metadata**" folder, review the site-template content, and edit if needed. (Site template is prepared to setup "**SiteGenesis**" and "**RefArch**" sites - you may want to change that to your actual sites and delete the ones that are not needed). Zip the directory and you will have "**site-template.zip**" installation package.
- Log into the SCC Business Manager.
- Click Administration -> Sites Development -> Site Import & Export
- Browse to the directory where you saved the "**site-template.zip**".
- Click "**Upload**"
- Select the uploaded site zip and click "**Import**".

***Note**: Please, review the default **service.xml** file in the **site-template.zip** and update the configuration for Playground and Production accordingly before importing.*

3.2. Configuration

- Add your account settings to the KlarnaCountries Custom Objects.
    - Log into the SCC Business Manager.
    - Select the desired site from the tabs across the top of the page.
    - Click Custom Objects -> Custom Object Editor
    - Change the Object Type dropdown to "**KlarnaCountries**".
    - Click the "**Find**" button.
    - Click the desired country you wish to edit (See screenshot below).
    - Update the required fields as mentioned in "**KlarnaCountries**" section
    - Repeat for the other countries.

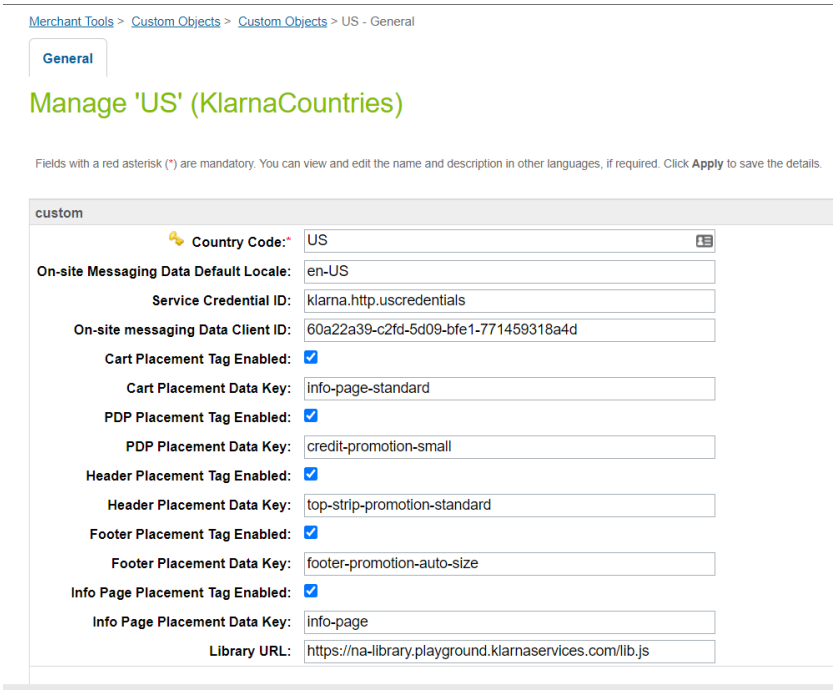


Figure 39 KlarnaCountries Settings

- Configure Klarna Payment Custom Preferences using the SCC Business Manager
    - Log into the SCC Business Manager
    - Select the desired site from the tabs across the top of the page.
    - Click Site Preferences -> Custom Preferences -> KlarnaPayment
    - Fill out the settings as desired. Descriptions of the site preferences are in the **Site Preferences** section.

- Configure Klarna Payment Service using the SCC Business Manager
    - Log into the SCC Business Manager
    - Click Administration > Operations > Services.
    - Click the Credentials tab.
    - Each Klarna credential correspond to one of the **KlarnaCountries** custom objects. Click on the one you want to edit.
    - Enter the MID API username and API password you received from Klarna.
    - Edit URL field if Production environment. Klarna API URLs information - https://developers.klarna.com/api/#api-urls.

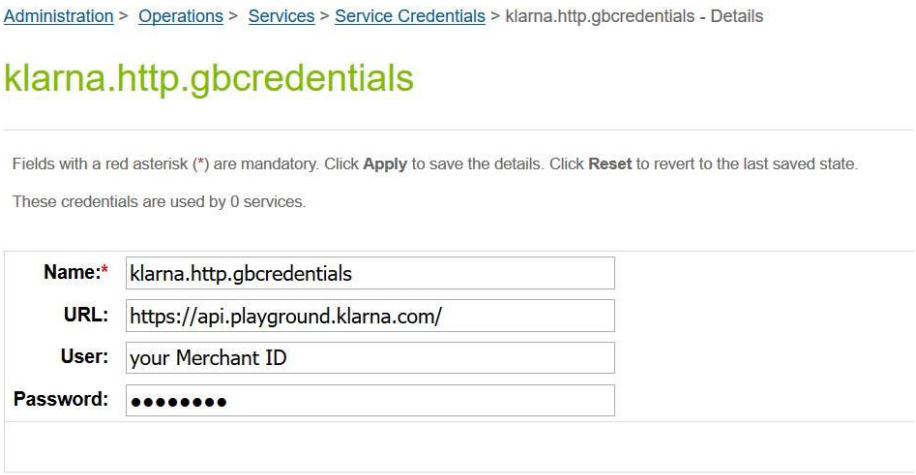
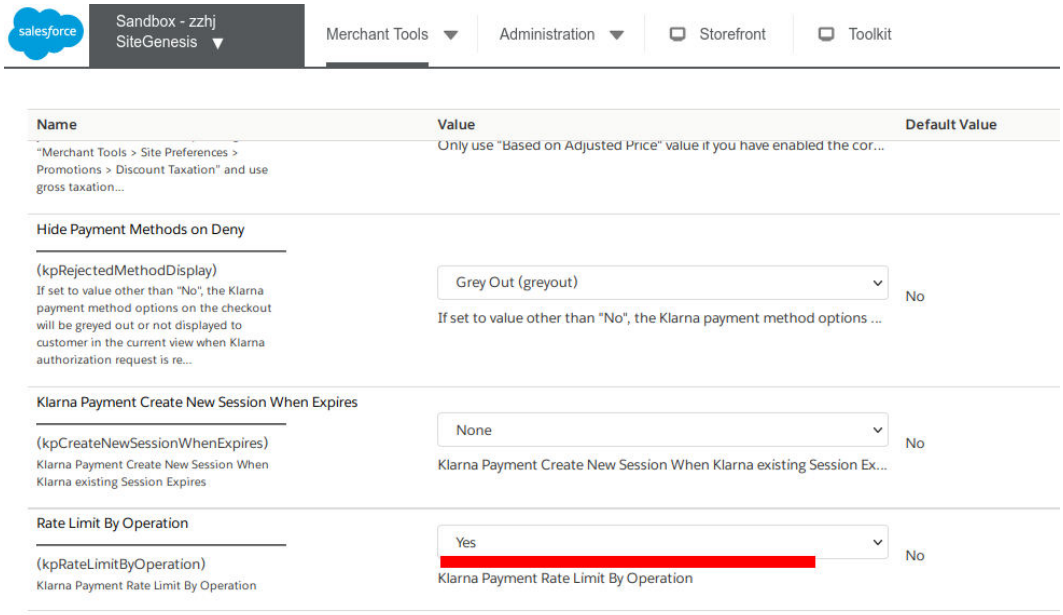
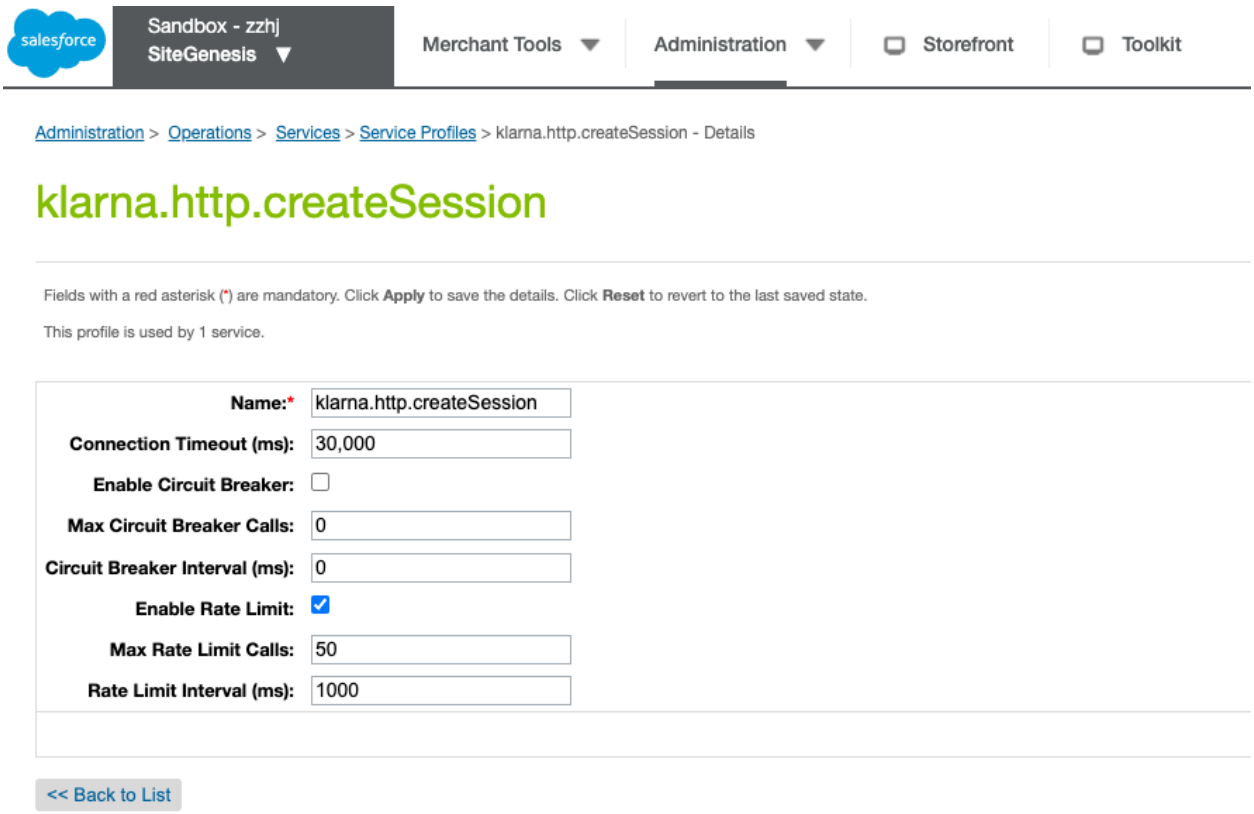


Figure 40 Service Settings

- Configure Klarna Rate Limited Service Profile using the SCC Business Manager (Optional)
    - Log into the SCC Business Manager
    - Navigation : Merchant Tools > Site preferences > Custom Site Preferences group > Klarna Payments
    - Enable Service Limit configuration > Select “Rate Limit By Operation” to Yes. If it is selected to NO, the default service will control the rate limit.



- Configure Custom Rate limits in SCC Business Manager (Optional)
    - Log into the SCC Business Manager
    - Navigation: Administration > Operations > Service
    - Select a required profile (e.g : Klarna.http.createSession)
    - Enable Rate Limit: Check box
    - Max Rate Limit Calls: 50 (e.g.: The higher rate limit agreed with Klarna)
    - Rate Limit Internal (millisecond): 1000
    - Repeat for all Service operation with respective agreed rate limits



3.3. Template Updates

Templates have been updated to support On-site messaging and Addresses forms for Klarna. To be used as reference but feel free to customize the templates to match your specific needs. Final review and sign-off as per project requirements and contract agreements.

3.4. Jobs

3.4.1. Job "OrderCleanUp" (Optional)

This 1-time clean-up job is only applicable to merchants integrated with Klarna Payments cartridge version (< 19.1.6), utilizing (or previously used) virtual card-based settlement (VCN) and stored decrypted card details within Business Manager.

The job iterates over orders with status "**Exported**" and attribute "**custom.kpIsVCN=true**" to remove the sensitive details saved in fields kpVCNPAN, kpVCNCSC, kpVCNExpirationMonth, kpVCNExpirationYear as part of the previous releases. There are no parameters passed to the script.

Upon successful run, the job will log the result of processed orders in the custom debug log located in "**webdav/Sites/Logs**". Depending on the setup, you will receive a message for the processed orders count for each storefront or message that there are no orders needing update.

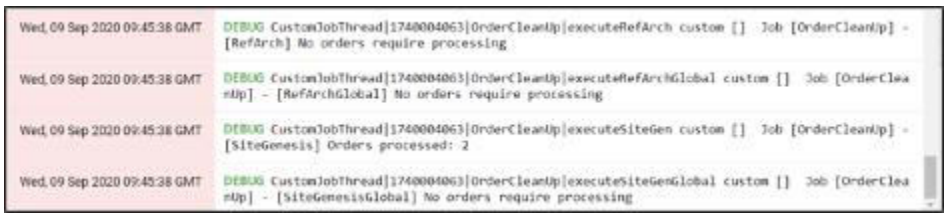


Figure 41 Job Logs

Upon error, the cause of the failure (message and stack) will be logged in the standard error log.

Figure 42 Job Logs

To setup the job, go to **Administration > Operations > Import & Export** and import file "**jobs.xml**". Out of the box, the XML file includes only the RefArch scope, but it can be configured with multiple flows if you have more than one site using this functionality as seen bellow. Each site should be added as a separate flow.

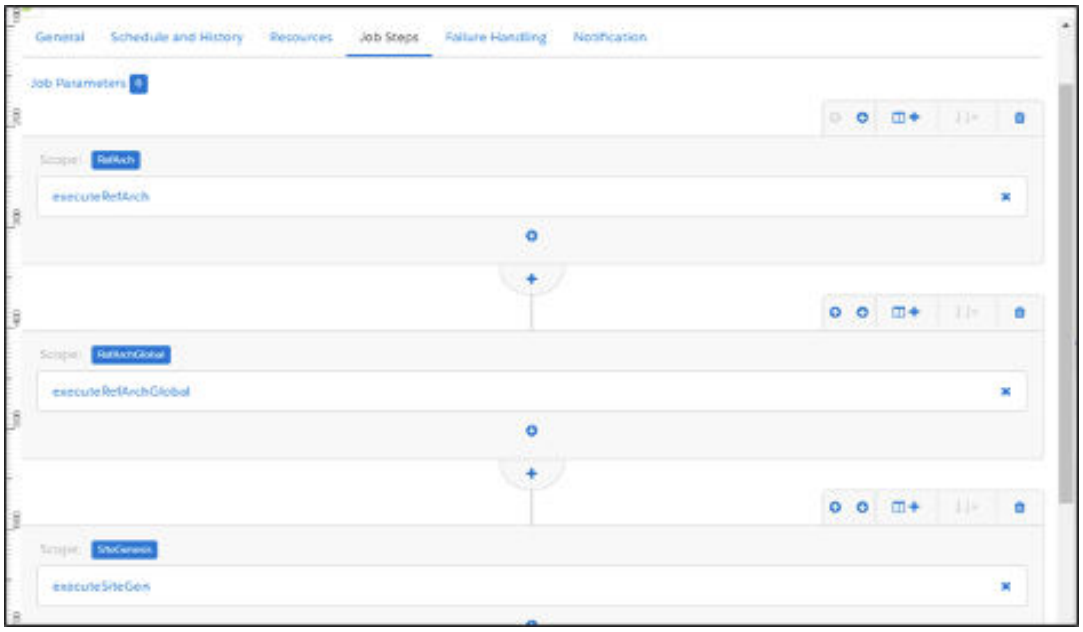


Figure 43 Job Steps

To set up the required job parameters and add new flow, follow the bellow steps. If you only have one storefront and need to change the scope to the correct one – proceed to steps 4-5 directly.

1. Click on the "Add a sequential flow" button at the bottom of the current flow.

Figure 44 Add New Job Step

2. Once done, click on “Configure a step” button within the flow. In the flyout that has just opened search for “script” and select “ExecuteScriptModule”.

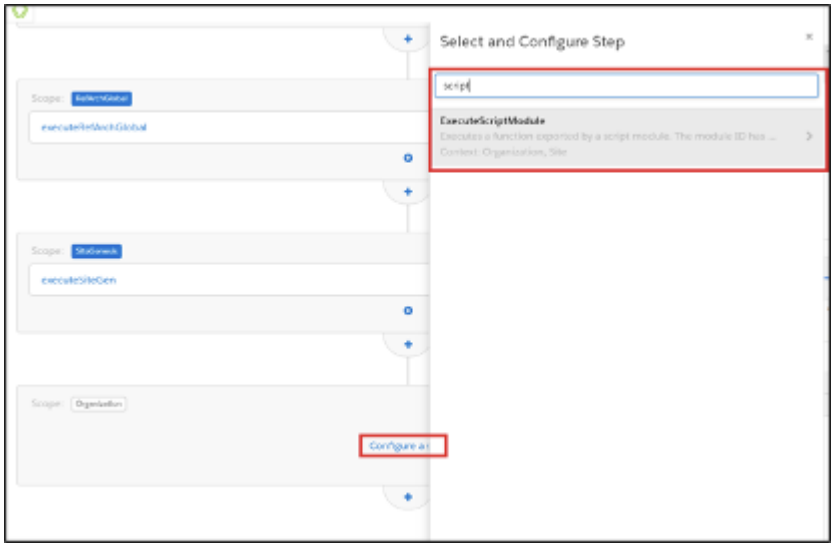


Figure 45 Configure Step

3. In the flyout populate these fields and click the “Assign” button.

   3.1. ID – Enter any meaningful name in the field. In case you have multiple flows, this should not be a duplicate one. If you enter a duplicate name, the details won’t be saved and you will see an error message.

   3.2. ExecuteScriptModule.Module – Enter the location of the “OrderCleanUpJob.js “ file. Out of the box it should be “int_klarna_payments/cartridge/scripts/job/OrderCleanUpJob.js” or the location where you have placed the script file.

   3.3. ExecuteScriptModule.FunctionName – Leave the field value to “execute”

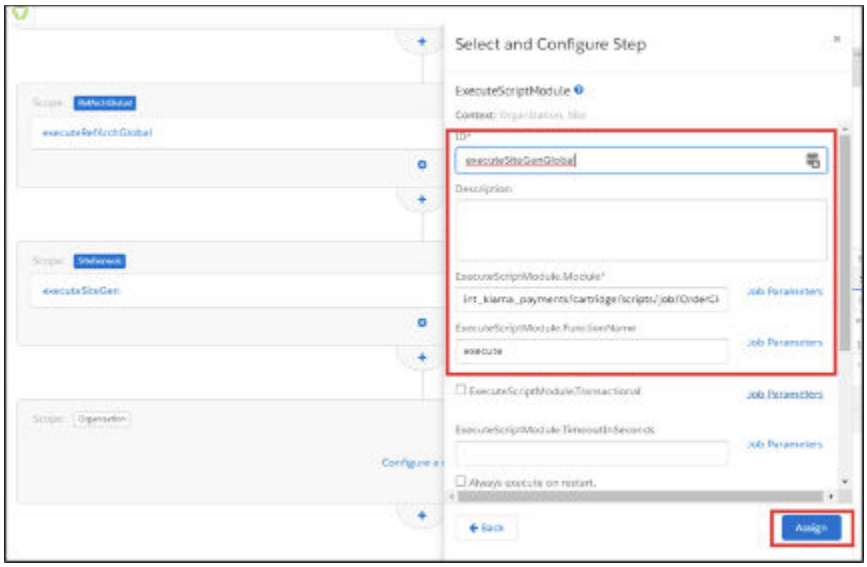


Figure 46 Configure Step (cont.)

4. Once the step has been added, you should make sure to assign it to the correct site scope. Click on "Organization" and select "Specific Sites" from the drop-down.

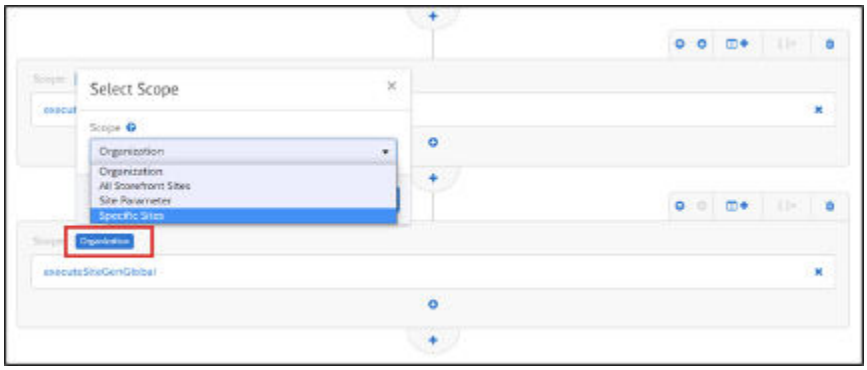


Figure 47 Job Scope

5. From the list of sites, select the respective site ID (i.e. SiteGenesisGlobal) and click on "**Assign**".

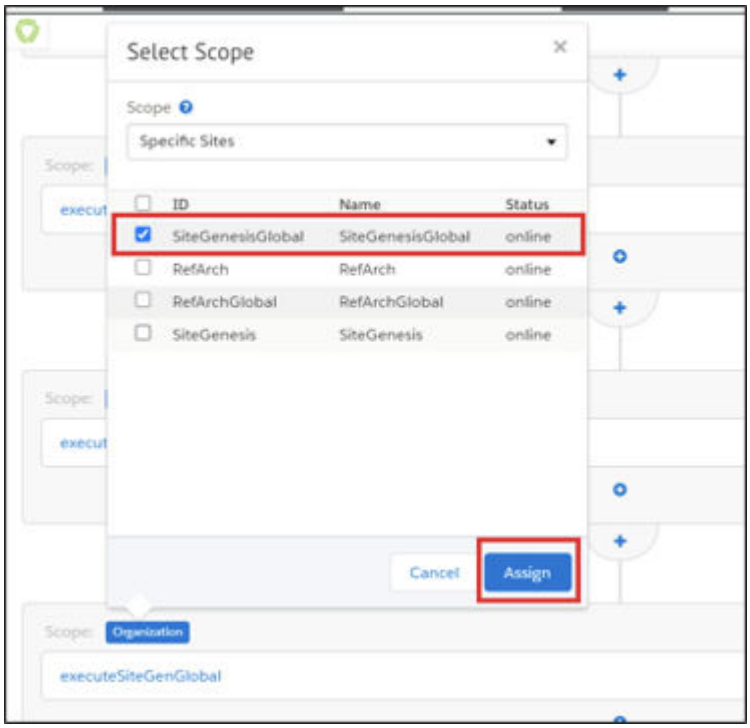


Figure 48 Job Scope (cont.)

6. Repeat steps 1-5 for each site/storefront that you have using Klarna VCN and nned additional configuration.

3.5.1. Job "RecurringOrders"

The job iterates over all customers checking for subscriptions entries. Process all subscription entries eligible for charge – subscription should be enabled and nextChargeDate or nextRetryDate should match the current date. New SFCC orders are created and the old ones are replaced. Orders with expiring trial period, are only charged.

By default in jobs.xml the job is configured to run on RefArch site but this can be changed either in file or in storefront.

The job has one job step – createOrder with following configuration:

- ExecuteScriptModule.Module - **int_klarna_payments/cartridge/scripts/job/RecurringOrdersJob.js**
- ExecuteScriptModule.FunctionName – **execute**

The job is executed on site level.

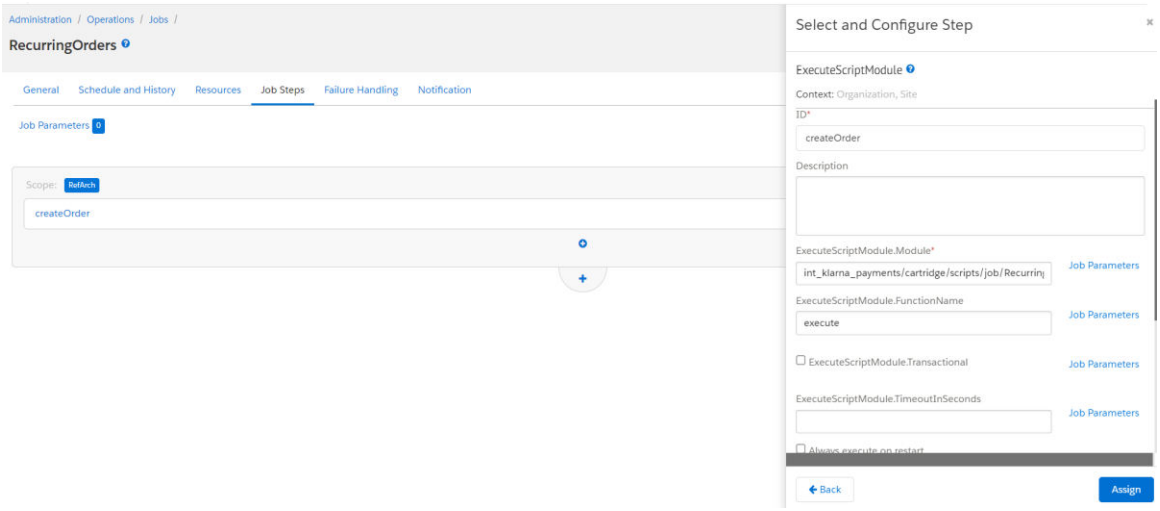


Figure 49 RecurringOrders job

3.5. Custom Code

Integration may vary based on the customers' storefront. Site Genesis version 105.0.0 is used as a reference to demonstrate Klarna Payments integration.

3.5.1. Template Modifications

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

- default/checkout/summary/summary.isml
- default/checkout/billing/billing.isml
- default/checkout/billing/paymentmethods.isml
- default/checkout/shipping/minishipments.isml
- default/components/header/header.isml
- default/components/footer/footer.isml
- default/components/footer/footer_UI.isml
- default/product/producttopcontentPS.isml
- default/product/productcontent.isml
- default/product/productcontent.isml

- default/checkout/cart/cart.isml
- default/mail/orderconfirmation.isml
- default/components/order/ordrdetailsemail.isml
- default/checkout/cart/minicart.isml

3.5.1.1. default/checkout/summary/summary.isml

Add Code:

```
<isif condition="${session.privacy.KlarnaPaymentsFinalizeRequired}">
    <script>
        <isinclude template="/resources/klarnapaymentsresources.isml"/>
    </script>
    <script type="text/javascript" src="${URLUtils.staticURL('/js/klarna-payments-
finalize.js')}"></script>
    <script src=" https://x.klarnacdn.net/kp/lib/v1/api.js" async></script>
</isif>

<isif condition="${!session.privacy.KlarnaPaymentsFinalizeRequired &&
dw.system.Site.getCurrent().getCustomPreferenceValue(
'kpUseAlternativePaymentFlow' )}">
    <isinclude template="klarnapayments/modules.isml"/>
    <isset name="billingAddress" value="${pdict.Basket.billingAddress}"
scope="page"/>
    <iskpbillingaddresshelper p_address="${billingAddress}"
p_email="${pdict.Basket.customerEmail}"/>

    <isloop items="${JSON.parse(session.privacy.KlarnaPaymentMethods)}"
var="klarnaPaymentMethod">
        <div class="payment-method" data-
test="${session.privacy.KlarnaPaymentsFinalizeRequired}" data-
method="${'klarna_payments_' + klarnaPaymentMethod.identifier}" hidden>
            <div id="${'klarna_payments_' + klarnaPaymentMethod.identifier +
'_container'}" style="text-align: center;"></div>
            <isif condition="${empty(pdict.Basket.custom.kpSessionId)}">
                <div class="klarna_payments_error" style="text-align: center;
font-weight: bold; color: red;"><isprint
value="${KlarnaPaymentNotAvailable}"/></div>
            </isif>
        </div>
    </isloop>

    <script>
```

```
        <isinclude template="/resources/klarnapaymentsresources.isml"/>
    </script>
    <script type="text/javascript" src="${URLUtils.staticURL('/js/klarna-payments-alternative-flow.js')}"></script>
    <script src=" https://x.klarnacdn.net/kp/lib/v1/api.js" async></script>
</isif>
```

Before `</isdecorate>` closing tag as shown below at the end of file:

summary.isml

```
            </fieldset>
        </form>

    </div>
    <isif condition="${session.privacy.KlarnaPaymentsFinalizeRequired}">
        <script>
            <isinclude template="/resources/klarnapaymentsresources.isml"/>
        </script>
        <script type="text/javascript" src="${URLUtils.staticURL('/js/klarna-payments-finalize.js')}"></script>
        <script src=" https://x.klarnacdn.net/kp/lib/v1/api.js" async></script>
    </isif>

    <isif condition="${!session.privacy.KlarnaPaymentsFinalizeRequired && dw.system.Site.getCurrent().getCustomPreferenceValue( 'kpUseAlternat
        <isinclude template="klarnapayments/modules.isml"/>
        <isset name="billingAddress" value="${pdict.Basket.billingAddress}" scope="page"/>
        <iskpbillingaddresshelper p_address="${billingAddress}" p_email="${pdict.Basket.customerEmail}"/>

        <isloop items="${JSON.parse(session.privacy.KlarnaPaymentMethods)}" var="klarnaPaymentMethod">
            <div class="payment-method" data-test="${session.privacy.KlarnaPaymentsFinalizeRequired}" data-method="${'klarna_payments_' + klar
                <div id="${'klarna_payments_' + klarnaPaymentMethod.identifier + '_container'}" style="text-align: center;"></div>
                <isif condition="${empty(pdict.Basket.custom.kpSessionId)}">
                    <div class="klarna_payments_error" style="text-align: center; font-weight: bold; color: red;"><isprint value="${KlarnaPaym
                </isif>
            </div>
        </isloop>

        <script>
            <isinclude template="/resources/klarnapaymentsresources.isml"/>
        </script>
        <script type="text/javascript" src="${URLUtils.staticURL('/js/klarna-payments-alternative-flow.js')}"></script>
        <script src=" https://x.klarnacdn.net/kp/lib/v1/api.js" async></script>
    </isif>

</isdecorate>
```

Figure 50 Modifications in summary.isml

3.5.1.2. 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>

<link rel="stylesheet" href="${URLUtils.staticURL('/css/klarna-payments.css')}" />
```

Before `</isdecorate>` closing tag as shown below:

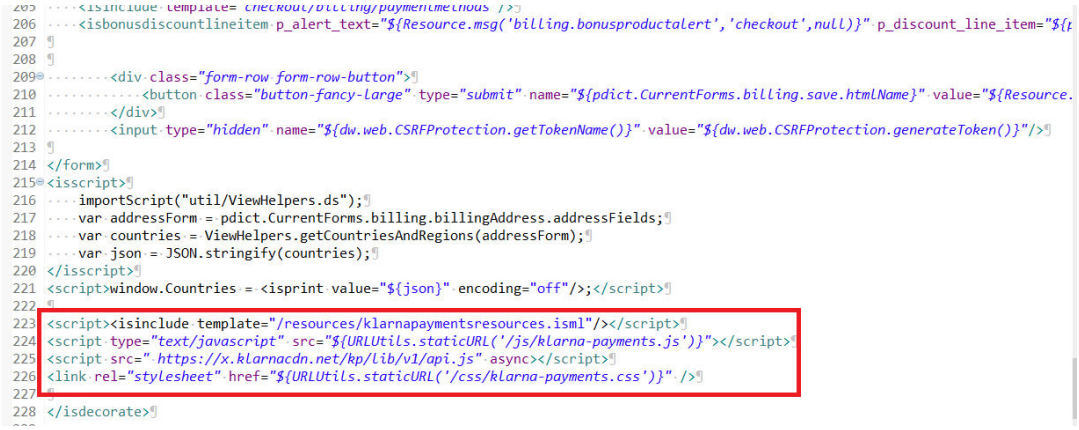


Figure 51 Modifications in billing.isml

3.5.1.3. 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:

Figure 52 Modifications in paymentmethods.isml

Add Code:

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

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

Figure 53 Modifications in paymentmethods.isml (cont.)

Add Code:

```
<iscomment>

    Klarna Payments

    ------------------------------------------------------------

</iscomment>

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

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

Figure 54 Modifications in paymentmethods.isml (cont.)

3.5.1.4. default/checkout/shipping/minishipments.isml

Add the following code:

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

In the end beginning of the file as shown below:

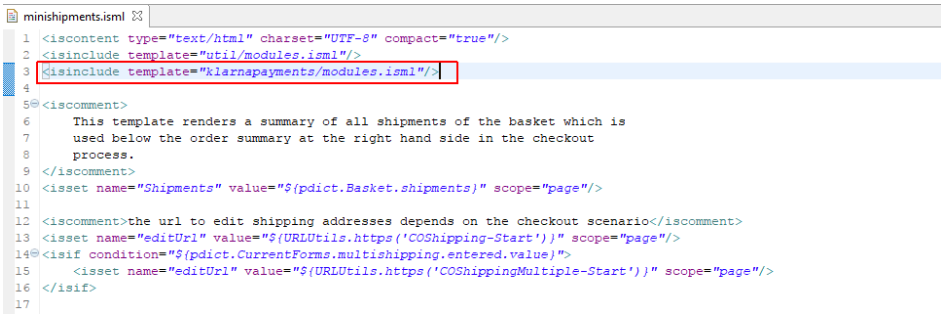


Figure 55 Modifications in minishipments.isml

Add the following code:

```
<iskpaddresshelper p_shipment="${shipment}"
p_address="${shipment.shippingAddress}"/>
```

After `<isminicheckout_address p_address="${shipment.shippingAddress}"/>` as shown below:

Figure 56 Modifications in minishipments.isml (cont.)

3.5.1.5. default/components/footer/footer_UI.isml

Add Code:

```
<isinclude template="klarnapayments/scripts.isml"/>
```

Before `<script src="${URLUtils.staticURL('/js/app.js')}"></script>` script tag as shown below:

```
27
28 <isinclude template="klarnapayments/scripts.isml"/>
29
30 <script src="${URLUtils.staticURL('/js/app.js')}"></script>
```

Figure 57 Modifications in footer_UI.isml

3.5.1.6. default/components/footer/footer.isml

Add the following code:

```
<!-- Klarna OSM footer -->
<div class="klarna-footer">
    <isinclude template="klarnapayments/modules.isml"/>
    <iskosmfooter />
</div>
<!-- /Klarna OSM footer -->
<!-- Klarna KEB form -->
<iskebform />
<!-- Klarna KEB form -->
```

right after the `</footer>` end tag and before `<iscontentasset aid="footer-copy"/>` as shown below:

Figure 58 Modifications in footer.isml

3.5.1.7. default/product/producttopcontentPS.isml

Add the following code:

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

```
<iskosmpdp p_product="${psProduct}" />
```

, right under the `pricing` template include as shown below:

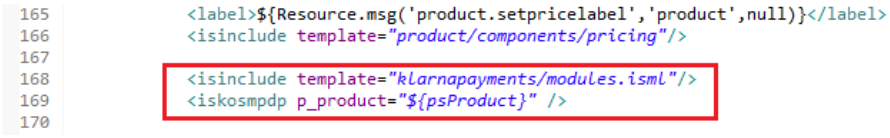


Figure 59 Modifications in producttopcontentPS.isml

3.5.1.8. default/product/productcontent.isml

Add the following code:

```
<isif condition="${!isQuickView}">
```

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

```
    <iskosmpdp p_product="${pdict.Product}" />
```

```
</isif>
```

, right under the `pricing` template include as shown below:

```
77    <isinclude template="product/components/pricing"/>
78
79    <isif condition="${!isQuickView}">
80    <isinclude template="klarnapayments/modules.isml"/>
81    <iskosmpdp p_product="${pdict.Product}" />
82    </isif>
83
84    <isset name="pam" value="${pdict.Product.getAttributeModel()}" scope="page"/>
```

Figure 60 Modifications in producttopcontent.isml

3.5.1.9. js/pages/product/variant.js

Add the following code:

```
if (window.Klarna && window.Klarna.OnsiteMessaging) {
    window.Klarna.OnsiteMessaging.refresh();
}
```

in the variation update callback, on line 35

```
29        callback: function () {
30            if (SitePreferences.STORE_PICKUP) {
31                productStoreInventory.init();
32            }
33            image.replaceImages();
34            tooltip.init();
35            if (window.Klarna && window.Klarna.OnsiteMessaging) {
36                window.Klarna.OnsiteMessaging.refresh();
37            }
38        }      Vitalii Dolynskyi, 14 months ago • SiteGenesis initial c
39    });
```

Figure 61 Modifications in variant.js

3.5.1.10. default/checkout/cart/cart.isml

Add the following code on line 5:

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

```
cart.isml
1  <iscontent type="text/html" charset="UTF-8" compact="true"/>
2  <isdecorate template="checkout/cart/pt_cart">
3      <isinclude template="util/modules" />
4      <isinclude template="util/reporting/ReportBasket.isml" />
5      <isinclude template="klarnapayments/modules.isml"/>
6
7      <isset name="enableCheckout" value="${pdict.EnableCheckout}" scope="page" />
8      <isif condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('enableStorePickUp')}">
9      <isset name="store" value="${dw.catalog.StoreMgr.getStore(pdict.CurrentSession.custom.storeId)}" scope="page" />
10     </isif>
11     <isslot id="cart-banner" description="Banner for Cart page" context="global" />
```

Figure 62 Modifications in cart.isml

Add the following code on line 823:

```
<iskosmcart p_lineitemctnr="${pdict.Basket}" />
```

, right after the `<isordertotals>`:

```
cart.isml
811                                 </div>
812                             </td>
813                         </tr>
814                     </isloop>
815                 </tfoot>
816             </table>
817
818             <div class="cart-footer">
819
820                 <input type="hidden" name="${pdict.CurrentForms.cart.updateCart.htmlName}" value="${pdict.CurrentFor
821                 <div class="cart-order-totals">
822                     <isordertotals p_lineitemctnr="${pdict.Basket}" p_totallabel="${Resource.msg('global.estimatedto
823                     <iskosmcart p_lineitemctnr="${pdict.Basket}" />
824                 </div>
825                 <div class="cart-coupon-code">
826
827                     <input type="text" name="${pdict.CurrentForms.cart.couponCode.htmlName}" id="${pdict.CurrentForms
828
829                     <button type="submit" value="${pdict.CurrentForms.cart.addCoupon.htmlName}" name="${pdict.Current
```

Figure 63 Modifications in cart.isml (cont.)

Add the following code on lines 33 & 864:

```
<iskebcart />
```

, right after the `<isif condition="${enableCheckout}">`

```
28      <div class="cart-actions cart-actions-top">
29          <iscomment>continue shop url is a non-secure but checkout needs a secure and that is why separate forms!</isc
30          <form class="cart-action-checkout" action="${URLUtils.httpsContinue()}" method="post" name="${pdict.CurrentFo
31              <fieldset>
32                  <isif condition="${enableCheckout}">
33                      <iskebcart />
34                      <button class="button-fancy-large" type="submit" value="${Resource.msg('global.checkout','locale
35                          ${Resource.msg('global.checkout','locale',null)}
36                      </button>
37                      <isapplepay></isapplepay>
38                  <iselse/>
39                      <button class="button-fancy-large" disabled="disabled" type="submit" value="${Resource.msg('globa
40                          ${Resource.msg('global.checkout','locale',null)}
41                      </button>
42                  </isif>
43              </fieldset>
44          </form>
```

Figure 64 Modifications in cart.isml (cont.)

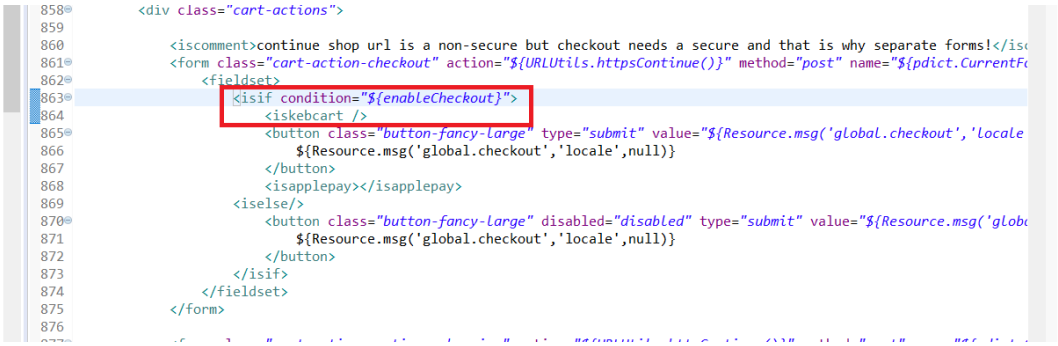


Figure 65 Modifications in cart.isml (cont.)

Add the following code on line 182:

```
<isset name="kpIsStandardProduct"
    value="${!empty(product.custom.kpIsStandardProduct) ?
product.custom.kpIsStandardProduct : true}" scope="page" />

<iskpsubscription lineitem="${lineItem}"
showsubscription="${product.custom.kpIsSubscriptionProduct}"
disablesubscribe="${product.custom.kpIsSubscriptionProduct &&
!kpIsStandardProduct}"

kptrialdaysusage="${product.custom.kpTrialDaysUsage}"

lmkpsubscription="${lineItem.custom.kpSubscription}"/>
```

after deliveryoptions include:

```
<isinclude template="checkout/cart/storepickup/deliveryoptions" />
```

Figure 66 Modifications in cart.isml (cont.)

Add the following code on line 65:

```
<iselseif condition="${pdict.BasketStatus.code != null && pdict.BasketStatus.code
== 'SubscriptionError'}">

      ${pdict.BasketStatus.message}
```

in div with class ‘error-form’.

```
60          <div class="error-form">
61              <i class="fa fa-exclamation-triangle fa-2x pull-left"></i>
62              <isif condition="${pdict.BasketStatus.code != null && pdict.BasketStatus.code=='CouponError'}">
63                  ${Resource.msg('cart.cartcouponinvalid','checkout',null)}
64              <iselseif condition="${pdict.BasketStatus.code != null && pdict.BasketStatus.code=='TaxError'}">
65                  ${Resource.msg('cart.taxinvalid','checkout',null)}
66              <iselseif condition="${pdict.BasketStatus.code != null && pdict.BasketStatus.code == 'SubscriptionError'}">
67                  ${pdict.BasketStatus.message}
68              <iselse/>
69                  ${pdict.BasketStatus.code}
70                  ${Resource.msg('cart.carterror','checkout',null)}
71              </isif>
72          </div>
```

Figure 67 Modifications in cart.isml (cont.)

Add following code on line 862:

```
<isinclude template="klarnapayments/subscription/cartSubscriptionDetails" />
```

at the end in div 'cart-footer'

```
858                     </div>
859                 </isif>
860             </div>
861
862             <isinclude template="klarnapayments/subscription/cartSubscriptionDetails" />
863
864         </div>
865
```

Figure 68 Modifications in cart.isml (cont.)

3.5.1.11. default/components/header/header.isml

Add the following code:

```
<!-- Klarna OSM header -->

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

<iskosmheader />

<!-- /Klarna OSM header -->
```

At the end of the file as shown below:

header.isml

```
                <span>${Resource.msg('global.header.storelocator', 'locale', null)}</span>
            </a>
        </li>

        <iscomment>INCLUDE: Customer login information, login, etc. (contains personal information, do not cache)</isc
        <isinclude url="${URLUtils.url('Home-IncludeHeaderCustomerInfo')}"/>

    </ul>

    <iscomment>Country Selector</iscomment>
    <isinclude template="components/header/countryselector"/>

</nav>

<iscomment>INCLUDE: Mini-cart, do not cache</iscomment>
<div id="mini-cart">
    <isinclude url="${URLUtils.url('Cart-MiniCart')}"/>
</div>

</div><!-- /header -->

<!-- Klarna OSM header -->
<isinclude template="klarnapayments/modules.isml"/>
<iskosmheader />
<!-- /Klarna OSM header -->
```

Figure 69 Modifications in header.isml

3.5.1.12. default/mail/orderconfirmation.isml

Add the following code:

```
<tr>
    <td style="font-size:12px;font-family:arial;padding:20px 10px;vertical-align:top;">
        <isset name="confirmationAsset" value="${require('*/cartridge/scripts/util/klarnaHelper'.getConfirmationEmailAsset()}" scope="page" />
        <isprint value="${confirmationAsset}" encoding="off" />
    </td>
</tr>
```

Before the `</table>` closing tag as shown:

orderconfirmation.isml

```
${Resource.msg('order.orderconfirmation-email.storephone','order',null)}
|>

colspan="2" style="font-size:12px;font-family:arial;padding:20px 10px;vertical-align:top;">

<table  style="background:#ffffff;border:1px solid #999999;width:680px;">
    <tr>
        <th style="background:#cccccc;padding:5px 20px;font-size:12px;font-family:arial;text-align:left;">${Resource.msg('confirmation.thankyou','ch
    </tr>
    <tr>
        <td style="font-size:12px;font-family:arial;padding:20px 10px;vertical-align:top;">
            <p>${Resource.msg('confirmation.message','checkout',null)}</p>
            <p>${Resource.msg('confirmation.contact','checkout',null)}</p>
        </td>
    </tr>
    <tr>
        <td style="font-size:12px;font-family:arial;padding:20px 10px;vertical-align:top;">
            <isset name="confirmationAsset" value="${require('*/cartridge/scripts/util/KlarnaHelper').getConfirmationEmailAsset()}" scope="page" />
            <isprint value="${confirmationAsset}" encoding="off" />
        </td>
    </tr>
</table>

|>
```

Figure 70 Modifications in orderconfirmation.isml

3.5.1.13. *default/components/order/ordrdetailsemail.isml*

Add the following code:

```
<iselseif condition="${paymentInstr.paymentMethod.equals('Klarna')}" >
${Resource.msg('email.order.reference','klarnapayments',null)}:
${Order.custom.kpOrderID} <br />
```

Before the closing `</isif>` tag on line 51 as shown below:

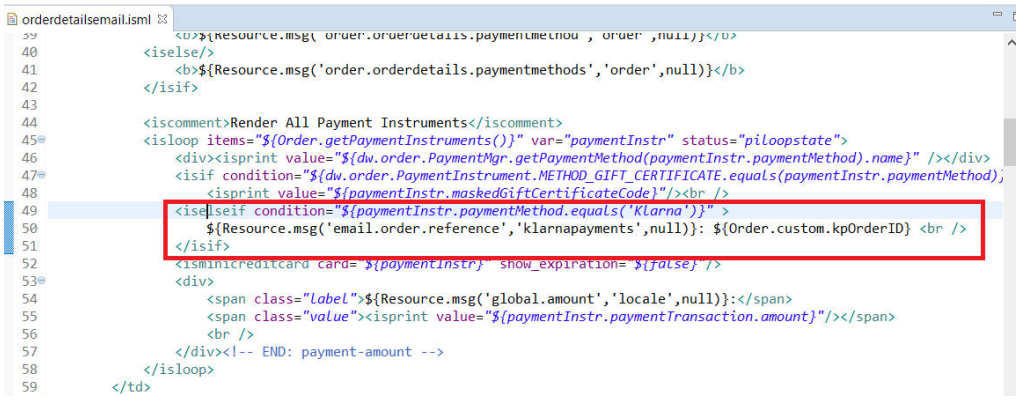


Figure 71 Modifications in orderdetailsemail.isml

3.5.1.14. *default/checkout /cart /minicart.isml*

Add the following code:

```
<issetname="KlarnaOSM"
value="${require('*/cartridge/scripts/marketing/klarnaOSM')}" scope="page" />
<isif condition="${KlarnaOSM.isEnabledMCExpressButton()}">
    <isinclude template="klarnapayments/modules.isml"/>
    <ismckeb />
    <script   type="text/javascript"   src="${URLUtils.staticURL('/js/minicart-
keb.js')}"></script>
</isif>
```

Before the `<a class="mini-cart-link-checkout"...` tag on line 74 as shown below:

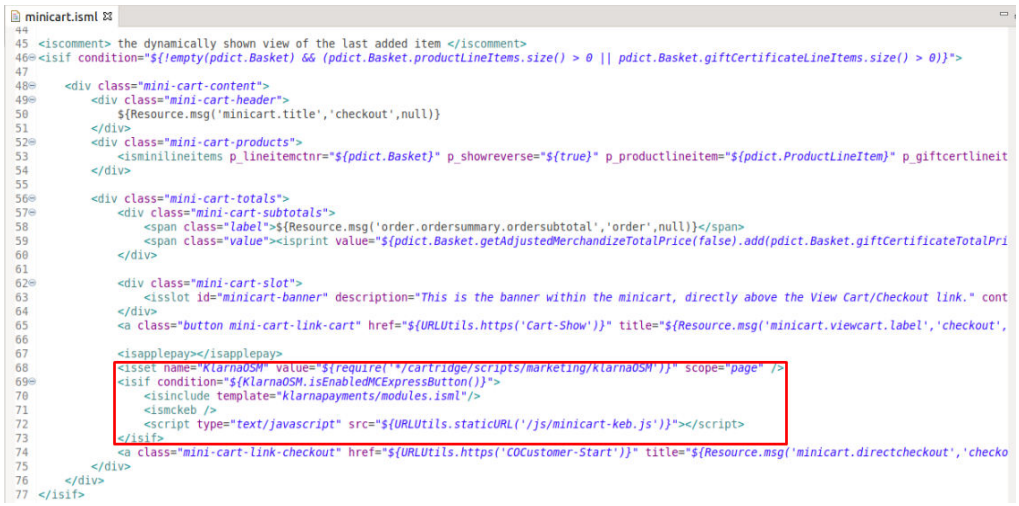


Figure 72 Modifications in minicart.isml

3.5.1.10. js/pages/cart.js

Add the following code after line 6:

```
    util = require('../util'),
```

```
var account = require('./account'),
    bonusProductsView = require('../bonus-products-view'),
    quickview = require('../quickview'),
    util = require('../util'),
    cartStoreInventory = require('../storeinventory/cart');
```

Figure 73 Modifications in cart.js

Add the following code after line 40:

```
    $('body').on('change', '.kp-subscription', async function () {
        var isSubscribed = $(this).is(":checked");
        var productID = $(this).data('pid');
        var url = $(this).data('action');
        var uuid = $(this).data('uuid');

        var urlParams = {
            pid: productID,
            subscription: isSubscribed,
            uuid: uuid
        };
        url = util.appendParamsToUrl(url, urlParams);

        $.ajax({
            url: url,
```

```
            type: 'get',
            context: this,
            dataType: 'json',
            success: function (data) {
                if (data.isSubscriptionBasket) {
                    $('.subscription-data').show();
                } else {
                    $('.subscription-data').hide();
                }
            },
            error: function (err) {
                if (err.responseJSON.redirectUrl) {
                    window.location.href = err.responseJSON.redirectUrl;
                }
            }
        });
    });
```

at the end of initializeEvents() function.

```
41
42  $('body').on('change', '.kp-subscription', async function () {
43      var isSubscribed = $(this).is(":checked");
44      var productID = $(this).data('pid');
45      var url = $(this).data('action');
46      var uuid = $(this).data('uuid');
47
48      var urlParams = {
49          pid: productID,
50          subscription: isSubscribed,
51          uuid: uuid
52      };
53      url = util.appendParamsToUrl(url, urlParams);
54
55      $.ajax({
56          url: url,
57          type: 'get',
58          context: this,
59          dataType: 'json',
60          success: function (data) {
61              if (data.isSubscriptionBasket) {
62                  $('.subscription-data').show();
63              } else {
64                  $('.subscription-data').hide();
65              }
66          },
67          error: function (err) {
68              if (err.responseJSON.redirectUrl) {
69                  window.location.href = err.responseJSON.redirectUrl;
70              }
71          }
72      });
73  });
74
75  $('body').on('change', '.subscription-period, .subscription-frequency', function () {
76      var selectedValue = $('option:selected', this).val();
```

Figure 74 Modifications in cart.js (cont.)

Add the following code on line 75:

```
        $('body').on('change',   '.subscription-period,   .subscription-frequency',
function () {
```

```
        var selectedValue = $('option:selected', this).val();
        console.log(selectedValue);
        var url = $(this).data('url');
        var subscriptionField = $(this).data('field');

        var urlParams = {
            selectedValue: selectedValue,
            subscriptionField: subscriptionField
        };

        url = util.appendParamsToUrl(url, urlParams);

        $.ajax({
            url: url,
            type: 'get',
            context: this,
            dataType: 'json',
            success: function (data) {
                if (data.error) {
                    console.error(data.errorMessage);
                }
            },
            error: function (err) {
                if (err.responseJSON.redirectUrl) {
                    window.location.href = err.responseJSON.redirectUrl;
                }
            }
        });
    });
}
```

```
$('body').on('change', '.subscription-period, .subscription-frequency', function () {
    var selectedValue = $('option:selected', this).val();
    console.log(selectedValue);
    var url = $(this).data('url');
    var subscriptionField = $(this).data('field');

    var urlParams = {
        selectedValue: selectedValue,
        subscriptionField: subscriptionField
    };

    url = util.appendParamsToUrl(url, urlParams);


    $.ajax({
        url: url,
        type: 'get',
        context: this,
        dataType: 'json',
        success: function (data) {
            if (data.error) {
                console.error(data.errorMessage);
            }
        },
        error: function (err) {
            if (err.responseJSON.redirectUrl) {
                window.location.href = err.responseJSON.redirectUrl;
            }
        }
    });
});
```

Figure 75 Modifications in cart.js (cont.)

3.5.1.11. default/checkout/shipping/singleshipping.isml

Add the following code after line 98:

```
<fieldset>
    <isinclude template="checkout/shipping/subscriptionDetails" />
</fieldset>
```

after shipping method list.

```
<div id="shipping-method-list">
    <isinclude url="${URLUtils.https('COShipping-UpdateShippingMethodList')}"/>
</div>

<fieldset>
    <isinclude template="checkout/shipping/subscriptionDetails" />
</fieldset>
</isif>
<fieldset>
```

Figure 76 Modifications in singleshipping.isml

3.5.1.12. scripts/cart/ValidateCartForCheckout.js

Add the following code block in validate function:

```
var Resource = require('dw/web/Resource');
var SubscriptionHelper = require('*/cartridge/scripts/subscription/subscriptionHelper');
var subValidation = SubscriptionHelper.validateCartProducts(basket);
if (subValidation && subValidation.error) {
    return {
        BasketStatus: new Status(Status.ERROR, 'SubscriptionError', subValidation.message)
```

```
        };
    }
    var subscriptionUserError = session.privacy.guest_subscription_error;
    session.privacy.guest_subscription_error = null;
    if (subscriptionUserError) {
        var msg = Resource.msg('klarna.subscription.checkout.guestuser.error',
'subscription', null);
        return {
            BasketStatus: new Status(Status.ERROR, 'SubscriptionError', msg),
            EnableCheckout: true
        };
    }
```

before the DONE section on line 115.

```
113
114  var Resource = require('dw/web/Resource');
115  var SubscriptionHelper = require('*/cartridge/scripts/subscription/subscriptionHelper');
116  var subValidation = SubscriptionHelper.validateCartProducts(basket);
117  if (subValidation && subValidation.error) {
118      return {
119          BasketStatus: new Status(Status.ERROR, 'SubscriptionError', subValidation.message)
120      };
121  }
122
123  var subscriptionUserError = session.privacy.guest_subscription_error;
124  session.privacy.guest_subscription_error = null;
125  if (subscriptionUserError) {
126      var msg = Resource.msg('klarna.subscription.checkout.guestuser.error', 'subscription', null);
127      return {
128          BasketStatus: new Status(Status.ERROR, 'SubscriptionError', msg),
129          EnableCheckout: true
130      };
131  }
132
133
134  // ======================================================
135  // =====            DONE                          =====
136  // ======================================================
```

Figure 77 Modifications in ValidateCartForCheckout.js

4.5.1.18. scripts/util/Resource.ds

Add the following code block before validation messages section:

```
        CANCEL_SUBSCRIPTION                 :
Resource.msg('heading.cancel.subscriptions', 'subscription', null),

        CANCEL                              : Resource.msg('global.cancel',
'locale', null),
```

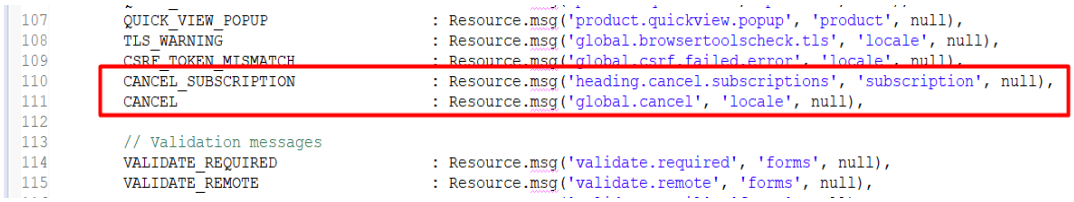


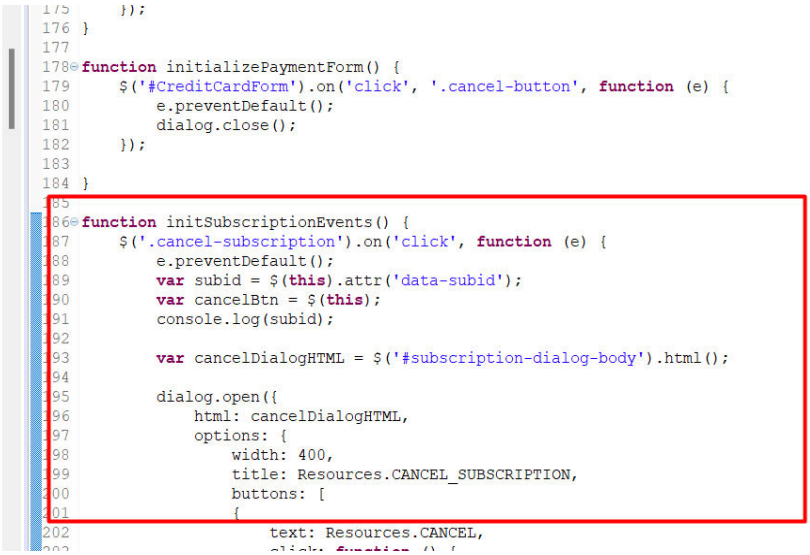
Figure 78 Modifications in Resource.de

4.5.1.19. js/pages/account.js

Add the following code block after initializePaymentForm() function:

```
function initSubscriptionEvents() {
    $('.cancel-subscription').on('click', function (e) {
        e.preventDefault();
            var subid = $(this).attr('data-subid');
            var cancelBtn = $(this);
            console.log(subid);

        var cancelDialogHTML = $('#subscription-dialog-body').html();

        dialog.open({
            html: cancelDialogHTML,
            options: {
                width: 400,
                title: Resources.CANCEL_SUBSCRIPTION,
                buttons: [
                    {
                    text: Resources.CANCEL,
                    click: function () {
                        $(this).dialog('close');
                    }
                },
                    {
                    text: Resources.OK,
                                click: function () {
                                    e.preventDefault();

                                    var url = cancelBtn.data('action');
                                    var subid = cancelBtn.data('subid');
```

```
                                    url = util.appendParamToURL(url, 'subid',
subid);


                                    $.ajax({
                                        url: url,
                                        type: 'get',
                                        dataType: 'json',
                                        success: function (data) {
                                            if (data.status.toLowerCase()
=== 'ok') {

        cancelBtn.prop('disabled', true);

        cancelBtn.closest('.order-history-header').find('.subscription-
status').text(data.statusMsg);

        cancelBtn.closest('.order-history-header').find('.subscription-
status').addClass('error');
                                                dialog.close();
                                            } else {
                                                dialog.close();
                                            }

                                            if (data.message) {

        window.alert(data.message);
                                            }

                                        },
                                        error: function (err) {
                                            page.refresh();
                                        }
                                    });
                                }
                            }]
                        }
                    });
                });
            }
```

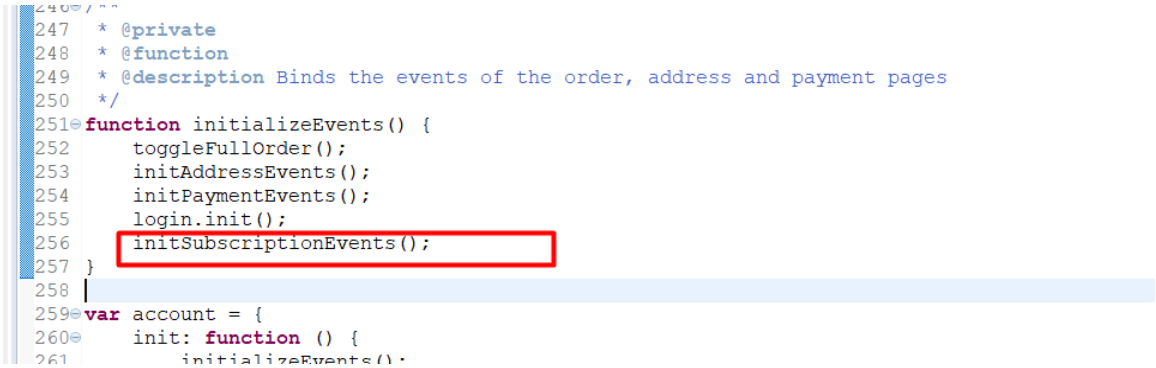


```
175     });
176 }
177
178 function initializePaymentForm() {
179     $('#CreditCardForm').on('click', '.cancel-button', function (e) {
180         e.preventDefault();
181         dialog.close();
182     });
183
184 }
185
186 function initSubscriptionEvents() {
187     $('.cancel-subscription').on('click', function (e) {
188         e.preventDefault();
189         var subid = $(this).attr('data-subid');
190         var cancelBtn = $(this);
191         console.log(subid);
192
193         var cancelDialogHTML = $('#subscription-dialog-body').html();
194
195         dialog.open({
196             html: cancelDialogHTML,
197             options: {
198                 width: 400,
199                 title: Resources.CANCEL_SUBSCRIPTION,
200                 buttons: [
201                 {
202                     text: Resources.CANCEL,
```

Figure 79 Modifications in account.js

Add following code in initializeEvents():

```
initSubscriptionEvents();
```



```
247  * @private
248  * @function
249  * @description Binds the events of the order, address and payment pages
250  */
251 function initializeEvents() {
252     toggleFullOrder();
253     initAddressEvents();
254     initPaymentEvents();
255     login.init();
256     initSubscriptionEvents();
257 }
258
259 var account = {
260     init: function () {
261         initializeEvents();
```

Figure 80 Modifications in account.js (cont.)

3.5.2. Controller Modification

If using a controller based SiteGenesis integration, additionally follow the instructions in this chapter.

3.5.2.1. COBilling.js

Go to COBilling.js controller, `returnToForm()` method and add the following code block:

```
try                                                                    {
require('int_klarna_payments_controllers/cartridge/controllers/KlarnaPayments.
js').CreateOrUpdateSession();
```

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

after `pageMeta.update()` function (see screen shot below)

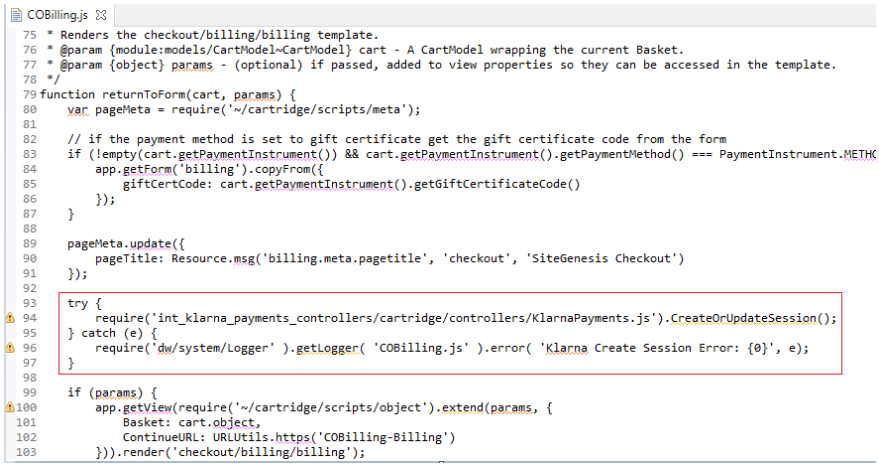


Figure 81 Modifications in COBilling.js

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));
            cart.removePaymentInstruments(cart.getPaymentInstruments('Klarna'));
        } else if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD_CRE
            app.getForm('billing').object.paymentMethods.bml.clearFormElement();

            cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD_BML));
            cart.removePaymentInstruments(cart.getPaymentInstruments('PayPal'));
            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'));
            cart.removePaymentInstruments(cart.getPaymentInstruments('Klarna'));
        }
        return true;
    });

```

Figure 82 Modifications in COBilling.js (cont.)

In method `handlePaymentSelection(cart)`, go to line 441 and add the following code block:

```
if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value !==
'Klarna') {
        require('int_klarna_payments_controllers/cartridge/controllers/KlarnaPayme
nts.js').CancelAuthorization();
}
```

```
        };
    }

    if (empty(PaymentMgr.getPaymentMethod(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value).paymentProcessor)) {
        result = {
            error: true,
            MissingPaymentProcessor: true
        };
    }

    if (!result) {
        if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value !== 'Klarna') {
            require('int_klarna_payments_controllers/cartridge/controllers/KlarnaPayments.js').CancelAuthorization();
        }
        result = app.getModel('PaymentProcessor').handle(cart.object, app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value);
    }
    return result;
}

/**
 * Gets or creates a billing address and copies it to the billingaddress form. Also sets the customer email address
```

Figure 83 Modifications in COBilling.js (cont.)

3.5.2.2. COSummary.js

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

```
try {
require('int_klarna_payments_controllers/cartridge/controllers/KlarnaPayments.
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)

Figure 84 Modifications in COSummary.js

3.5.2.3. 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/KlarnaPayments.js')
.PendingOrder(order);
    } 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)

OrderModel.js

```
/* API Includes */
var AbstractModel = require('./AbstractModel');
var Order = require('dw/order/Order');
var OrderMgr = require('dw/order/OrderMgr');
var Resource = require('dw/web/Resource');
var Status = require('dw/system/Status');
var Transaction = require('dw/system/Transaction');

/**
 * Place an order using OrderMgr. If order is placed successfully,
 * its status will be set as confirmed, and export status set to ready.
 * @param {dw.order.Order} order
 */
function placeOrder(order) {

    if(session.privacy.KlarnaPaymentsFraudStatus === 'PENDING') {
        try {
            require('int_klarna_payments_controllers/cartridge/controllers/KlarnaPayments.js').PendingOrder(order);
        } catch(e) {
            require('dw/system/Logger' ).getLogger( 'OrderModel.js' ).error( 'Klarna Payments Pending Order Error: {0}', e);
        }
        return;
    }

    var placeOrderStatus = OrderMgr.placeOrder(order);
    if (placeOrderStatus === Status.ERROR) {
        OrderMgr.failOrder(order);
        throw new Error('Failed to place order.');
    }
    order.setConfirmationStatus(Order.CONFIRMATION_STATUS_CONFIRMED);
    order.setExportStatus(Order.EXPORT_STATUS_READY);
}
```

Figure 85 Modifications in OrderModel.js

In placeOrder method add the following code at the end:

```
    if (session.privacy.customer_token) {
        var SubscriptionHelper = require('*/cartridge/scripts/subscription/subscriptionHelper');
        SubscriptionHelper.updateCustomerSubscriptionData(order);
    }
```

```
function placeOrder(order) {

    if (session.privacy.KlarnaPaymentsFraudStatus === 'PENDING') {
        try {
            require('int_klarna_payments_controllers/cartridge/controllers/KlarnaPayments.js').PendingOrder(order);
        } catch (e) {
            require('dw/system/Logger').getLogger('OrderModel.js').error('Klarna Payments Pending Order Error: {0}', e);
        }
        return;
    }

    var placeOrderStatus = OrderMgr.placeOrder(order);
    if (placeOrderStatus === Status.ERROR) {
        OrderMgr.failOrder(order);
        throw new Error('Failed to place order.');
    }
    order.setConfirmationStatus(Order.CONFIRMATION_STATUS_CONFIRMED);
    order.setExportStatus(Order.EXPORT_STATUS_READY);

    if (session.privacy.customer_token) {
        var SubscriptionHelper = require('*/cartridge/scripts/subscription/subscriptionHelper');
        SubscriptionHelper.updateCustomerSubscriptionData(order);
    }
}
/**
 * Order helper class providing enhanced order functionality.
```

Figure 86 Modifications in OrderModel.js (cont.)

3.5.2.4. CartModel.js

Add the following code in addProductListItem function:

```
var productLineItem = cart.createProductLineItem(productListItem, shipment);
productLineItem.setQuantityValue(quantity);
if (productLineItem) {
```

```
var isSubscriptionProduct =
productListItem.product.custom.kpIsSubscriptionProduct;

var isStandardProduct =
!empty(productListItem.product.custom.kpIsStandardProduct) ?
productListItem.product.custom.kpIsStandardProduct : true;

if (isSubscriptionProduct && !isStandardProduct) {

      productLineItem.custom.kpSubscription = true;

}
}
```

after shipment is assigned on line 153.

```
151             Transaction.wrap(function () {
152                 var shipment = cart.object.defaultShipment;
153                 var productLineItem = cart.createProductLineItem(productListItem, shipment);
154                 productLineItem.setQuantityValue(quantity);
155                 if (productLineItem) {
156                     var isSubscriptionProduct = productListItem.product.custom.kpIsSubscriptionProduct;
157                     var isStandardProduct = !empty(productListItem.product.custom.kpIsStandardProduct)
158                                             ? productListItem.product.custom.kpIsStandardProduct : true;
159                     if (isSubscriptionProduct && !isStandardProduct) {
160                         productLineItem.custom.kpSubscription = true;
161                     }
162                 }
163
164                 cart.calculate();
165             });
166         }
```

Figure 87 Modifications in CartModel.js

Add the following code in addProductItem function:

```
var isSubscriptionProduct = product.custom.kpIsSubscriptionProduct;

var isStandardProduct = !empty(product.custom.kpIsStandardProduct) ?
product.custom.kpIsStandardProduct : true;

if (isSubscriptionProduct && !isStandardProduct) {

      productLineItem.custom.kpSubscription = true;

}
```

on line 220.

```
213             } else {
214                 productLineItem = cart.createProductLineItem(product, productOptionModel, shipment);
215
216                 if (quantity) {
217                     productLineItem.setQuantityValue(quantity);
218                 }
219             }
220
221             var isSubscriptionProduct = product.custom.kpIsSubscriptionProduct;
222             var isStandardProduct = !empty(product.custom.kpIsStandardProduct) ? product.custom.kpIsStandardProduct : true;
223             if (isSubscriptionProduct && !isStandardProduct) {
224                 productLineItem.custom.kpSubscription = true;
225             }
226
227             /**
228              * By default, when a bundle is added to cart, all its child products are added too, but if those products are
229              * variants then the code must replace the master products with the selected variants that get passed in the
```

Figure 88 Modifications in Cartmodel.js (cont.)

3.5.2.5. Cart.js

Add following code before module exports section:

```
function updateSubscriptionAjax() {
```

```
    var productId = request.httpParameterMap.pid.stringValue;
    var subscription = request.httpParameterMap.subscription.stringValue ===
'true';
    var uuid = request.httpParameterMap.uuid.stringValue;
    var SubscriptionHelper =
require('*/cartridge/scripts/subscription/subscriptionHelper');

    let r = require('~/cartridge/scripts/util/Response');

    var cart = app.getModel('Cart').goc();

    if (!cart) {
        r.renderJSON({
            status: 'error',
            success: false,
        });
    }

    var matchingLineItem = cart.getProductLineItemByUUID(uuid);

    if (matchingLineItem) {
        Transaction.wrap(function () {
            matchingLineItem.custom.kpSubscription = subscription;
        });
    }

    if (matchingLineItem) {
        r.renderJSON({
            success: true,
            isSubscriptionBasket:
SubscriptionHelper.isSubscriptionBasket(cart.object)
        });
    } else {
        r.renderJSON({
            statusCode: 500,
            success: false,
            errorMessage: Resource.msg('error.cannot.update.product.subscription',
'subscription', null)
        });
    }
```

```
}
```

Add the following code before module exports section:

```
function updateSubscriptionDetailsAjax() {
    var SubscriptionHelper =
require('*/cartridge/scripts/subscription/subscriptionHelper');

    var selectedValue = request.httpParameterMap.selectedValue.stringValue;
    var subscriptionField =
request.httpParameterMap.subscriptionField.stringValue;

    let r = require('~/cartridge/scripts/util/Response');

    var cart = app.getModel('Cart').goc();

    if (!cart) {
        r.renderJSON({
            status: 'error',
            success: false,
        });
    }

    var updated = SubscriptionHelper.updateSubscriptionAttribute(cart.object,
subscriptionField, selectedValue);

    if (updated) {
        r.renderJSON({
            success: true
        });
    } else {
        r.renderJSON({
            status: 'error',
            success: false,
        });
    }
}
```

Add following code at the end of exposed methods section:

```
/** update subscription status
```

```
 * @see {@link module:controllers/Cart~updateSubscriptionAjax} */
exports.UpdateSubscription = guard.ensure(['https'], updateSubscriptionAjax);
/** update subscription details
 * @see {@link module:controllers/Cart~updateSubscriptionDetailsAjax} */
exports.UpdateSubscriptionDetails = guard.ensure(['https'],
updateSubscriptionDetailsAjax);
```

```
538  * @see {@link module:controllers/Cart~addBonusProductJson} */
539 exports.AddBonusProduct = guard.ensure(['post'], addBonusProductJson);
540 /** update subscription status
541  * @see {@link module:controllers/Cart~updateSubscriptionAjax} */
542 exports.UpdateSubscription = guard.ensure(['https'], updateSubscriptionAjax);
543 /** update subscription details
544  * @see {@link module:controllers/Cart~updateSubscriptionDetailsAjax} */
545 exports.UpdateSubscriptionDetails = guard.ensure(['https'], updateSubscriptionDetailsAjax);
546
```

Properties Problems Console

Figure 89 Modifications in Cart.js

3.5.3.6. COShipping.js

Add following code after cart calculation on line 100:

```
var isSubscriptionBasket =
require('*/cartridge/scripts/subscription/subscriptionHelper').isSubscriptionBaske
t(cart.object);
if (isSubscriptionBasket && !customer.authenticated) {
    session.privacy.guest_subscription_error = true;
    response.redirect(URLUtils.https('Cart-Show'));
    return;
}
```

```
100
101 var isSubscriptionBasket = require('*/cartridge/scripts/subscription/subscriptionHelper').isSubscriptionBasket(cart.object);
102 if (isSubscriptionBasket && !customer.authenticated) {
103     session.privacy.guest_subscription_error = true;
104     response.redirect(URLUtils.https('Cart-Show'));
105     return;
106 }
107 // Go to billing step, if we have no product line items, but only gift certificates in the basket, shipping is not required.
108 if (cart.getProductLineItems().size() === 0) {
109     app.getController('COBilling').Start();
110 } else {
```

Figure 90 COShipping.js modifications

Update prepareShipments function with param in attributes and pass it to cart model:

```
function prepareShipments(param) {
    var cart, homeDeliveries;
    cart = app.getModel('Cart').get(param);
```

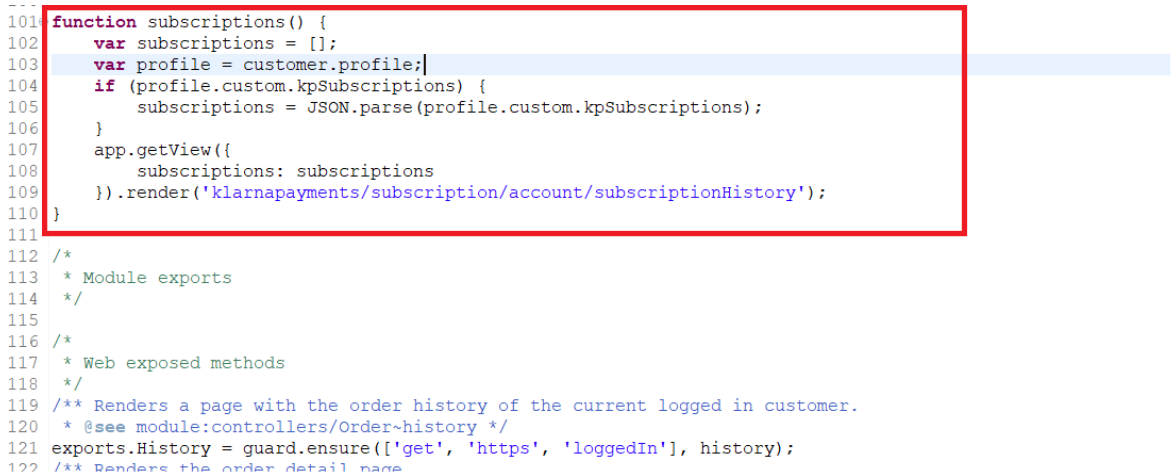
```
32      */
33      function prepareShipments(param) {          Rumyana Topalska,
34          var cart, homeDeliveries;
35          cart = app.getModel('Cart').get(param);
36
37          homeDeliveries = Transaction.wrap(function () {
38              var homeDeliveries = false;
39
40              cart.updateGiftCertificateShipments();
```

Figure 91 COShipping.js modifications (cont.)

4.5.3.7. Order.js

Add subscription function at the end of controller:

```
/**
 * Renders a page with all customer's subscriptions
 */
function subscriptions() {
    var subscriptions = [];
    var profile = customer.profile;
    if (profile.custom.kpSubscriptions) {
        subscriptions = JSON.parse(profile.custom.kpSubscriptions);
    }
    app.getView({
        subscriptions: subscriptions
    }).render('klarnapayments/subscription/account/subscriptionHistory');
}
```

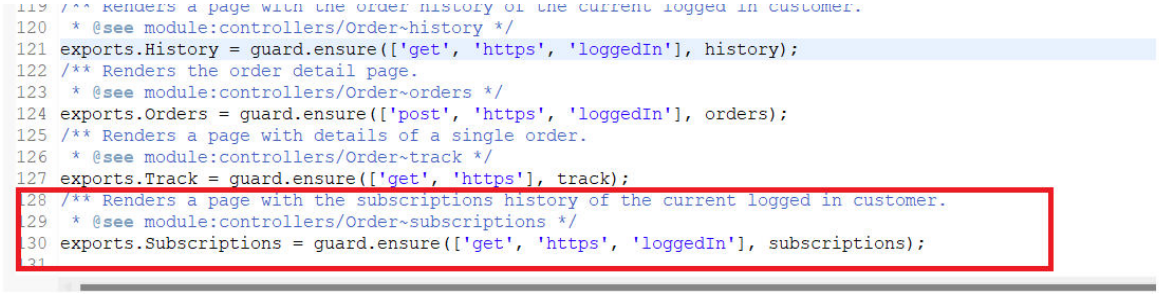


```
function subscriptions() {
    var subscriptions = [];
    var profile = customer.profile;
    if (profile.custom.kpSubscriptions) {
        subscriptions = JSON.parse(profile.custom.kpSubscriptions);
    }
    app.getView({
        subscriptions: subscriptions
    }).render('klarnapayments/subscription/account/subscriptionHistory');
}

/*
 * Module exports
 */

/*
 * Web exposed methods
 */
/** Renders a page with the order history of the current logged in customer.
 * @see module:controllers/Order~history */
exports.History = guard.ensure(['get', 'https', 'loggedIn'], history);
/** Renders the order detail page
```

Figure 92 Order.js modifications

Add module export for the same:

```
/** Renders a page with the subscriptions history of the current logged in customer.
 * @see module:controllers/Order~subscriptions */
exports.Subscriptions = guard.ensure(['get', 'https', 'loggedIn'], subscriptions);
```



```
 * @see module:controllers/Order~history */
exports.History = guard.ensure(['get', 'https', 'loggedIn'], history);
/** Renders the order detail page.
 * @see module:controllers/Order~orders */
exports.Orders = guard.ensure(['post', 'https', 'loggedIn'], orders);
/** Renders a page with details of a single order.
 * @see module:controllers/Order~track */
exports.Track = guard.ensure(['get', 'https'], track);
/** Renders a page with the subscriptions history of the current logged in customer.
 * @see module:controllers/Order~subscriptions */
exports.Subscriptions = guard.ensure(['get', 'https', 'loggedIn'], subscriptions);
```

Figure 93 Order.js modifications (cont.)

4.5.3.8. COCustomer.js

Add the following code on line 38:

```
    var SubscriptionHelper =
require('*/cartridge/scripts/subscription/subscriptionHelper');


    var subValidation =
SubscriptionHelper.validateCartProducts(Cart.goc().object);
    if (subValidation && subValidation.error) {
        response.redirect(URLUtils.https('Cart-Show'));
        return;
    }
```

```
SubscriptionHelper.updateCartSubscriptionDetails(Cart.goc().object);
```

after removeAllPayments() call

```
34    Transaction.wrap(function () {
35        Cart.goc().removeAllPaymentInstruments();
36    });
37
38    var SubscriptionHelper = require('*/cartridge/scripts/subscription/subscriptionHelper');
39
40    var subValidation = SubscriptionHelper.validateCartProducts(Cart.goc().object);
41    if (subValidation && subValidation.error) {
42        response.redirect(URLUtils.https('Cart-Show'));
43        return;
44    }
45
46    SubscriptionHelper.updateCartSubscriptionDetails(Cart.goc().object);
47
48    // Direct to first checkout step if already authenticated.
49    if (customer.authenticated) {
50        response.redirect(URLUtils.https('COShipping-Start'));
51        return;
52    } else {
53        var loginForm = app.getForm('login');
```

Figure 94 Modifications in COCustomer.js

4.5.3.9. COPlaceOrder.js

Update start() function with param in attributes:

```
function start(param) {
var cart = Cart.get(param);
if (!cart) {
app.getController('Cart').Show();
return {};
}
var COShipping = app.getController('COShipping');
// Clean shipments.
COShipping.PrepareShipments();
}
```

```
86  function start(param) {
87      var cart = Cart.get(param);
88
89      if (!cart) {
90          app.getController('Cart').Show();
91          return {};
92      }
93
94      var COShipping = app.getController('COShipping');
95
96      // Clean shipments.
97      COShipping.PrepareShipments();
98
99      // Make sure there is a valid shipping address, accounting for gift certificates that do not have one.
100     if (cart.getProductLineItems().size() > 0 && cart.getDefaultShipment().getShippingAddress() === null) {
101         COShipping.Start();
102         return {};
```

Figure 95 COPlaceOrder.js modifications

3.6. External Interfaces

All requests are done through Klarna's REST API and encrypted using SHA-256 with the shared secret provided by Klarna. Only HTTPS is allowed. JSON is used across all communications.

The full reference guide, along with the resource structure for requests & responses, can be found in the developer portal - https://docs.klarna.com/klarna-payments/api/

4. Testing

Klarna has a set of testing credentials and triggers that can be used.

Please, refer to the following URL: https://docs.klarna.com/resources/test-environment/

5. Operations, Maintenance

5.1. Data Storage

5.1.1. System Object Extensions

5.1.1.1. Basket

Parameter Name	Attribute ID	Description
Klarna Session ID	kpSessionId	The Klarna session ID returned after "Create Session" API endpoint is called. ***(Applicable since version 21.2.0)***
Klarna Client Token	kpClientToken	Client token returned by "Create Session" API endpoint and used to initialize the JS SDK.
Klarna Subscription Frequency	kpSubscriptionFrequency	Subscription frequency values (day, month, etc.)
Klarna Subscription Period	kpSubscriptionPeriod	Predefined subscription period in numbers

Table 1 Basket Attributes

5.1.1.2. Order

Parameter Name	Attribute ID	Description
Klarna Payments Order ID	kpOrderID	The Klarna payments Order ID for Klarna payment method selected by customer
VCN Brand	kpVCNBrand	Klarna Payments virtual card scheme name
VCN Holder	kpVCNHolder	Klarna Payments virtual card holder name
VCN Card ID	kpVCNCardID	Klarna Payments Virtual Card - Card ID
VCN PCI Data	kpVCNPCIData	Klarna Payments Virtual Card PCI Data in encrypted format
VCN Initialization Vector	kpVCNIV	Klarna Payments Virtual Card Initialization Vector
VCN AES Key	kpVCNAESKey	Klarna Payments Virtual Card AES Key
Is VCN Used	kpIsVCN	True if virtual card is enabled & used for payment of the order, otherwise false
Klarna Session ID	kpSessionId	The Klarna session ID returned after "Create Session" API endpoint is called ***(Applicable since version 21.2.0)***
Klarna Client Token	kpClientToken	Client token returned by "Create Session" API endpoint and used to initialize the JS SDK ***(Applicable since version 21.2.0)***
Klarna Subscription Frequency	kpSubscriptionFrequency	Subscription frequency values (day, month, etc.)
Klarna Subscription Period	kpSubscriptionPeriod	Predefined subscription period in numbers

Parameter Name	Attribute ID	Description
Klarna Authorization Token	kpAuthorizationToken	This attribute stores the Klarna authorization token, which is a string value used to authenticate and finalize the order creation process with Klarna's payment system. ***(Applicable since version 23.2.0)***
Klarna Redirect URL	kpRedirectURL	This attribute holds the URL to which the customer is redirected after the payment authorization is successfully completed by Klarna's system. ***(Applicable since version 23.2.0)***

Table 2 Order Attributes

5.1.1.3. Order Payment Instrument

Parameter Name	Attribute ID	Description
Klarna Payment Category ID	klarnaPaymentCategoryID	ID of Klarna payment category
Klarna Payment Category Name	klarnaPaymentCategoryName	Name of Klarna payment category

Table 3 Order Payment Instrument Attributes

5.1.1.4. Payment Transaction

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

Table 4 Payment Transaction Attributes

5.1.1.5. Site Preferences

The site custom preferences have been extended with a new group called "*Klarna_Payments*". The table below describes the preferences within that group:

Parameter Name	Attribute ID	Description
Auto-capture	kpAutoCapture	When enabled "Yes", a full order capture will be attempted automatically. The standalone order management API capture request will include total order amount value for "captured_amount". Default value is "No"
Klarna Payments Service Name	kpServiceName	The service name used for the current site
Send product_url and image_url	sendProductAndImageURLs	If set to true, product_url and image_url fields will be included in the Klarna session and order API calls. This enhances shopper experience post purchase.

Parameter Name	Attribute ID	Description
		Default value is “Yes”
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 vary. Note: Merchant Reference 1 value is always set to the SCC order ID
Border Color Preference	kpColorBorder	CSS (hex value) color set for Border in Klarna Payments iFrame
Border Selected Color Preference	kpColorBorderSelected	CSS (hex value) color set for selected element Border in Klarna Payments iFrame
Button Color Preference	kpColorButton	CSS (hex value) color set for Button in Klarna Payments iFrame
Button Text Color Preference	kpColorButtonText	CSS (hex value) color set for Button text in Klarna Payments iFrame
Checkbox Color Preference	kpColorCheckbox	CSS (hex value) color set for Checkbox in Klarna Payments iFrame
Checkbox Checkmark Color Preference	kpColorCheckboxCheckmark	CSS (hex value) color set for checkbox checked(selected) in Klarna Payments iFrame
Details Color Preference	kpColorDetails	CSS (hex value) color set for details in Klarna Payments iFrame
Header Color Preference	kpColorHeader	CSS (hex value) color set for Header in Klarna Payments iFrame
Rate limit By Operation	kpRateLimitByOperation	Select “Rate Limit By Operation” to Yes. If it is

Parameter Name	Attribute ID	Description
		selected to NO, the default service profile will be utilzied. The API rate limit for the Klarna service will the standard values as mentioned on docs.klanra.com. The default service id is klarna.http.defaultendpoint.
Klarna Payment Create New Session When Expires	kpCreateNewSessionWhenExpires	If set to Yes, then a new Klarna session will be created if Klarna session expires before SFCC basket expires
Link Color Preference	kpColorLink	CSS (hex value) color set for link in Klarna Payments iFrame
Text Color Preference	kpColorText	CSS (hex value) color set for text in Klarna Payments iFrame
Secondary Text Color Preference	kpColorTextSecondary	CSS (hex value) color set for secondary text in Klarna Payments iFrame
Border Radius Preference	kpRadiusBorder	Value (in pixels) of the border radius to be used in Klarna Payments iFrame
Attachments	kpAttachments	Toggle (Yes/No) for the inclusion of attachments when creating an order. Specific to inclusion of EMD (customer_account_info, other_delivery_address) when applicable. Default is “No”.
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"

Parameter Name	Attribute ID	Description
		} **Note: This is deprecated and will be removed in next releases!**
Virtual Card Number Enabled	kpVCNEnabled	If this option is set to “Yes”, Klarna settlement request will generate a Virtual Card Number for every Klarna order. Note: the option will only work if VCN private/public keys are configured properly as below and public key shared in advance with Klarna
VCN Public Key ID	kpVCNkeyId	UUIDv4 value corresponding to the key pair. Shared with Klarna respectively for Production & Playground (test) env.
VCN Private Key	vcnPrivateKey	SSL private key used only to decode Virtual Card information (used with kpVCNEnabled). Refer to section **9.3 Decrypt VCN Card Details**
VCN Public Key	vcnPublicKey	SSL public key used with Virtual Card integration (used with kpVCNEnabled). Shared with Klarna and stored here for reference.
VCN Settlement Retry Enabled	kpVCNRetryEnabled	If set to “Yes”, SFCC will retry the VCN settlement once again in case of service error. Default is “No”
Promotion Price Taxation	kpPromoTaxation	Only use "Based on Adjusted Price" value if you have enabled the corresponding value in “Merchant Tools >

Parameter Name	Attribute ID	Description
		Site Preferences > Promotions > Discount Taxation" and use gross taxation. Default: Based on Price
Hide Payment Methods on Deny	kpRejectedMethodDisplay	If set to value other than "No", the Klarna payment method options on the checkout will be greyed out or not displayed to customer in the current view when Klarna authorization request is rejected in the response (.i.e hard reject - "show_form" and "approved" values are both "false")
Alternative Klarna Payment Flow	kpUseAlternativePaymentFlow	If set to "Yes", Klarna Authorization and Order creation steps will be triggered on Checkout Review page when customer clicks CTA/Place Order Button. Default: No
Enable OMS	kpOMSEnabled	If set to "Yes", it updates the order information ingested in SFOMS (SalesForce Order Management System) in the format that Klarna expects. It is required when there is an integration between SFCC and SFOMS. Default: No
Klarna Payments Additional Logging	kpAdditionalLogging	If set to "Yes", it writes additional logging info to get more order details when issue occurs.

Parameter Name	Attribute ID	Description
		Default: No
Agent User Name	kpAgentUserName	User Agent Name for orders on behalf
Agent User Password	kpAgentUserPassword	User Agent Password for orders on behalf
Enable Retry for Recurring Orders	kpEnableRecurringOrderRetry	Enable retry for failed orders
Number Of Retries	kpRecurringNumberOfRetry	Recurring number of retries
Recurring Retry Frequency	kpRecurringRetryFrequency	Retry frequency for recurring orders
Klarna Create Order Token	kpCreateOrderToken	Token used to verify the caller for recurring orders
Use Bank Transfer callback	kpBankTransferCallback	If set to “Yes”, SFCC will wait for Klarna callback to place an order. Default is “No”

Table 5 Site Preferences

5.1.1.1. Product

Parameter Name	Attribute ID	Description
Is Klarna Standard Product	kpIsStandardProduct	Boolean attribute to define if the product is standard. ***(Applicable since version)***
Is Klarna Subscription Product	kpIsSubscriptionProduct	Boolean attribute to define if the product is eligible for subscription.
Klarna Trial Days Usage	kpTrialDaysUsage	Numeric value used for free trial definition.

5.1.1.2. ProductLineItem

Parameter Name	Attribute ID	Description
Is Selected for Subscription product	kpSubscription	Boolean attribute to define if the product is selected for subscription in the basket.
Enriched data for product line item.	klarna_oms__lineItemJSON	String data for product line item.

5.1.1.8. Profile

Parameter Name	Attribute ID	Description
Klarna Subscriptions	kpSubscriptions	Text attribute to store customer subscriptions to Klarna

5.1.2. Custom Objects

5.1.2.1. Klarna Express Button

The KEB button can be configured and adapted to the needs of your storefront. Steps to enable KEB button:

- To enable the button, your Klarna Merchant ID (MID) and domains must be white-listed. Please reach out to your Klarna delivery manager for support with this activity prior to testing and production go-live (as the domain names white-listed defer). Please ensure that the MID matches that which is configured in the BM service credentials for your storefront.
- To configure the KEB settings for a given site, you must visit "***Merchant Tools – Custom Object Editor***" and search for **KlarnaCountries** custom object.
- Choose the respective country key (e.g.: "US") and provide the below details:
    - Express Button Merchant ID: *Allow-listed Merchant ID*
    - Express Button environment: *Klarna environment – Production/Playground*
    - Express Button Cart Enabled: *Check box to display on cart page*

- MiniCart Express Button Enabled: *Check box to display on minicart page*
- MiniCart Express Button Theme/Shape: *theme/shape to display on minicart page*
- Express Button Library URL: Available here
- Express Button Category: *The Klarna category to be pre-selected on Checkout, e.g., "pay_over_time"*

- Please review **Fig 69** for reference.

More information to customize the button can be found here.

***Note**: Klarna Express Button is currently available in these markets with more to follow!*

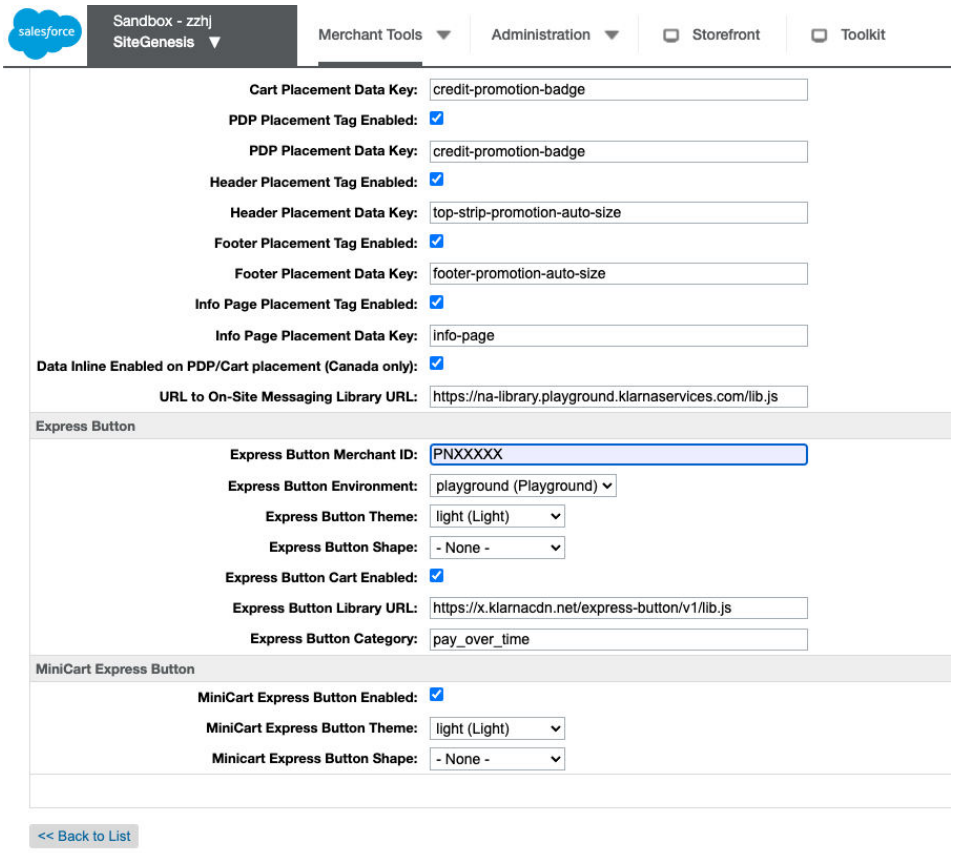


Figure 96 Klarna Express Button completed configuration

Shoppers experience on the storefront cart is as below:

1. Lands on cart page where KEB is displayed
2. Clicks on KEB button
3. Shopper prompted to share their Klarna registered credentials to complete authentication
4. Re-direct to checkout with Klarna Payment Method pre-selected

5.1.2.2. KlarnaCountries

The respective object is dynamically selected based on the request locale country, e.g., SFCC site with locale "**de_DE**" or "**en_DE**" will use the "**DE**" custom object. In cases when the request locale country can't be dynamically resolved (i.e. with "default" SFCC locale) – attribute "**klarnaLocale**" can be utilized to pass the proper locale to Klarna. For all other cases, this field can be left blank and will not be taken into consideration.

Even if you have locales that are not supported by Klarna Payments, we recommend creating a corresponding entry in the custom object for that locale. Thus, on the billing page of the unsupported locale you will have the Klarna Payments widget showing an appropriate message.

The custom objects store data such as Klarna default locale, service credential IDs and Klarna Payments placement data keys to ensure that Klarna Payments integration is correctly configured.

Note: The same custom object is used by Klarna Checkout cartridge integration!

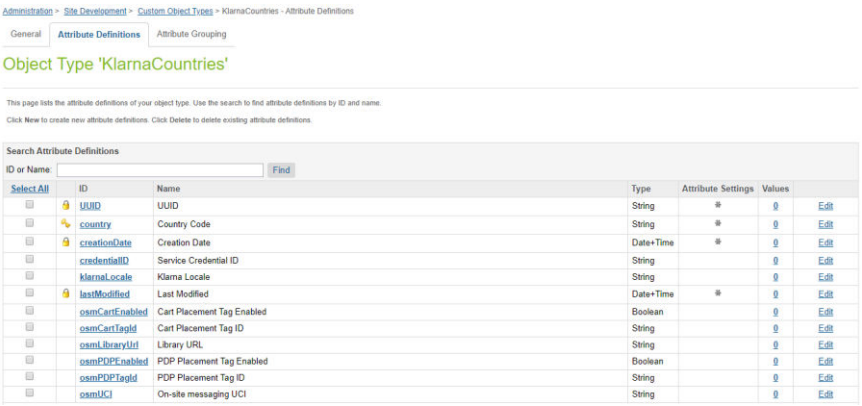


Figure 97 KlarnaCountries Attributes

The table below describes attributes of the **KlarnaCountries** custom object:

Attribute Name	Attribute ID	Description
Country Code	country	Two-letter country code
On-site Messaging Data Default Locale	klarnaLocale	Fallback, if the request locale can't be dynamically resolved, i.e., when using "default" SFCC locale
Service Credential ID	credentialID	The ID of service credentials for this locale.
On-site messaging Data Client ID	osmUCI	The Klarna On-site Messaging "data-client-id" applicable for a given country
Cart Placement Tag Enabled	osmCartEnabled	To enable Cart Placement for a given locale.
Cart Placement Tag ID	osmCartTagId	The Klarna On-site Messaging "data-key" of placement applicable for Cart Page for a given locale.
PDP Placement Tag Enabled	osmPDPEnabled	To enable PDP Placement for a given locale.
PDP Placement Tag ID	osmPDPTagId	The Klarna On-site Messaging "data-key" of placement applicable for Product Display page for a given locale.
Header Placement Tag Enabled	osmHeaderEnabled	To enable Klarna Header Placement in a given storefront
Header Placement Data Key	osmHeaderTagId	The Klarna On-site Messaging "data-key" of placement applicable for the Header
Footer Placement Tag Enabled	osmFooterEnabled	To enable Klarna footer Placement in a given storefront
Footer Placement Data Key	osmFooterTagId	The Klarna On-site Messaging "data-key" of placement applicable for the footer
Info Page Placement Tag Enabled	osmInfoPageEnabled	To enable Klarna Info Page Placement in a given storefront
Info Page Placement Data Key	osmInfoPageTagId	The Klarna On-site Messaging "data-key" of placement applicable for the Info Page
Data Inline Enabled on PDP/Cart placement (Canada only)	osmDataInlineEnabled	Enable this when using PayBright payment method in Canada
URL to On-Site Messaging Library URL	osmLibraryUrl	URL for On-Site Messaging library, applicable for testing or production must be saved. Please use only Klarna production URL in live storefront. Verify test environment URL which includes "playground" in URL For production or live environment, ensure URL includes

Attribute Name	Attribute ID	Description
		"production". E.g: Test URL> https://na-library.playground.klarnaservices.com/lib.js , e.g: Live URL>https://na-library.production.klarnaservices.com/lib.js
Express Button Environment	kebEnvironment	The express button environment. Default is playground
Express Button Merchant ID	kebMerchantID	The merchant ID used to display the button
Express Button Library URL	kebLibraryUrl	URL of the express button library
Express Button Cart Enabled	kebCartEnabled	To enable the Klarna Express Button on cart page
Express Button Theme	kebTheme	The theme of the button. Options include default, light & dark
Express Button Category	kebCategory	The Klarna category to be pre-selected on Checkout Page, e.g., "pay_over_time"

Table 6 KlarnaCountries Attributes

***Note**: The data-client-id and data-key values used in the OSM placements are available in the Klarna Merchant Portal (Europe/US (CA included)/Oceania) within the On-site Messaging App. When selecting the data-key values, ensure that the filter is set to the right country and language.*

5.1.3. Session Attributes & Cookies

The following session custom attributes are saved in "**session.privacy**" storage and accessible in checkout. The attributes are retained for the session lifetime & cleared when the customer logs out of their profile.

Attribute	Description
KlarnaLocale	The Klarna locale in use
KlarnaPaymentsSessionID ***(Not applicable since version 21.2.0)***	The Klarna session ID returned after "Create Session" API endpoint is called
KlarnaPaymentsClientToken ***(Not included as session attributes***	Client token returned by "Create Session" API endpoint and

Attribute	Description
since version 21.2.0)	used to initialize the JS SDK
KlarnaPaymentMethods	The available payment method categories for the respective Klarna session; Saved in JSON format
SelectedKlarnaPaymentMethod	The selected payment method on checkout (e.g., “pay_later”)
KlarnaPaymentsAuthorizationToken	The authorization token returned by JS SDK “Authorize” call
KlarnaPaymentsFinalizeRequired	Whether finalization is required for the payment method; Returned by JS SDK “Authorize” call
KlarnaExpressCategory	The KEB payment category; Currently applicable for US and defaults to “pay_over_time”
KlarnaPaymentsOrderID	The Klarna order ID returned by the “Create Order” API call
KlarnaPaymentsRedirectURL	The URL to redirect the customer to after placing the order; Returned by the “Create Order” API call
KlarnaPaymentsFraudStatus	The fraud status of the order returned by the “Create Order” API call

Table 7 Klarna Session Attributes

The following cookies are being set by Klarna integration:

Cookie Name	Description
selectedKlarnaPaymentCategory	The selected payment method on checkout (e.g. “pay_later”)

Table 8 Klarna Cookies

5.1.4. Library

In addition to the configurations, the following 2 library assets will be added:

- “**footer-about**” – Updated OOTB asset including link to Klarna OSM dedicated page in the footer.
- “**klarna-email-info**” – Asset containing links to review the Klarna Payment information. Used in the confirmation email sent to the customers.

5.1.5. Services

An HTTP service “***klarna.http.defaultendpoint***” has been added with “**klarna.http.service**” profile and service credentials for each country (described in **KlarnaCountries** custom object).

---

*Please, note that up until version **21.2.0** of the cartridge, the **KlarnaCountries** custom object was **replicable**. To avoid issues with service credentials during replication, merchants should use the same service credential name in staging, development, and production environments!*

*Please, review section **9.4 Update KlarnaCountries Definition** on possible ways to update the definition in your instances.*

5.2. Logs

The integration includes the following logs:

- Service communication logs – starts with “service-klarna-***”. These logs contain every request and response to the Klarna endpoints. Personal information, i.e. emails & names required for the Klarna API calls are masked in the logs.
- Custom errors and debug info are logged under “customerror-***”, “custodebug-***” & “custominfo-***” files depending on the case.

5.3. Availability

Cartridge functionality will be dependent on the availability of the Klarna API service. Current Klarna operational status can be viewed here - http://status.klarna.com/

5.4. Failover/Recovery Process

If Klarna API is not available, Klarna is not presented as a payment option. In case of any failure within the Klarna API, contact Klarna for support.

5.5. Support

Klarna's service center is responsible for handling operational tasks. The service center is divided into the following two teams: Customer Service and Merchant Support.

5.5.1. Customer Service

A customer service workshop can be conducted during the implementation process before going live to align the operational processes and ensure customer satisfaction. Klarna provides all customers with the possibility to log into Klarna App via website: https://app.klarna.com/login or download the Klarna App (free) on a mobile (Android/iOS). The customers can contact support, view their statements, pay for their purchase, track delivery updates, and prolong the due dates if they have chosen to pay after delivery.

5.5.2. Merchant Support

Reporting core SFCC functionality issues in the Klarna cartridge technical integration – please contact commercecloud@klarna.com

For production issue related to Klarna API availability, merchant representative should reach their Klarna Account manager after reviewing the current operational status at http://status.klarna.com/. Report the problem in Production (Post Go-live) if you have a suspicion about degraded performances or issues with Klarna's service. The Klarna contact would then be able to report this internally to the incident management team who have established routines to handle and resolve

reported incidents. The Klarna contact may request additional information from the individual reporting the problem to help internal team ascertain and identify the issue. The KAM may also advice the merchant to follow the updates on the status page if it is a known incident with on-going updates.

Pre-requisite information to be provided by merchant when reporting incident to help with speedy investigation and resolution:

- Merchant's affected MID or market
- Impact and examples of customer orders (order_id or Klarna session_id if available)
- Screenshots, timeframe, additional information as required

6. User Guide

6.1. Roles, Responsibilities

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

6.2. Storefront Functionality

When Klarna has been setup, the Klarna Payments options and iframe widgets will be shown on the billing step. All the SFCC 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 one of Klarna's payment options as the payment method on billing step of checkout process and click the “Next: Place Order” button:

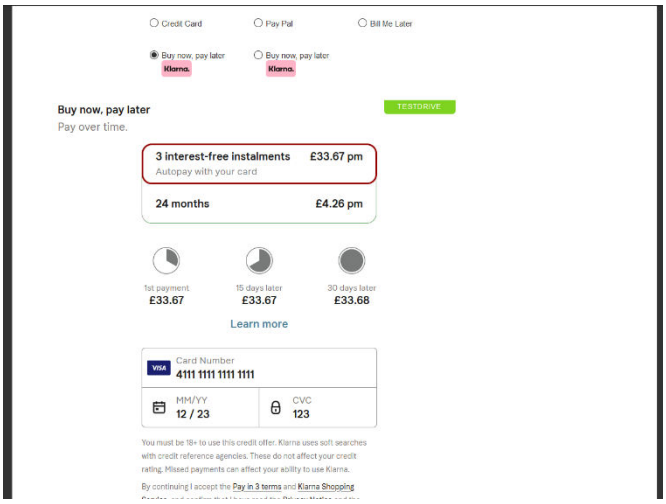


Figure 98 Payment Options on Checkout

Depending on the payment method selected and the region, you will see one of Klarna's popup windows to provide the details. Follow the steps on the screen:

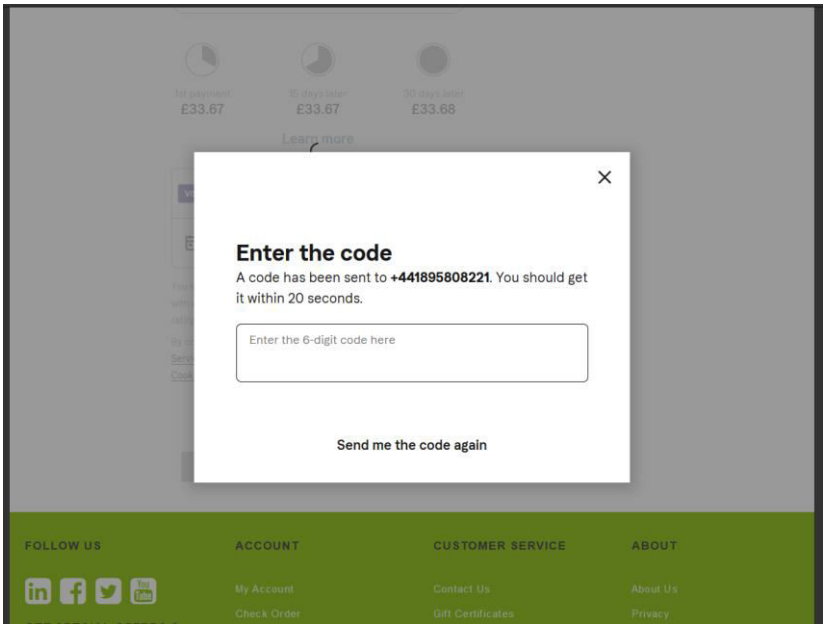


Figure 99 Klarna Popup Screen

On the Review step click on "Place Order" button:

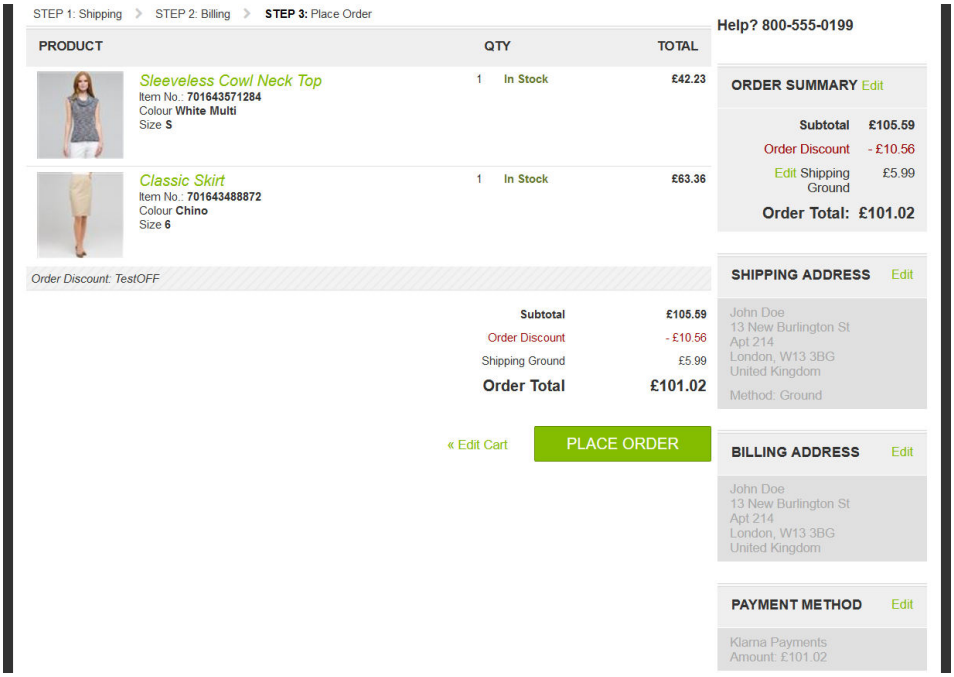


Figure 100 Payment Review Screen

The customer's browser is sent to the "**redirect_url**" and immediately thereafter shown the Commerce Cloud Order Confirmation page.

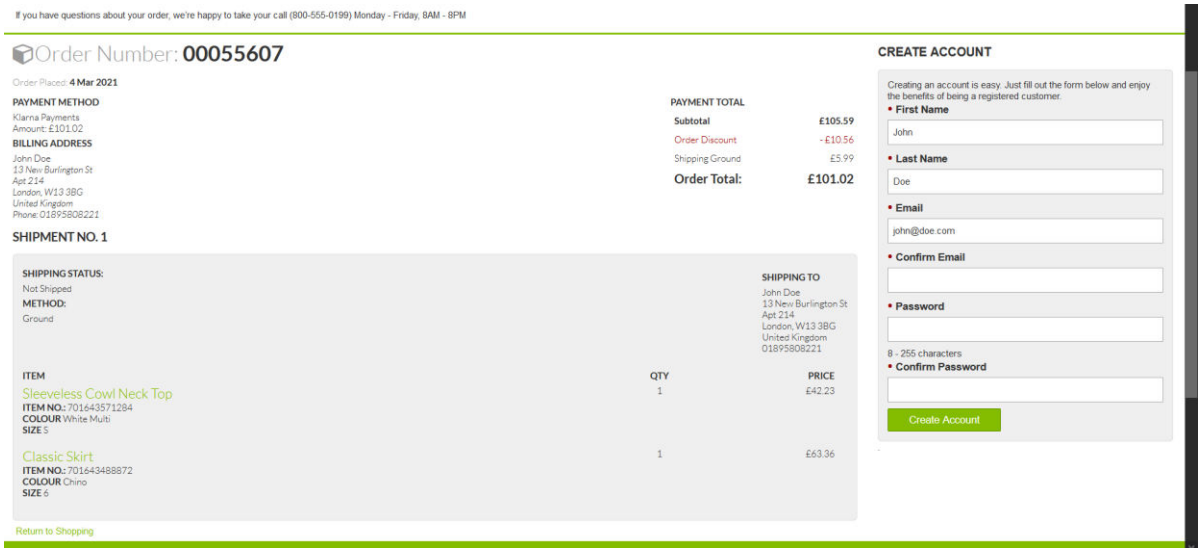


Figure 101 Order Confirmation Page

The newly created order can be the inspected in Business Manager:

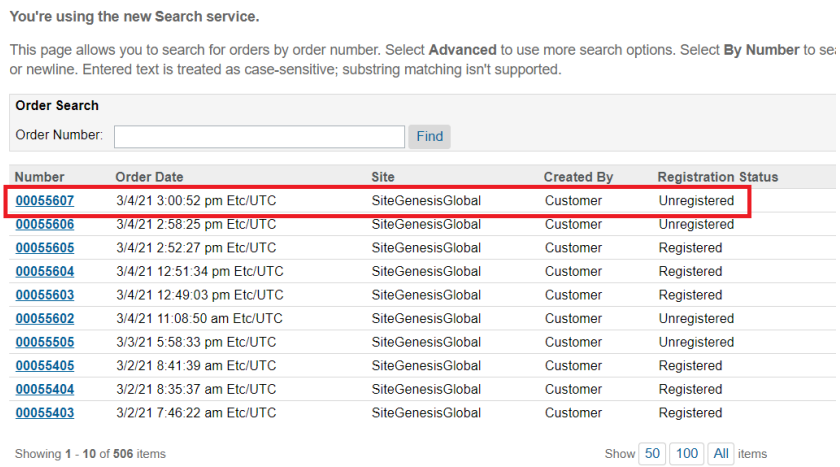


Figure 102 Orders List in BM

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

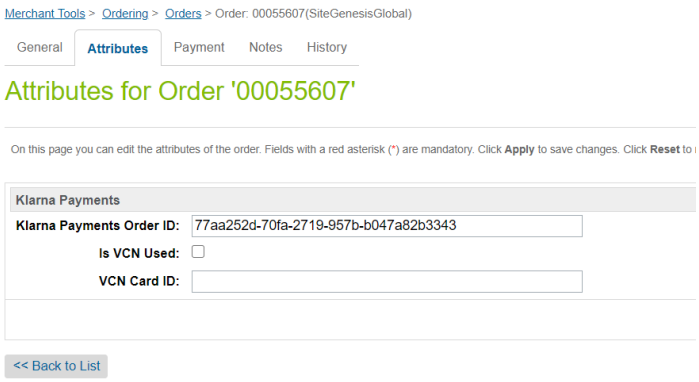


Figure 103 Order Attributes

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

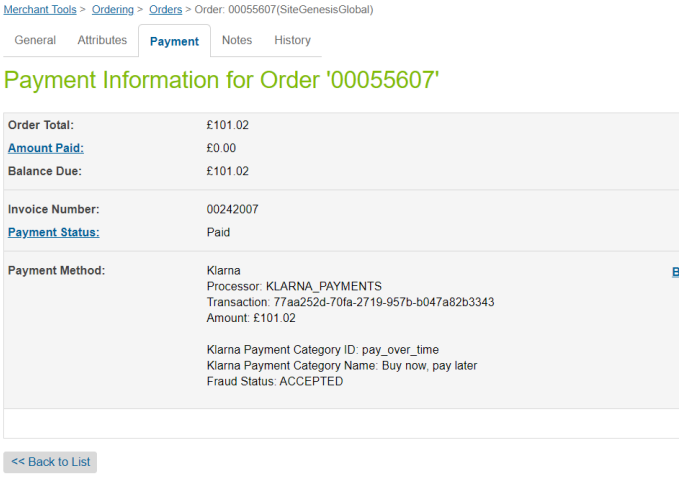


Figure 104 Order Payment Detail

Order can be further inspected in Klarna Merchant Portal:

- EU: eu.portal.klarna.com
- US: us.portal.klarna.com
- OC: us.portal.klarna.com

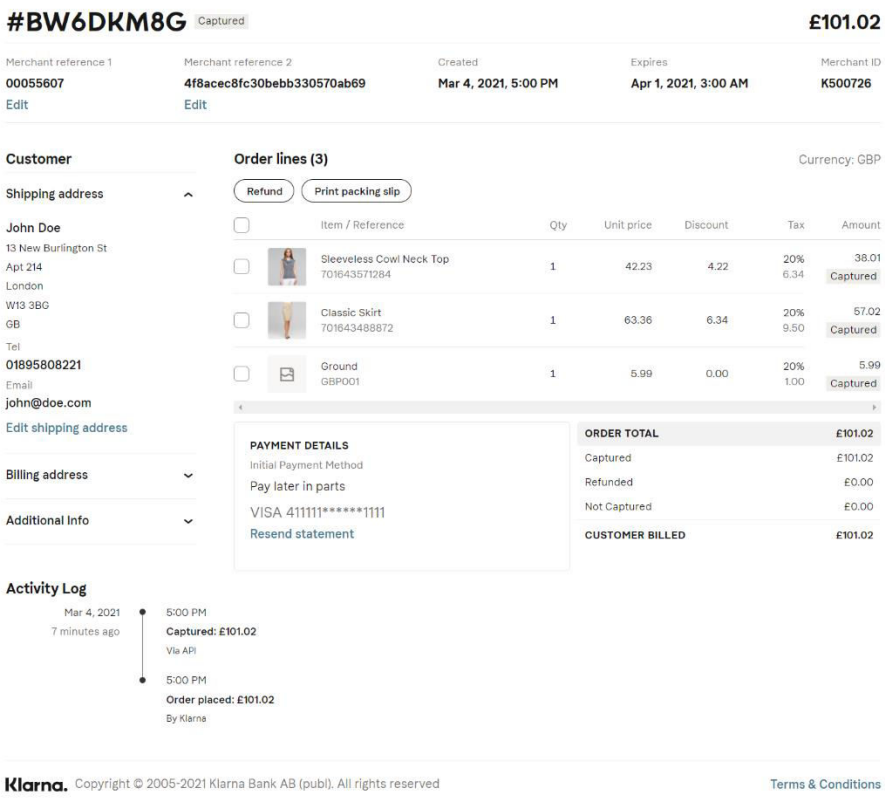


Figure 105 Klarna Portal Order View

7. Known Issues

The LINK Cartridge has no known issues.

8. Release History

Version	Date	Changes
18.1.0		Initial release of Klarna Payments
19.1.0		Added SFRA version
19.1.1		Updated VCN to use the newest API version
19.1.2		Fix auto capture for the pipelines cartridge
19.1.4		New country locales added. Minor bug fixes. Cartridge templates and forms updated for latest SFRA.
19.1.5		Added additional verification for all notifications. Minor fixes around the configuration objects. Added Canadian support. Documentation updates.
19.1.6		New country locales added. Updated VCN to store encrypted card details
21.1.0		Fixes around discounts taxation & VCN error handling. Added VCN improvements, additional On-site Messaging placements, BOPIS support. New IT, CA, FR & NZ country locales. Removed acknowledge call. Documentation updates.
21.1.1		New On-Site Messaging setting for Canada. Remove not required locale templates for SG Spain & Belgium. Documentation updates.
21.1.2		Fixed core file naming convention issues in 21.1.0 and 21.1.1. Please upgrade to the latest version if you are currently using 21.1.0 or 21.1.1. Removed deprecated “scripts/util/Builder.js” file.
21.2.0		Added Klarna Express Button.

Version	Date	Changes
		Moved Klarna session ID & client token from SFCC session privacy to Basket attributes. Changed KlarnaCountries definition to not replicable. Code cleanup. Documentation updates.
21.3.0		Improvements for create_session errors Expired user session issues related to empty shipment.shippingMethod Additional locale (PL) included in config files
21.3.1		Documentation updates.
22.1.0		Improvements for create and update session errors Add Klarna Express Button in mini-cart Support for long running basket Rate-limits by operations
22.2.0		One Klarna Optimisation Mexico locale support
22.2.1		Rollback of One Klarna Optimisation
22.3.0		Rollback hide VAT from Checkout functionality
22.3.1		Fix User-agent version sent to Klarna services
22.4.0		Intent field addition in Klarna Payment session creation Combine Klarna Authorization and Create Order in Checkout Review Step
22.5.0		OMS support
23.1.0		Fix issue with incorrect values for EMD Improvement Klarna Auto Capture and error handling Logging information for troubleshooting bugs

Version	Date	Changes
		Add Auto_finalise=True to the review checkout flow
23.1.1		Fixed an issue where sessions with negative order_tax_amount occurred due to SFCC session expiration Compatibility mode 21.2 support Replace deprecated window.KlarnaOnsiteService.push with window.Klarna.OnsiteMessaging.refresh.
23.2.0		Subscription Payments support: recurring payments and subscription handling directly within the SFCC environment. This update includes configuration options, subscription management in the cart and checkout pages, and a customer dashboard for subscription oversight. Klarna Bank Transfer payments: added a new server-side authorization callback feature for Klarna Bank Transfer payments, enhancing reliability across EU markets and supporting all existing KP cartridge functionalities.

9. Additional Information

9.1. Klarna API Information

The Klarna Payments API is accessible through different endpoint based on the context of the webstore. There are separate endpoints for testing and live and the Klarna merchant identifier (MID) is configured for respective markets in regions (EU, NA, OC) by endpoint.

9.1.1. Live Environment

The API for the European production environment can be found at

- https://api.klarna.com/

The API for the North America production environment can be found at

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

The API for the Oceania production environment can be found at

- https://api-oc.klarna.com/

9.1.2. Testing Environment

The API for the European Playground/testing environment can be found at

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

The API for the North America Playground/testing environment can be found at

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

The API for the Oceania Playground/testing environment can be found at

- https://api-oc.playground.klarna.com/

9.2. Generate Key Pair and Key Id for Virtual Card Settlements (VCN)

The recommend RSA keypair size of 4096 bits. This key pair must be associated with a key_id (UUIDv4). The public key must be shared in JWK format with Klarna contact. Note that for production and playground, the key_id and keypair combination shared are different and must be configured prior to testing/go-live of the virtual card product.

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
```

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
tV2281w3UE9WO4oxfknJtKEdHn84x55ULt8KQTh9NVtdeKC8nTfTgyvMt/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-----
```

9.3. Decrypt VCN Card Details

To decrypt the virtual card details stored on order level and authorize the credit card processor you can use the following code snippet. You can find more information about the decryption process here.

```
var OrderMgr = require( 'dw/order/OrderMgr' );
var Cipher = require( 'dw/crypto/Cipher' );
var Encoding = require( 'dw/crypto/Encoding' );
var Site = require( 'dw/system/Site' );

var Order = OrderMgr.getOrder( "order_id" );
var VCNPrivateKey = Site.getCurrent().getCustomPreferenceValue(
'vcnPrivateKey' );
var cipher = new Cipher();

var keyEncryptedBase64 = Order.custom.kpVCNAESKey;
var keyEncryptedBytes = Encoding.fromBase64( keyEncryptedBase64 );
var keyDecrypted = cipher.decryptBytes( keyEncryptedBytes, VCNPrivateKey,
"RSA/ECB/PKCS1PADDING", null, 0 );
var keyDecryptedBase64 = Encoding.toBase64( keyDecrypted );
var cardDataEncryptedBase64 = Order.custom.kpVCNPCIData;
var cardDataEncryptedBytes = Encoding.fromBase64( cardDataEncryptedBase64 );
var cardDecrypted = cipher.decryptBytes( cardDataEncryptedBytes,
keyDecryptedBase64, "AES/CTR/NoPadding", Order.custom.kpVCNIV, 0 );

var cardDecryptedUtf8 = decodeURIComponent( cardDecrypted );
var cardObj = JSON.parse( cardDecryptedUtf8 );
var expiryDateArr = cardObj.expiry_date.split( "/" );

// Retrieve ecnrypted card details
var cardPAN = cardObj.pan, cardCVV = cardObj.cvv,
    cardExpiryMonth = expiryDateArr[0], cardExpiryYear = expiryDateArr[1];
```

9.4. Update KlarnaCountries Definition

With version 21.2.0 of the cartridge, the KlarnaCountries custom object definition changed to non-replicable. If you are using an earlier version, to mitigate any issues that may be present in your environments, please follow these steps to update the definitions:

5. Back-up KlarnaCountries configurations per country for each site as changing the definition will remove all configurations on your environment!
    a. For merchant with one site - Go to "**Merchant Tools > Custom Objects > Import & Export**", select KlarnaCountries & export the data.
    b. For merchant with multiple sites using Klarna – You can export the configurations via "**Admin > Site Development > Site Import & Export**" and select each site that uses Klarna to be included in the export zip file. This section requires Account Manager access for users and will export all custom objects, not just Klarna ones.
6. Export custom object definitions from "**Admin > Site Development > Import & Export**". This action will export all custom object definitions that you have.
7. Update KlarnaCountries.xml definitions in file exported in step 2 and set **"<staging-mode>source-to-target</staging-mode>**" to **"<staging-mode>no-staging</staging-mode>**"
8. Import the updated KlarnaCountries.xml definition in "**Admin > Site Development > Import & Export**".
9. Import the KlarnaCountries configs exported in step 1 (manually or via bulk import)
    a. If you've followed step 1.b with multiple sites, you may want to edit the zip file and remove everything else apart from KlarnaCountries.xml