Klarna for Salesforce Commerce Cloud

Version 24.4.0 SiteGenesis

Klarna

Contents

Contents 2
1. Summary 6
2. Klarna Payments 7
2.1. Key features 7
2.2. Klarna Payments cartridge integration 8
2.3. Locales 9
2.4. Use Cases 10
2.4.1. Multiple payment options 10
2.4.2. Klarna Display Conditions and Authorization Handling 11
2.4.3. Authorizing and Placing Klarna Orders in Checkout 12
2.4.4. Refusal of Klarna Payments on Authorization 14
2.4.5. Klarna Payment Option Not Available for Current Purchase - Billing Page 15
2.4.6. Handling Unavailability of Klarna API or Inapplicable Storefronts 16
2.4.7. Notification Handling and Fraud Status Updates 16
2.4.8. Virtual Card Settlements 18
2.4.9. Enabling and Managing Auto-Capture for Payments 24
2.4.10. Widget Customizations 25
2.4.11. Customizing Payment Method Name 26
2.5. Advanced features 28
2.5.1. Klarna Payment Method Based Promotions 28
2.5.2. Price Adjustment Taxation Handling 28
2.5.3. Buy Online, Pickup in Store (BOPIS) 30
2.5.4. Service Rate Limits 31
2.5.5. Klarna Subscriptions 32
2.5.5.1. Configuration 32
2.5.5.2. Cart Page 32
2.5.5.3. Checkout 33
2.5.6. Account Subscription Dashboard 34
2.5.7. Recurring Subscription Order Creation 35
2.6. Klarna Express Button (KEB) 35
2.7. Compatibility 36
2.8. Privacy and Payment 36
2.8.1. GDPR Compliance 36
2.8.2. EMD (Extra Merchant Data) 36

2.8.3. PCI-DSS Compliance 41
3. Conversion Boosters 42
3.1. Klarna On-site Messaging 42
3.1.1. Configuration 42
3.1.1.1. Configuring OSM via KlarnaCountries custom object 42
3.1.1.2. Configuring OSM via site preferences 44
3.2. Klarna Express Checkout 48
3.2.1. Configuration 48
3.2.2. Placements 49
4. Implementation Guide 50
4.1. Setup of Business Manager 50
4.1.1. Cartridge Upload & Assignment 50
4.1.2. Metadata Import 51
4.2. Configuration 52
4.2.1. Configure Klarna Activation Site Preferences for single Klarna API credentials per site 52
4.2.2. Configure Klarna Activation Custom Object for multiple Klarna API credentials on one site 53
4.2.3. Add Account Settings to KlarnaCountries Custom Objects 54
4.2.4. Configure Klarna Payment Custom Preferences 55
4.2.5. Configure Klarna Payment Service 56
4.2.6. Configure Klarna Rate Limited Service Profile 57
4.2.7. Configure Custom Rate Limits 57
4.3. Extended Controllers 58
4.4. Template Updates 59
4.5. Jobs 59
4.5.1. Job “OrderCleanUp” (Optional) 59
4.5.2. Job “RecurringOrders” 63
4.6. Custom Code 64
4.6.1. Template modifications 65
4.6.1.1. default/checkout/summary/summary.isml 65
4.6.1.2. default/checkout/billing/billing.isml 68
4.6.1.3. default/checkout/billing/paymentmethods.isml 68
4.6.1.4. default/checkout/shipping/minishipments.isml 70
4.6.1.5. default/components/header/header.isml 71
4.6.1.6. default/components/footer/footer.isml 71
4.6.1.7. default/components/footer/footer_UI.isml 72
4.6.1.8. default/product/producttopcontentPS.isml 73
4.6.1.9. default/product/productcontent.isml 74
4.6.1.10. default/product/productcontent.isml 75

4.6.1.11. default/checkout/cart/cart.isml 79
4.6.1.12. default/mail/orderconfirmation.isml 80
4.6.1.13. default/components/order/ordrdetailsemail.isml 81
4.6.1.14. default/checkout/cart/minicart.isml 81
4.6.1.15. js/pages/cart.js 83
4.6.1.16. default/checkout/shipping/singleshipping.isml 86
4.6.1.17. scripts/cart/ValidateCartForCheckout.js 87
4.6.1.18. scripts/util/Resource.ds 88
4.6.1.19. js/pages/account.js 89
4.6.1.20. default/account/orders.isml 92
4.6.1.21. default/checkout/components/minicheckout_address.isml 92
4.6.1.22. js/minicart.js 93
4.6.2. Controller modification 94
4.6.2.1. COBilling.js 94
4.6.2.2. COSummary.js 96
4.6.2.3. OrderModel.js 97
4.6.2.4. CartModel.js 99
4.6.2.5. Cart.js 100
4.6.2.6. COShipping.js 103
4.6.2.7. Order.js 103
4.6.2.8. COCustomer.js 105
4.6.2.9. COPlaceOrder.js 106
4.7. External interfaces 107
5. Testing 108
6. Operations and Maintenance 109
6.1. Data Storage 109
6.1.1. System Object Extensions 109
6.1.1.1. Basket 109
6.1.1.2. Order 110
6.1.1.3. Order Payment Instrument 111
6.1.1.4. Payment Transaction 112
6.1.1.5. Site Preferences 112
6.1.1.6. Product 120
6.1.1.7. ProductLineItem 121
6.1.1.8. Profile 121
6.1.2. Custom Objects 122
6.1.2.1. Klarna Express Button 122
6.1.2.2. KlarnaCountries 123
6.1.2.3. Klarna Activation 129

6.1.3. Session Attributes & Cookies 130
6.1.4. Library 132
6.1.5. Services 132
6.2. Logs 133
6.3. Availability 133
6.4. Failover/Recovery Process 133
6.5. Support 133
6.5.1. Merchant Support 134
7. User Guide 135
7.1. Cartridge Upgrade 135
7.1.1. Upgrade Process 135
7.2. Roles and Responsibilities 136
7.3. Storefront Functionality 136
8. Release History 142
8.1. Known issues 148
9. Additional Information 149
9.1. Klarna API Information 149
9.1.1. Live Environment 149
9.1.2. Testing Environment 149
9.2. Generate Key Pair and Key ID for Virtual Card Settlements (VCN) 150
9.3. Decrypt VCN Card Details 152
9.4. Update KlarnaCountries Definition 155

1. Summary

The Klarna cartridge seamlessly integrates with Salesforce Commerce Cloud Storefront, allowing merchants to offer a variety of Klarna products. This guide is tailored for developers, providing detailed instructions on installing and integrating the cartridge into a Salesforce Commerce Cloud site, ensuring full compatibility with the SiteGenesis JavaScript Controllers (SGJS).

Key features:

- **Klarna Payments**: Offers a variety of payment options at checkout, including Pay Now, Pay Later, and Pay in Installments.
- **Express Checkout**: Enhances conversion rates by simplifying and speeding up the checkout process.
- **On-site Messaging**: Promotes the availability of Klarna's flexible payment methods throughout the shopping experience.

Integration components:

- **Cartridges**: The integration involves two cartridges - `int_klarna_payments` and `int_klarna_payments_controllers`.
- **Site-template Archive**: Includes new attributes and settings necessary for the integration.
- **Documentation**: This SiteGenesis integration guide provides comprehensive instructions.

To enable Klarna as a payment method, merchants need to configure the cartridge using valid credentials and site configurations within the Commerce Cloud Business Manager. The integration is designed based on the SiteGenesis demo store provided by Commerce Cloud. Merchants are required to sign a contract with Klarna to receive integration support and go-live in production. Klarna provides a playground environment for testing the integration before switching to the production environment. Additionally, Klarna offers assistance with integration and testing prior to the go-live sign-off.

2. Klarna Payments

Klarna Payments enables merchants to integrate flexible payment options like Pay Now, Pay Later, and Financing. It supports easy configuration through the Business Manager, enhancing the checkout experience and boosting conversion rates. Advanced features include virtual card settlements and streamlined fraud management for secure and efficient transactions.

2.1. Key features

Integrate Klarna Payments using Best Practices

- Available for international markets including North America, Europe, and Oceania.
- Klarna Payments can be configured independently on each site by region.
- Supports multiple Klarna payment methods: Pay Now, Pay Later, and Pay Over Time.
- Quick integration with a virtual card-based approach for settlement.
- Compliant with GDPR (EU) standards for checkout flow.
- Supports multi-shipping addresses.

Payment and Notification Handling

- Handles notifications for pending status updates (`reject`/`accept`) for suspected orders after review.
- Supports Klarna authorization with finalization for bank transfer methods (Pay Now).
- Supports Auto-Capture of payments.

Advanced Features

- BOPIS (Buy Online, Pickup in Store) support, including additional merchant data.
- Supports Klarna payment method-based promotions.
- Supports adjusted price promotions under the Gross Tax Policy.

2.2. Klarna Payments cartridge integration

The Klarna cartridge utilizes the Klarna Payments JSON REST API and JavaScript SDK for storefront integration. This integration displays various Klarna payment options via a widget (iframe) embedded on the billing page, known as the Klarna widget.

- **Customer Interaction:** Customers select Klarna as their payment method, review the payment terms, and authorize payment by clicking the "Place Order" button.
- **Order Creation:** Upon authorization, a Klarna order is created, and the customer is redirected to the confirmation page.
- **Fraud Status Management:**
    - Orders with a fraud status of `ACCEPTED` proceed with standard order creation in the SCC.

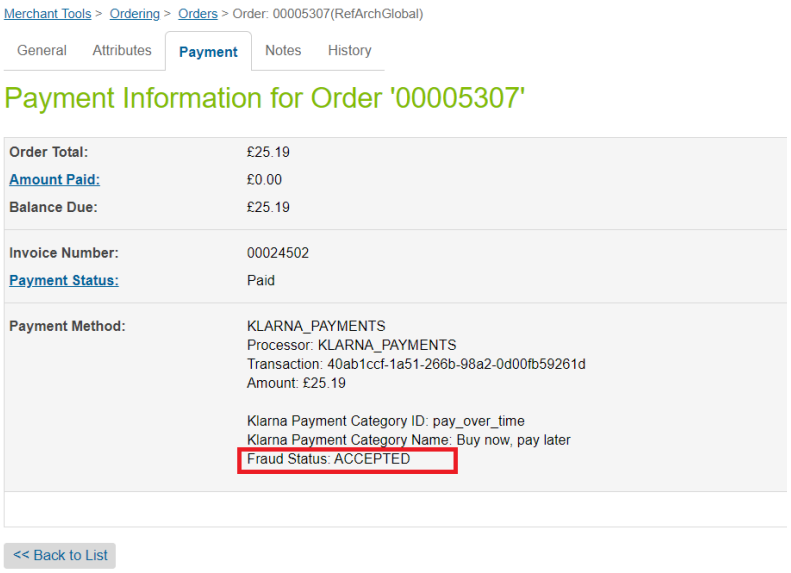


Figure 1. Klarna Payment Details in BM

- Orders with a `PENDING` status undergo further review. If later accepted as `FRAUD_RISK_ACCEPTED`, the SCC order status updates accordingly.

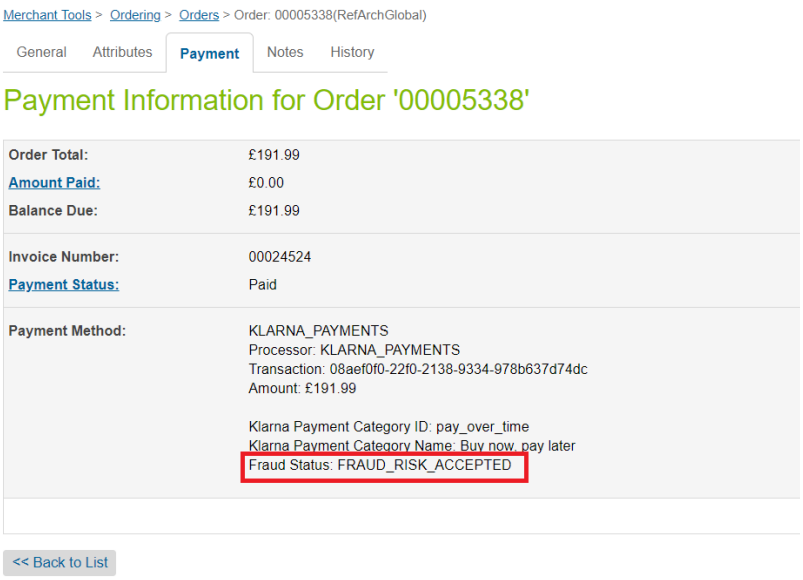


Figure 2. Klarna Payment Details in BM

Key Attributes and Business Manager Details

- **Custom Attribute for Payment Status:** The fraud status of Klarna Payments is stored in a custom attribute `kpFraudStatus` within the PaymentTransaction system object.
- **Business Manager View:** This status is visible in the Business Manager on the order details Payment tab.

2.3. Locales

The Klarna Payments cartridge supports multiple locales, ensuring a seamless experience for a global customer base. Supported locales include:

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

For a comprehensive list of the 25+ supported locales, please check Klarna's technical docs site.

2.4. Use Cases

2.4.1. Multiple payment options

During the checkout billing step, Klarna payment options are dynamically displayed based on the customer's cart details and the payment method categories available for the current Klarna session.

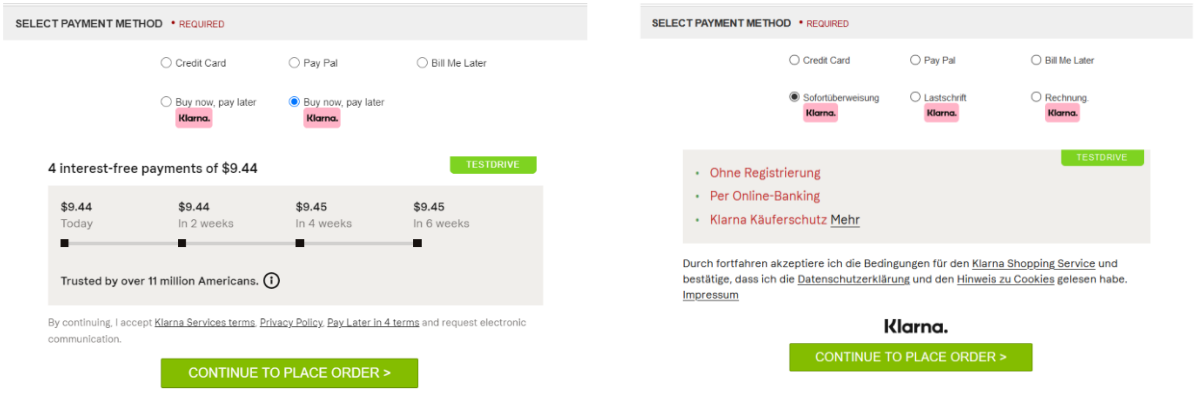


Figure 3. Example of payment options displayed at checkout

When the customer selects a payment method, a widget displaying additional information about the Klarna product will appear.

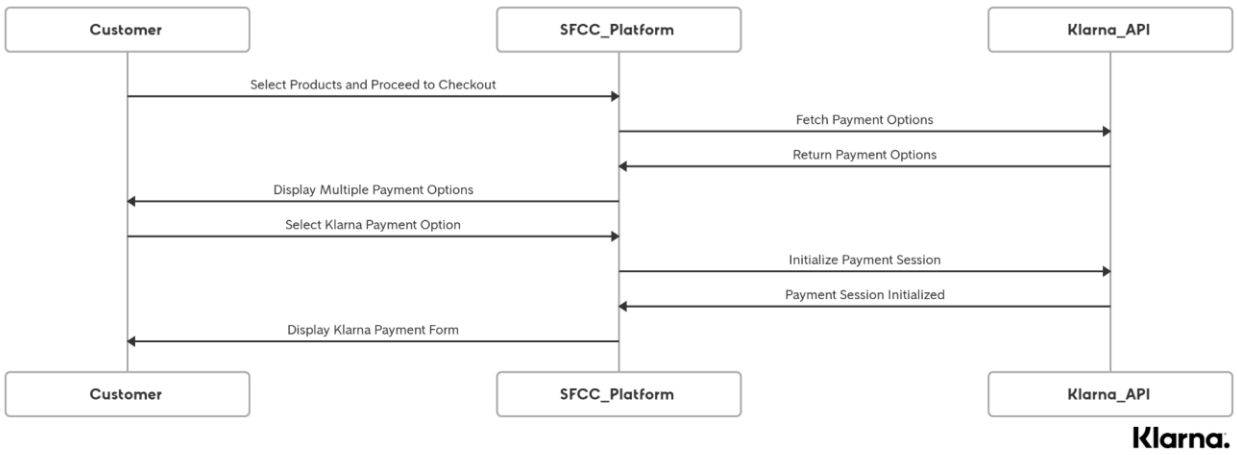


Figure 4. Klarna payment options retrieval

Note: The payment methods shown depend on market availability and the contractual agreement with Klarna.

2.4.2. Klarna Display Conditions and Authorization Handling

Display Conditions

Klarna payment methods may not be displayed under the following conditions:

1. **Payment Categories Not Returned:** Klarna is not displayed if payment categories are not returned in the session response.
2. **Session Linking Issue:** Payment method content is not displayed if we have payment methods in the session, but no session is linked to the basket.
3. **Authorization Denial:** Payment method category is hidden or grayed out if `show_form` is returned as false in the Klarna response on an authorization request and the payment method is denied.
4. **Session Creation Failure:** Overall, if Klarna is unavailable and a session is not created, payment categories will not be displayed.

Authorization Handling Preference

This functionality is deprecated as of release 24.4.0

By default no action will be applied upon authorization rejections. No configuration changes required.

We've implemented a site preference, `kpRejectedMethodDisplay`, allowing merchants to choose how to handle authorization rejections (refer to section 2.3.5 Refusal of Klarna Payments on Payment Method – Authorization). The available options are:

- **No action:** No specific handling for authorization rejections.
- **Hide:** Hide the payment method upon authorization rejection.

- **Gray out:** Gray out the payment method upon authorization rejection.

2.4.3. Authorizing and Placing Klarna Orders in Checkout

The cartridge implementation follows best practices for integrating Klarna payment methods, offering the inclusion of Extra Merchant Data (EMD) to optimize acceptance rates. EMD can include customer information and Buy Online, Pickup in Store (BOPIS) details when the custom site preference `attachments` is enabled. Merchants should review and validate EMD data based on their data privacy requirements prior to going live.

Authorization Process

When a customer selects a Klarna payment method and clicks "Continue to Place Order," their personally identifiable information (PII) is sent. This information is essential for assessing and verifying the customer's data to display available payment method options.

Steps:

1. **Customer Action:** The customer selects a Klarna payment method and clicks "Continue to Place Order."
2. **Authorization Initiation:** Authorization is initiated, sending PII to Klarna for verification.
3. **Authorization Outcome:**
    a. **Success:** The customer is taken to the summary page.
    b. **Additional Information:** Depending on the payment method and market, additional information may be requested before completing authorization.
    c. **Failure:** If authorization is not successful (`approved = false`), the customer remains on the billing page, no changes to payment options display.

Special Case: Pay Now

For the Pay Now payment category, an additional "finalize" call is required when the customer clicks the "Place Order" button on the review page. This ensures that funds are transferred only when the customer decides to place the order. A cookie called `selectedKlarnaPaymentCategory` is set in the customer's browser to initialize the "finalize" call to Klarna.

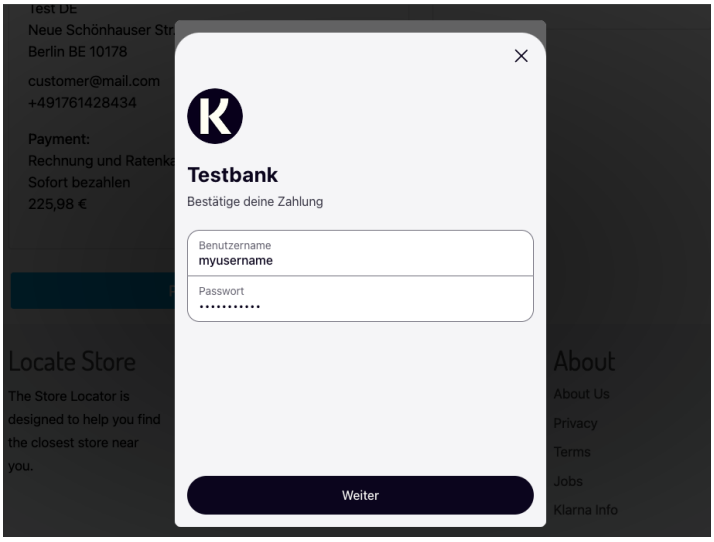


Figure 5. Finalize Call Screen

Klarna Order Placement

The Klarna order is placed just before the SFCC order is created:

- **Approved:** Successful payment authorizations (Klarna Payments authorization status: `APPROVED`) result in the creation of both Klarna and SFCC orders with a "Paid" order payment status and "Ready for Export" export status.
- **Rejected or Pending:** Refused and pending Klarna Payment orders (statuses `REJECTED` and `PENDING`) result in a "Not Paid" order payment status and "Not Exported" export status.
- **VCN Settlement Errors:** Klarna orders are canceled, and SFCC orders are set to a "Not Exported" export status.

Changing Payment Methods

If a customer authorizes a Klarna payment and then returns to the billing page to select a different non-Klarna payment method, an automatic `cancelAuthorization` call is triggered. This releases the authorized funds and frees up the available purchase amount. Merchants can utilize this function for specific use cases.

```
server.get('SaveAuth', function (req, res) {
```

```
	var KlarnaSessionManager =
require('*/cartridge/scripts/common/klarnaSessionManager');
	// var processor = require('*/cartridge/scripts/payments/processor');

	var token = req.httpHeaders['x-auth'];
	var finalizeRequired = req.httpHeaders['finalize-required'];

	//Cancel any previous authorizations
	//processor.cancelAuthorization();

	var klarnaSessionManager = new KlarnaSessionManager();
	klarnaSessionManager.saveAuthorizationToken(token, finalizeRequired);

	res.setStatusCode (200);
});
```

Testing

Merchants should thoroughly test the checkout flow, including scenarios such as:

- Klarna session flow
- Payment method switching
- Order amount updates
- Checkout with external payment methods

2.4.4. Refusal of Klarna Payments on Authorization

This functionality is deprecated as of release 24.4.0

By default no action will be applied upon authorization rejections. No configuration changes required.

Upon selecting Klarna as the payment method on the billing step of checkout and based on the customer information provided, Klarna Payments can be refused as a payment method. If the payment method is rejected with `show_form=false` and `approved=false` (i.e., hard reject), the merchant can choose how the payment option is displayed on the Billing page using the Business Manager (BM) preference **Hide Payment Methods on Deny** (`kpRejectedMethodDisplay`):

- **No:** Leave the payment visible to the customers.
- **Hide:** The payment option will be hidden from the customer.
- **Gray Out:** The payment option will be grayed out and not clickable.

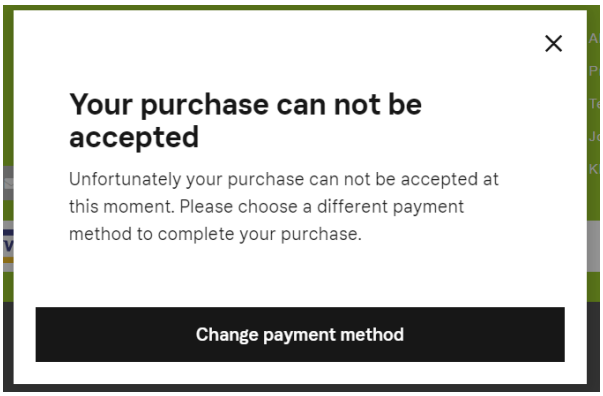


Figure 6. Denied order popup

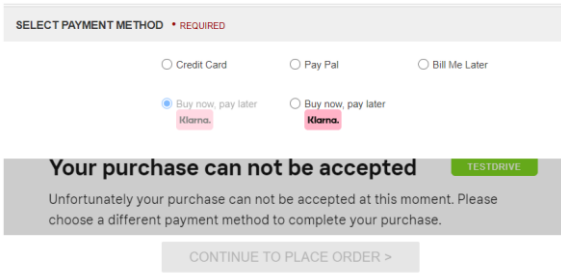


Figure 7. Greyed Out Payment Option

Note: Reloading the page will show the denied Klarna payment method again.

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

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

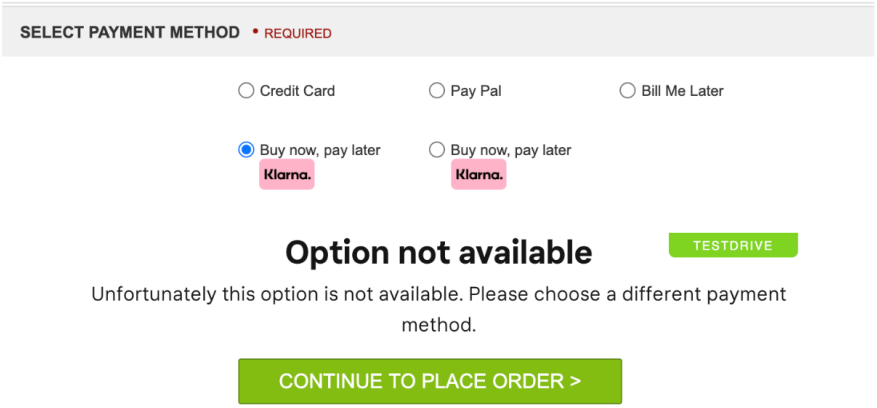


Figure 8. Option Not Available

2.4.6. Handling Unavailability of Klarna API or Inapplicable Storefronts

If the Klarna API is unavailable or the site/storefront is inapplicable, Klarna will not be presented as a payment option on the billing page. It is recommended that a Klarna session not be created when a customer selects a non-Klarna market or merchant store. This also applies to cases where merchants have multi-currency storefronts with a basket currency not supported by Klarna.

2.4.7. Notification Handling and Fraud Status Updates

When a Klarna order is created but flagged for additional review, the Commerce Cloud order remains in the `Created` status with a Fraud Status of `PENDING`. This order is marked with `EXPORT_STATUS_NOTEXPORTED`, confirmation status `NOTCONFIRMED`, and `NOTPAID`.

Notification Process

For orders with a Fraud Status of `PENDING`, updates are sent to the pre-configured `notification_url` on the merchant's Commerce Cloud site once the review is complete. The updated Fraud Status, such as `FRAUD_RISK_ACCEPTED`, is displayed in Business Manager (BM). Push notifications are sent repeatedly (up to 24 hours, every 10 minutes) until the POST request is acknowledged with a `200` response.

Event Types

Klarna sends one of the following event types in the notification to SFCC to update the risk status:

- `FRAUD_RISK_ACCEPTED`
- `FRAUD_RISK_REJECTED`
- `FRAUD_RISK_STOPPED`

Notifications are generally received within 4-24 hours. The order's payment transaction is updated accordingly (see `kpFraudStatus`). This status can be viewed in BM on the order details Payment tab.

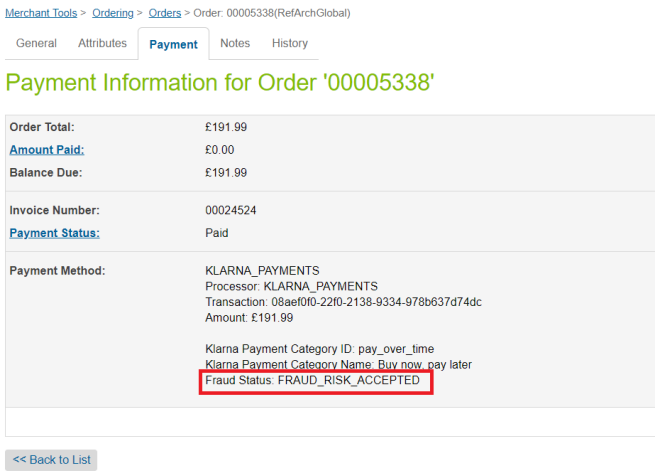


Figure 9. Fraud Status in Payment Details

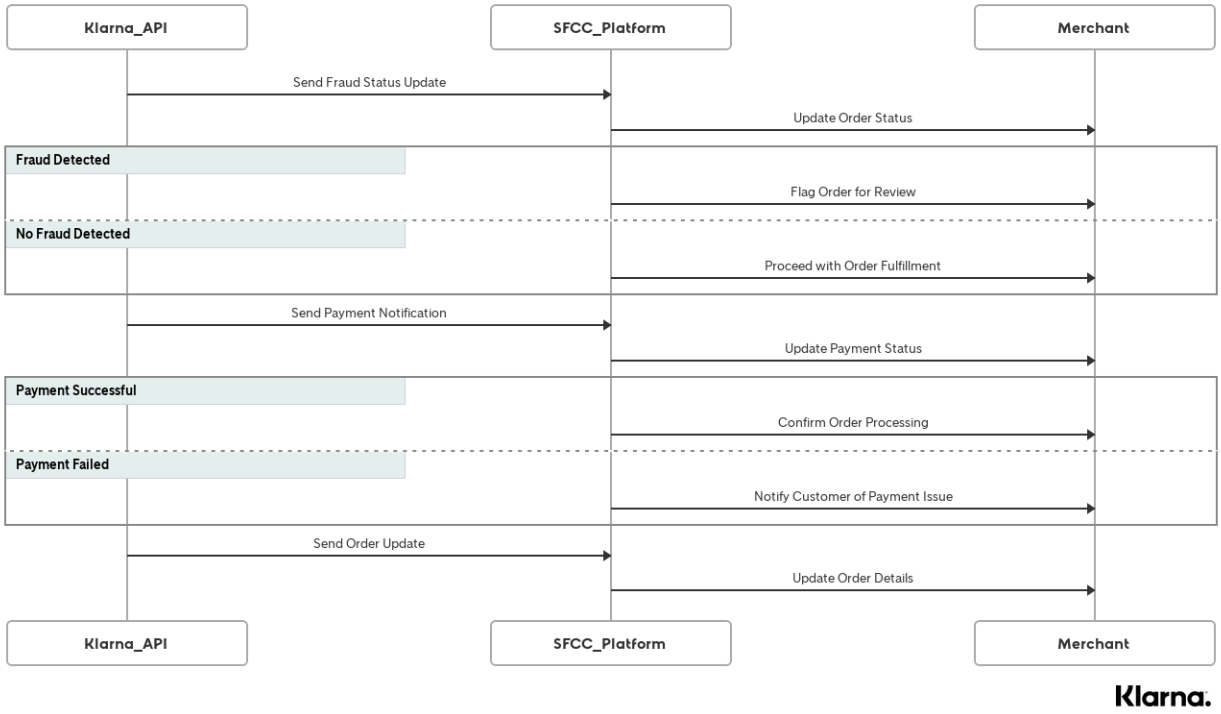


Figure 10. Notification handling and fraud status updates

Handling Fraud Risk Updates

- `FRAUD_RISK_ACCEPTED`
  - **Order Status:** Changes to `OPEN`.
  - **Confirmation Status:** Changes to `CONFIRMATION_STATUS_CONFIRMED`.
  - **Export Status:** Changes to `EXPORT_STATUS_READY`.
  - **Auto-Capture Orders:** The payment status will be set to `PAYMENT_STATUS_PAID`, and the full order amount will be captured.
- `FRAUD_RISK_REJECTED` or `FRAUD_RISK_STOPPED`
  - **Order Status:** Marked as failed (`FAIL`) in SFCC.

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

2.4.8. Virtual Card Settlements

This feature is disabled by default. However, if standard order management is not suitable for your Klarna integration, you can utilize Klarna's Merchant Card Service-based virtual card solution. The virtual card is issued against a Klarna order to capture the authorized order amount using standard card rails.

When a customer places an order, it is first booked in SFCC. Once Klarna accepts the order, the Klarna cartridge integration creates a virtual card-based settlement using the Merchant Card Services (MCSv3) API.

After the settlement is created (virtual card returned), the encrypted card details are saved in SFCC. These details can later be used by the merchant's OMS platform or custom PSP integration to authorize the virtual card until the Klarna order is valid. On successful order fulfillment, the funds on the virtual card may be captured. (For delays in capture or special use cases, please speak with the Klarna Key Account Manager in advance). While Klarna is the original payment method, the order amount will be settled with the merchant using the issued virtual credit card instead of a direct bank account transfer.

Please review the procedures with the Klarna team for details of the settlement process using virtual cards. Refer to the code in `scripts/payments/processor.js` and update accordingly to the integrated cards processor.

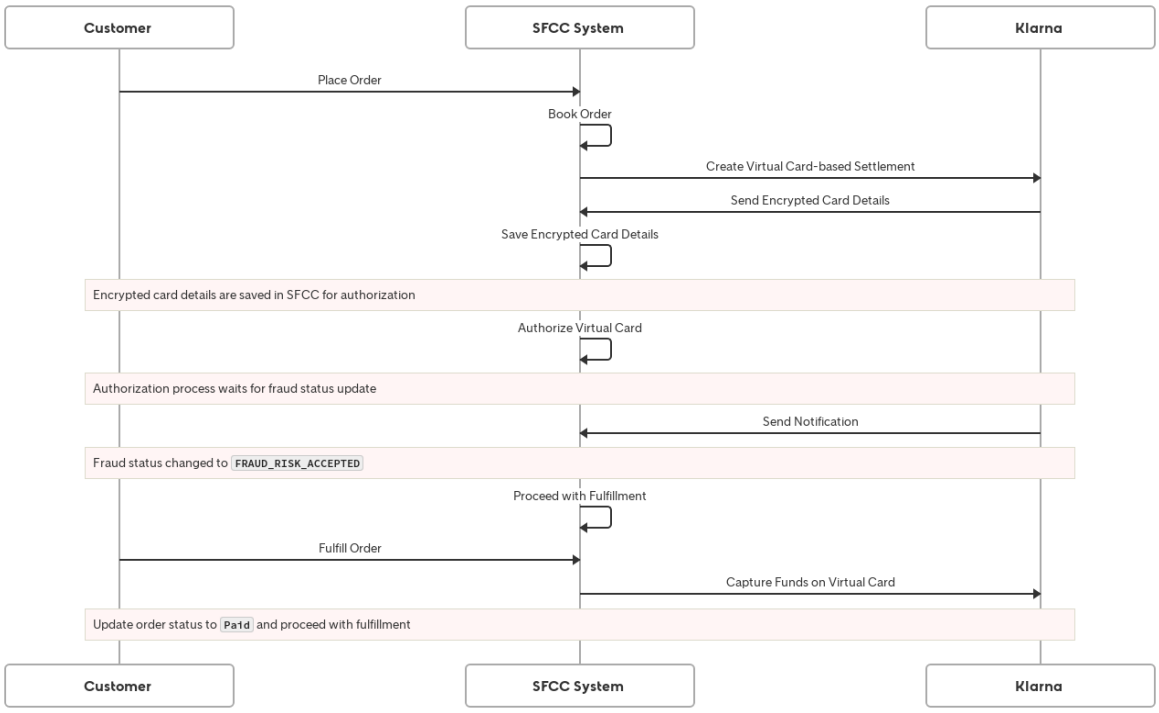


Figure 11. VCN Process in Klarna Payments Cartridge

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 Decrypt VCN Card Details.

Credit card authorization call:

```
/**
* Call Credit Card Authorization Hook (for VCN settlement)
* @param {dw.order.order}.order DW Order
* @returns {processorResult} authorization result
*/

function callCreditCardAuthorizationHook(order) {
      var processorResult = null;
      var paymentInstrument = order getPaymentInstruments(PAYMENT_METHOD)[0];
      var paymentProcessor = PaymentMgr
```

```
                .getPaymentMethod(paymentInstrument.paymentMethod)
                .paymentProcessor;
        var transactionID =
paymentInstrument.getPaymentTransaction().getTransactionID();

        var hook = 'app.payment.processor.' + CREDIT_CARD_PROCESSOR_ID;
        if (!HookMgr.hasHook(hook)) {
                throw new Error('File of app.payment.processor.' +
CREDIT_CARD_PROCESSOR_ID + 'hook is missing or the hook is not configured'
        }

        processorResult = HookMgr.callHook('app.payment.processor.' +
CREDIT_CARD_PROCESSOR_ID, 'Authorize', transactionID, paymentinstrument,
paymentProcessor);
return processorResult;
}
```

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

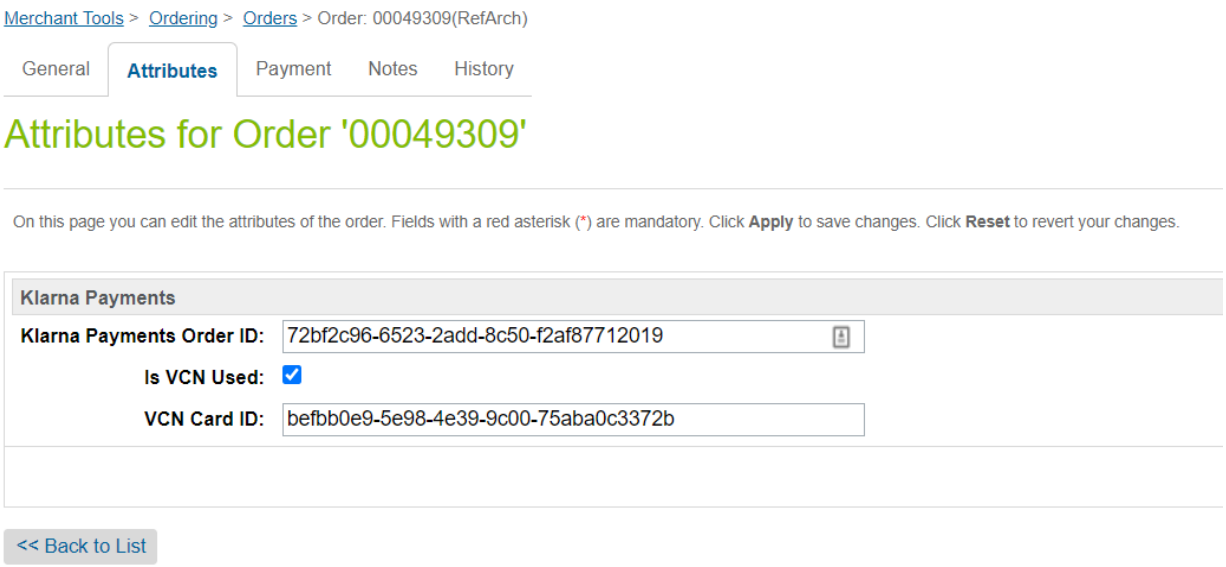


Figure 12. VCN Details in Order

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.

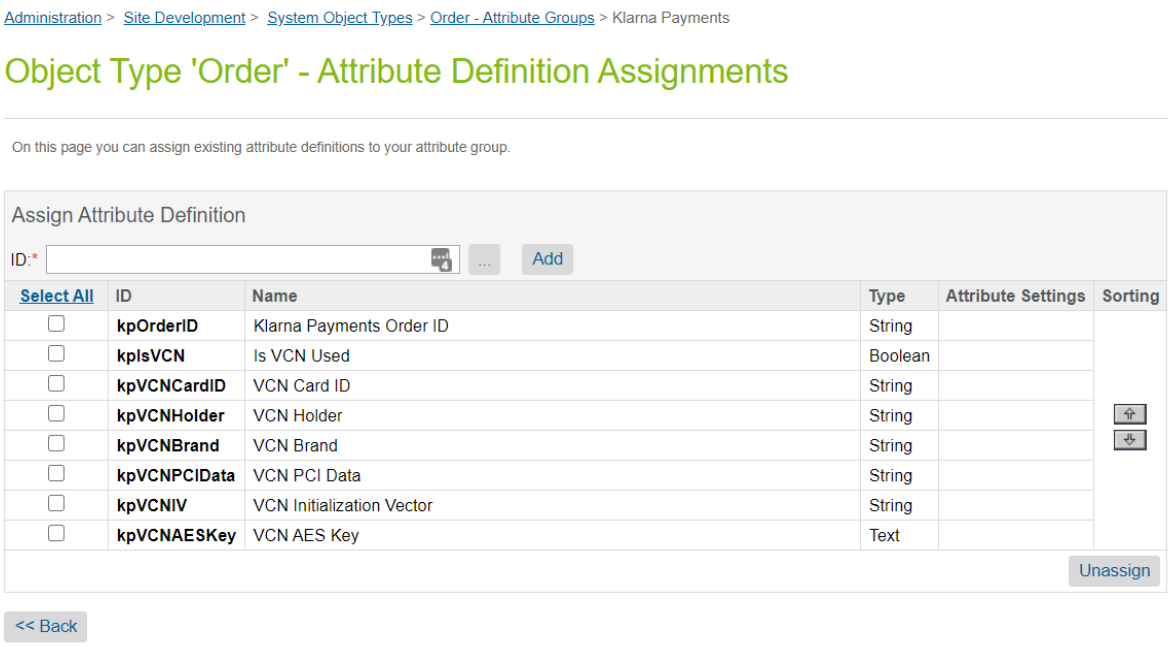


Figure 13. Full List of VCN Attributes

Work with your 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 about other supported use cases.

Important!

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 coordination 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 the "OrderCleanUp" job section).

Depending on their credentials configuration, merchants should follow these steps to utilize the virtual card integration option:

For single credentials per site configuration

1. Enable VCN option in Site Preferences as shown below.

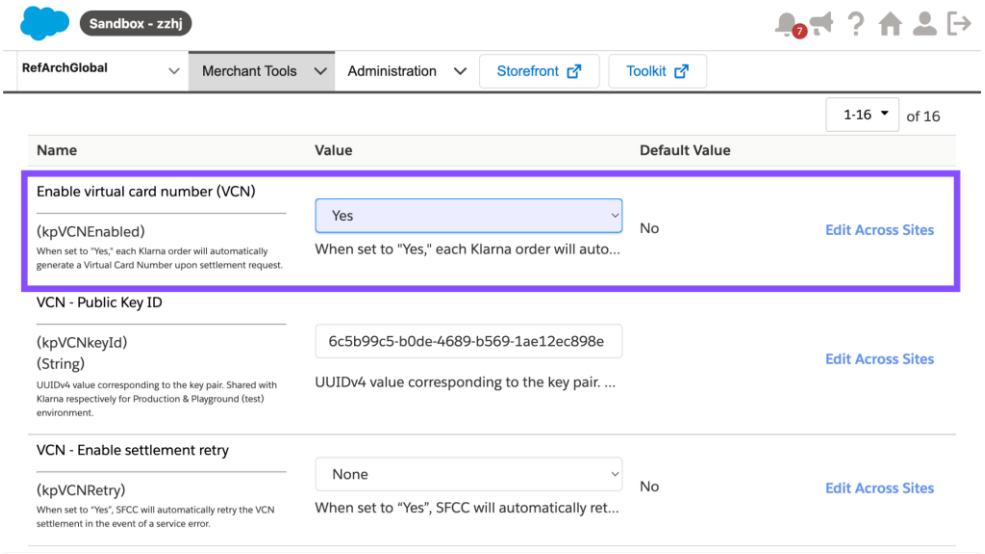


Figure 14. VCN enablement setting

2. Enter the VCN Public Key ID. Unique UUIDv4 value, which should be different for playground testing and Production (live site).

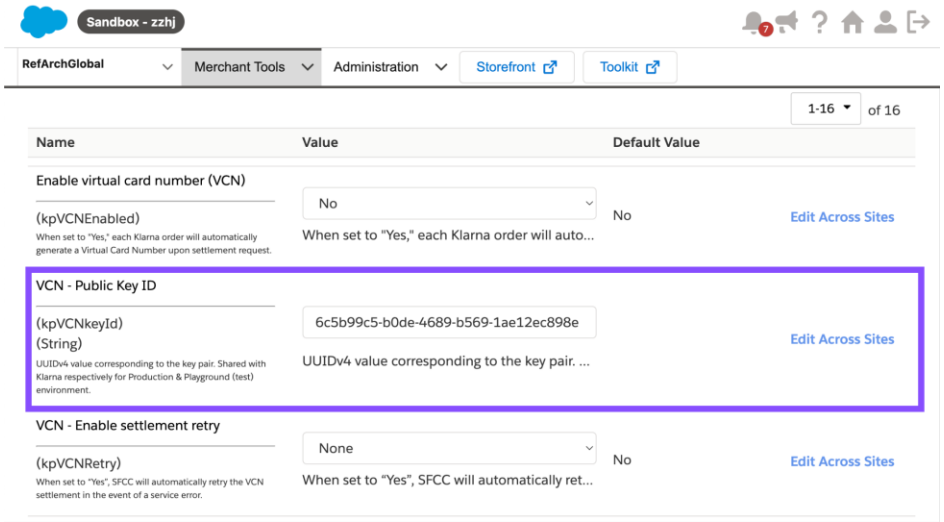


Figure 15. VCN Public Key ID

3. Generate a 4096-bit RSA key pair (Refer to section Generate Key Pair and Key Id for Virtual Card Settlements).
4. Update the VCN settlement retry setting. 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.

5. Send the generated unique `key_id` + `public key` combination in JWK format to Klarna before testing and going live. It will be used to encrypt the AES key, which encrypts the PCI data on Klarna's side when the settlement request is made. After confirmation from Klarna that the key has been successfully added to your merchant profile, you will be able to use the virtual card-based settlement option for Klarna payment methods.

For different credentials per site - (Using `KlarnaActivation` custom object)

1. Go to **Merchant Tools > Custom Objects > Manage Custom Objects** and select `KlarnaActivation`.
2. Search and locate the custom object entry.
3. Enable the VCN option.
4. Enter the VCN Public Key ID, a unique UUIDv4 value, which should be different for Playground (testing) and Production (live site) environments.
5. Update the VCN settlement retry setting. By default, this is disabled. If enabled, 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.

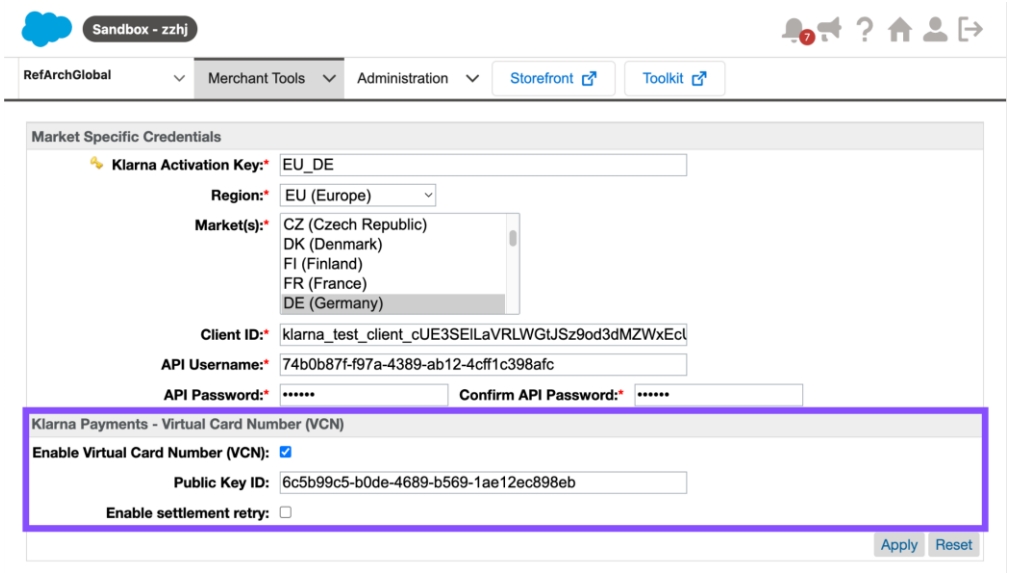


Figure 16. VCN settings in Klarna activation custom object

6. Generate a 4096-bit RSA key pair (Refer to the section "Generate Key Pair and Key Id for Virtual Card Settlements").
7. Send the generated unique `key_id` + `public key` combination in JWK format to Klarna before testing and going live. It will be used to encrypt the AES key, which encrypts the PCI data on Klarna's side when the settlement request is made. After

confirmation from Klarna that the key has been successfully added to your merchant profile, you will be able to use the virtual card-based settlement option for Klarna payment methods.

2.4.9. Enabling and Managing Auto-Capture for Payments

Auto-capture is enabled via a site preference `kpAutoCapture` located in the `Klarna_Payments` preference group. By default, auto-capture is disabled. When enabled, the system attempts to capture the full payment amount automatically.

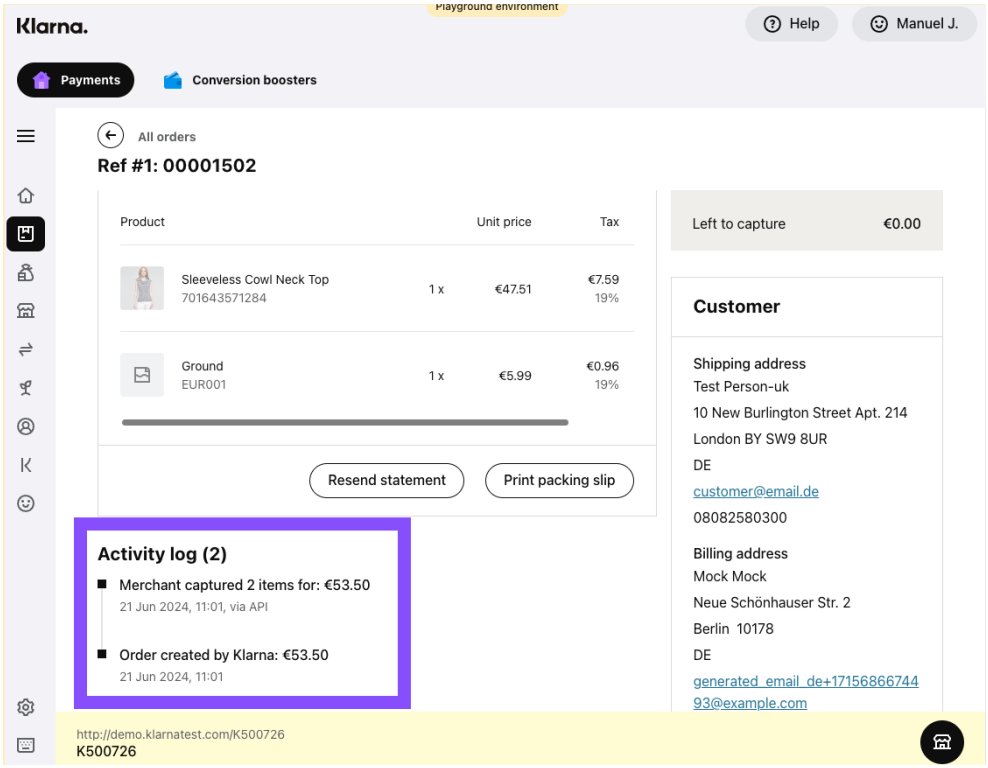


Figure 17. Order Details in Klarna Portal

Successful Capture

- The SFCC order's payment transaction is marked as `Paid`.
- The transaction status is viewable in the Business Manager.
- The order status in Klarna's Merchant Portal is updated to `Captured`.

Unsuccessful Capture

- An error is logged in the custom error log.
- The issue must be reviewed with the Klarna delivery team before testing and going live.

Note:

- Auto-capture is possible only for orders where Virtual Card Number (VCN) is not enabled.
- Ensure the setting is reviewed with the Klarna delivery team before testing and going live.

2.4.10. Widget Customizations

The Klarna Payments widget can be styled to align with the marketing and branding needs of the merchant's store. Various graphic elements of the widget can be customized through site preferences.

Customizable Graphic Elements

This functionality is partly deprecated as of release 24.4.0

Widget customization preferences must be entered using the new JSON object `kpColorCustomisation` instead of separate attributes.

For more details, refer to the Klarna Payments API documentation.

Below is a table of the graphic elements that can be customized along with their corresponding site preferences:

Element	Site Preference deprecated with release 24.4.0	New site preference since 24.4.0 release	Example Value
Details Color	`kpColorDetails`	`kpColorCustomization`	`#COFFEE`

Button Color	`kpColorButton`	Enter the needed customizations in the JSON object, make sure to follow the options object specified in the Klarna documentation	`#COFFEE`
Button Text Color	`kpColorButtonText`		`#COFFEE`
Checkbox Color	`kpColorCheckbox`		`#COFFEE`
Checkbox Checkmark Color	`kpCheckboxCheckmark`		`#COFFEE`
Header Color	`kpColorHeader`		`#COFFEE`
Link Color	`kpColorLink`		`#COFFEE`
Border Color	`kpColorBorder`		`#COFFEE`
Selected Border Color	`kpBorderSelected`		`#COFFEE`
Text Color	`kpColorText`		`#COFFEE`
Secondary Text Color	`kpColorTextSecondary`		`#COFFEE`
Border Radius	`kpRadiusBorder`		`0px`

Table 1. Klarna Payments widget customization elements

2.4.11. Customizing Payment Method Name

The payment method name “Klarna Payments” can be customized in the Business Manager.

Steps to Customize

1. Navigate to Merchant Tools > Ordering > Payment Methods in Business Manager.
2. Select the Klarna payment method.
3. Choose the desired language from the drop-down menu.

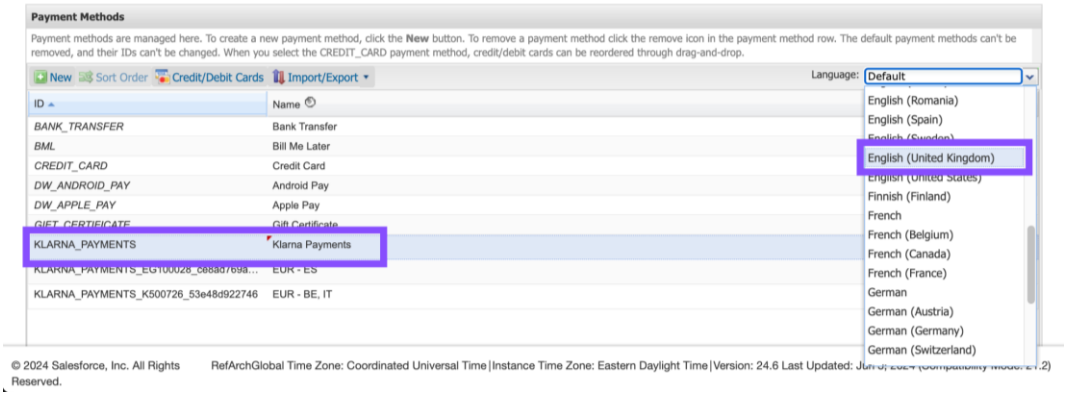


Figure 18. Customize Payment Name

The customized payment method name will be displayed in:

- The mini summary and confirmation screens
- Confirmation emails
- The My Account Order Details section

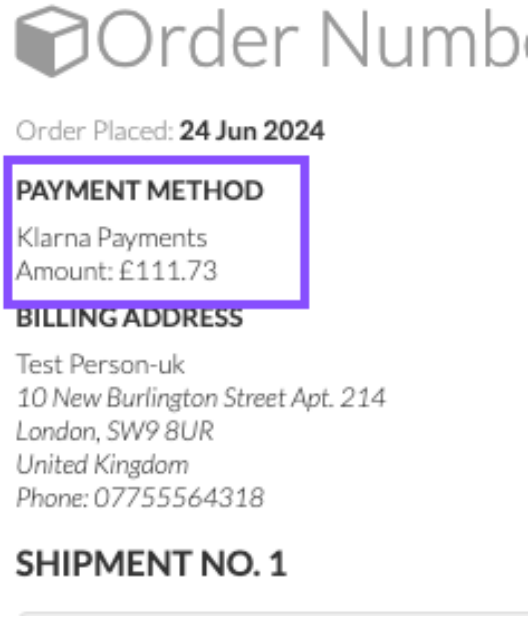


Figure 19. Payment Method Name in confirmation screen

2.5. Advanced features

2.5.1. Klarna Payment Method Based Promotions

Starting with the B2C 20.7 release, merchants can use payment methods as qualifiers for product, order, and shipping promotions.

By default, when a promotion is configured to use a payment method as a qualifier, the total order amount is shown to the customer upon reaching the review page. This can cause the Klarna authorization call to be made for a higher amount than the final total.

To resolve this, when a customer selects a payment option in the billing section, a backend call is triggered. This call recalculates the basket totals, including any applicable promotions, and updates the Klarna session details. Consequently, the Klarna iframe widgets and the mini summary section on the storefront are refreshed to display the final order details.

For payment methods other than Klarna, this logic should be customized by the merchant to handle any third-party payment integrations.

2.5.2. Price Adjustment Taxation Handling

Out of the box, the Klarna API sends product and shipping method details along with relevant discounts as separate line 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,
            "unit_price": 1919,
            "tax_rate": 500,
            "total_amount": 1919,
            "total_tax_amount": 68,
            "total_discount_amount": 0
      }
```

Handling Gross Taxation with Adjusted Prices

Merchants using gross taxation might opt to enable the “Tax Products and Shipping Only Based on Adjusted Price” preference. This preference is located under Merchant Tools > Site Preferences > Pricing and Promotion, and ensures that price adjustments are not taxed.

The setting `kpPromoTaxation` has been introduced for this purpose. Merchants should update this setting to match their promotion configuration as follows:

- **Price (Based on Price):** The product, shipping, and their discounts will be sent as separate line items. This is the default setting.
- **Adjustment (Based on Adjusted Price):** When selected, the product or shipping method line item will be sent with the attribute total_amount matching the prorated price and the 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,
            "unit_price": 1919,
            "tax_rate": 2200,
```

```
            "total_amount": 1419,
            "total_tax_amount": 256,
            "total_discount_amount": 500,
            "product_url": null,
            "image_url": null
        }
```

Note: Enabling this setting is not required for storefronts using net taxation, as the tax is not included in the product's base price. In such cases, the total order sales tax is sent to Klarna as a separate line item, not at the product or shipping line-item level.

2.5.3. Buy Online, Pickup in Store (BOPIS)

When store pickup is enabled on the storefront, the integration sends store details to Klarna during the authorization request and when placing the Klarna order. Store information is only sent after the customer interacts with the Klarna payment method widgets.

Address Handling

The store address(es) are included in the Enhanced Merchant Data (EMD) attachment under the `other_delivery_address` attribute when applicable. The handling of addresses in Klarna orders with store pickup is as follows:

- **Orders with Only Store Pickup Shipments:** If there is no home delivery address, the shipping address in the Klarna order will be set to the first store shipment's details.
- **Orders with Both Store Pickup and Home Delivery Shipments:** The home delivery address will be used as the shipping address in the Klarna calls.
- **Orders with No Store Pickups:** No information is sent in the `other_delivery_address` 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 to the Klarna documentation here.

2.5.4. Service Rate Limits

This functionality is deprecated as of release 24.4.0

- **Rate limits are managed by Klarna.** The plugin handles the errors in case these are exceeded. Default service `klarna.http.defaultendpoint` will be used for all API payments calls.
- For more information refer to the Klarna technical documentation here.

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, contact your Klarna account manager.

2.5.5. Klarna Subscriptions

The Klarna cartridge supports subscription handling, allowing merchants to offer products as subscriptions, manage subscription details, and handle recurring orders seamlessly.

2.5.5.1. Configuration

Subscription details are configured at the product level. Products can be set as subscription-only, standard, or both. The trial period must be an integer value.

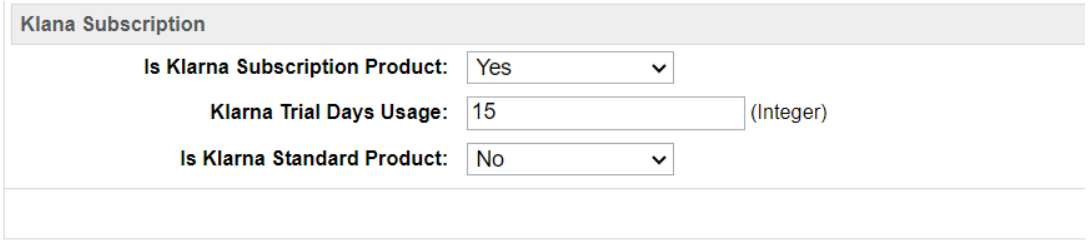


Figure 20. Product subscription configuration

2.5.5.2. Cart Page

There are two types of subscription products:

- **Subscription-only Products**: These are automatically added to the shopping cart as subscription line items.
- **Dual-purpose Products**: For products that can be either subscription or standard, users can select their preference on the cart page.

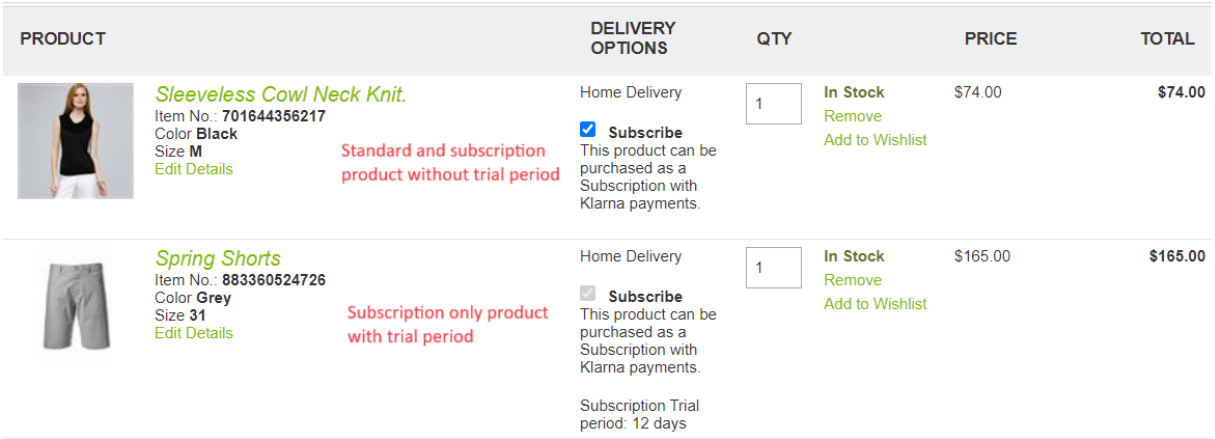


Figure 21. Subscription products on cart page

Important!

The checkout process will not 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.

Subscription Details

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

- `kpSubscriptionFrequency`
- `kpSubscriptionPeriod`

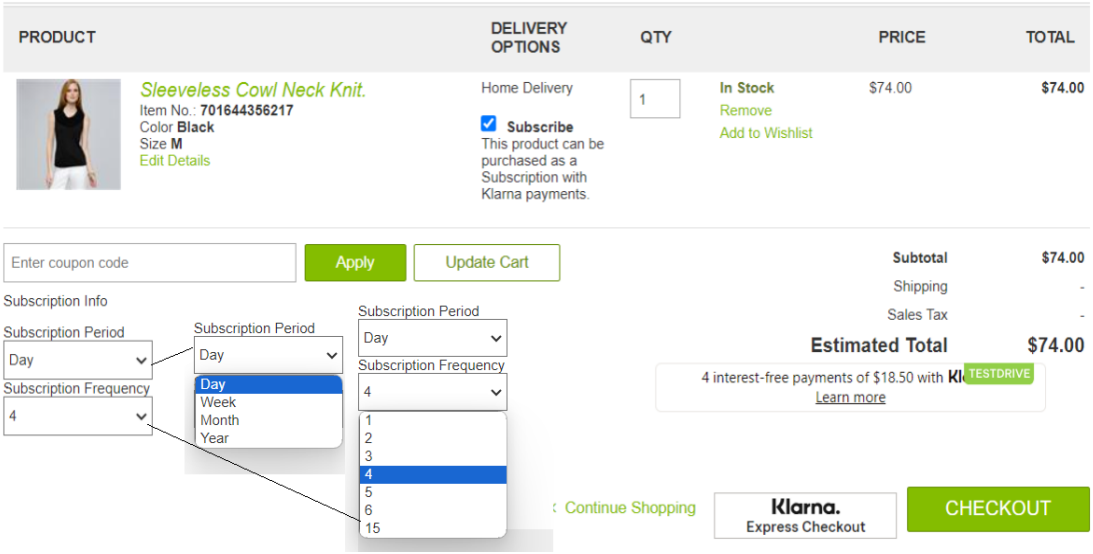


Figure 22. Subscriptions details in cart page

2.5.5.3. Checkout

Only logged-in users can complete a subscription checkout. Session intent is defined based on the basket content:

- **`tokenize`**: Basket contains products with a trial period; user is not charged on order creation.
- **`buy_and_tokenize`**: Basket contains subscription products without a trial period.
- **`buy`**: Basket contains only standard products, no subscription products.

For intents `tokenize` and `buy_and_tokenize`, a Klarna customer token for recurring payments is created and stored in the customer profile for future use.

User Profile Updates

Upon order creation, the user profile is updated with subscription data:

1. **`token`**: Subscription token.
2. **`enabled`**: Status of the subscription.
3. **`nextChargeDate`**: Calculated date for the next subscription charge.
4. **`subscriptionPeriod`**: Enumerated value representing the subscription period ('week', 'month', or 'year').
5. **`subscriptionFrequency`**: Numeric value representing the frequency of the subscription (1, 2, 3, 4, 5, 6, 15).
6. **`subscriptionProductID`**: ID of the corresponding subscription product.
7. **`lastOrderID`**: ID of the last order for the subscription.

2.5.6. Account Subscription Dashboard

Users can view a full list of their subscriptions in the My Account section. They can cancel subscriptions, which will be displayed with an Inactive status. The Cancel Subscription button deactivates the customer token, preventing further charges.

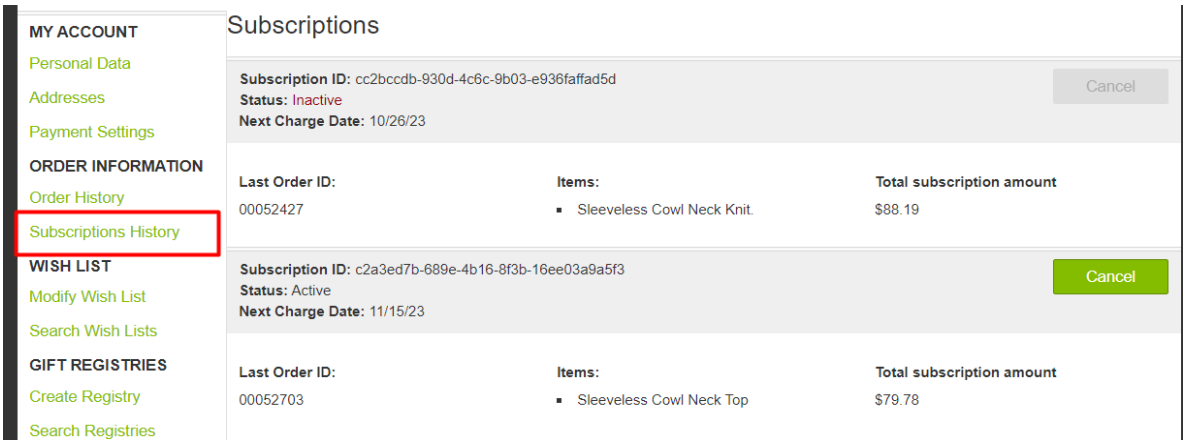


Figure 23. My Account Subscription dashboard

2.5.7. Recurring Subscription Order Creation

A back-end job processes recurring subscriptions for each user in SFCC if subscriptions exist. The job iterates through customers and checks for subscriptions due for payment on the same day.

Order Creation Process

1. **Create new order**: Using data from `lastOrderID`, a new order is created, and a charge call is made to Klarna using the token and price from `lastOrderID`.
2. **Update details on success**: On success, `nextChargeDate` is updated based on `subscriptionPeriod`, and `lastOrderID` is set to the ID of the newly placed order. The Subscription Dashboard is updated accordingly.

Handling Failures

- **Retry mechanism**: Merchants can configure a retry mechanism with specific retry intervals. If retries fail, the subscription can be deactivated.
    - **Retry**: Boolean field (Yes/No).
    - **Number of retries**: Number of retries (1, 2, etc.).
    - **Retry frequency**: Interval in days (1, 2, etc.).
- **Cancel subscription**: If retry is disabled, the subscription is canceled.

Orders with trial period subscriptions are paid after the trial period ends. On the next charge date, a new order is created with the channel type set to `SUBSCRIPTIONS`.

2.6. Klarna Express Button (KEB)

This functionality is deprecated as of release 24.4.0

Klarna Express Button has been replaced with Klarna Express Checkout.

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 prefilled checkout experience, which also includes first time shoppers on a merchant storefront.

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.

For storefronts with custom checkout design or address field validations, please ensure that the authenticated users address data is populated without modification to ensure the Klarna Payment method authorization is completed successfully.

Note: On redirect, in the standard checkout, any existing email ID and phone number are updated with the latest provided by the shopper.

2.7. Compatibility

This cartridge has been tested against API Version 22.6 (Compatibility Mode: 22.7) and SG version 105.0.0.

2.8. Privacy and Payment

2.8.1. GDPR Compliance

The cartridge is compliant with GDPR recommendations and follows best practices to ensure only necessary Personally Identifiable Information (PII) is transmitted to authorize the payment method. For detailed guidelines, refer to the implementation best practices here.

2.8.2. EMD (Extra Merchant Data)

The cartridge supports sending additional information on the customer's past purchase history and "Buy Online, Pickup in Store" (BOPIS) store addresses when enabled in custom preferences under "Attachments" (`kpAttachments`). The types of data that can be sent as an attachment are detailed here.

EMD is required for certain types of merchant orders and generally improves acceptance rates (e.g., `customer_account_info`: past interactions with the merchant store). EMD is included as part of the authorization step in the Commerce Cloud checkout. The data sent to Klarna is customizable and can be viewed in `int_klarna_payments/scripts/payments/additionalCustomerInfo.js`. This script should return a JSON string to be used as the value for the body sub-field of the attachment field as described here.

Example Schema for Additional Customer Information

If the example `additionalCustomerInfo.js` file is used unchanged, the data sent to Klarna follows this schema:

```
{
    "$schema": "http://json-schema.org/draft-03/schema#",
    "id": "http://klarna.com/v2/emd#",
    "description": "Extended Merchant Data Payload Schema",
    "type": "object",
    "properties": {
        "customer_account_info": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "unique_account_identifier": {
                        "type": "string",
                        "maxLength": 24
                    },
                    "account_registration_date": {
                        "description": "ISO 8601 e.g. 2012-11-24T15:00",
                        "type": "string",
                        "format": "date-time",
                        "pattern": "^[0-9]{4}-[0-1][0-9]-[0-3][0-9]T[0-2][0-9]:[0-
5][0-9](:[0-5][0-9])?Z?$"
                    },
                    "account_last_modified": {
                        "description": "ISO 8601 e.g. 2012-11-24T15:00",
                        "type": "string",
                        "format": "date-time",
                        "pattern": "^[0-9]{4}-[0-1][0-9]-[0-3][0-9]T[0-2][0-9]:[0-
5][0-9](:[0-5][0-9])?Z?$"
                    }
                }
```

```
            }
        },
        "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]{4}-[0-1][0-9]-[0-3][0-9]T[0-2][0-9]:[0-
5][0-9](:[0-5][0-9])?Z?$"
                    },
                    "date_of_first_paid_purchase": {
                        "description": "ISO 8601 e.g. 2012-11-24T15:00",
                        "type": "string",
                        "format": "date-time",
                        "pattern": "^[0-9]{4}-[0-1][0-9]-[0-3][0-9]T[0-2][0-9]:[0-
5][0-9](:[0-5][0-9])?Z?$"
                    }
                }
            }
        },
        "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"
            }]
        }
    }
}
```


Note: 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.

2.8.3. PCI-DSS Compliance

The virtual card (MCSv3) solution enables settlements using individual virtual cards issued against a Klarna order. To comply with PCI-DSS requirements, merchants must ensure that data is securely maintained and transmitted as part of their operation in their live store environment. The required steps must be completed in consultation with your payment

service provider/acquirer before going live. Review the order export details required for virtual card-based Klarna orders in advance. Any historical decrypted PCI data should be expunged, regardless of the VCN validity date.

Important!

DO NOT SAVE UNENCRYPTED PCI DATA ON THE SERVER.

3. Conversion Boosters

3.1. 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 inform shoppers about the different payment options available as they browse your site. By using Klarna, customers have access to 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

3.1.1. Configuration

3.1.1.1. Configuring OSM via KlarnaCountries custom object

This functionality is deprecated as of release 24.4.0

Current configuration using attributes from Custom Object is deprecated with version 24.4.0 In the new implementation, settings are retrieved from Site Preferences as explained in section 3.1.1.2.

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, 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 the Klarna Merchant Portal within the On-site Messaging App.

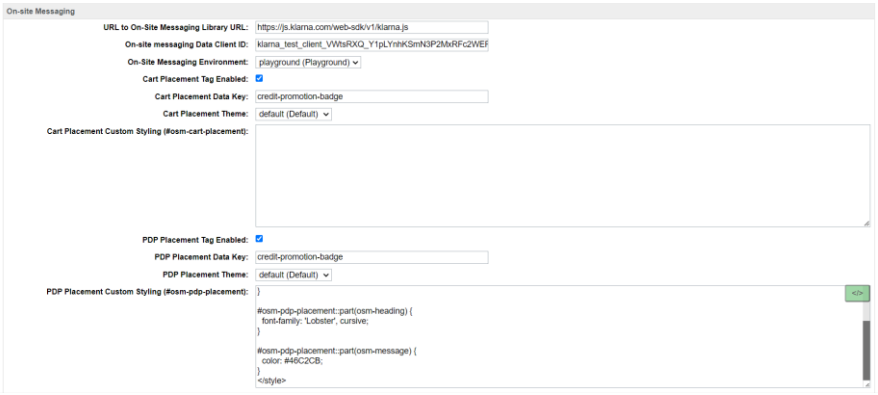


Figure 24. OSM Settings in BM

To enable the Placement tag for the Cart Page, the "Cart Placement Data Key" must be filled with the Data Key value and the "Cart Placement Tag Enabled" must be checked. To customize the Cart Placement, select a theme and/or enter custom CSS in the "Cart Placement Custom Styling" attribute using the placement id `#osm-cart-placement`.

Note: Cart placements amount must be updated, and the latest Klarna credit offering placement (where required) displayed to the customer when the order line quantity is updated on the cart page. Refer to section `app_storefront_base\cartridge\client\default\js\cart\cart.js (required)`.

To enable the Placement tag for the PDP Page, the "PDP Placement Data Key" must be filled with the Data Key value and the "PDP Placement Tag Enabled" must be checked. To customize the PDP Placement, select a theme and/or enter custom CSS in the "PDP Placement Custom Styling" attribute using the placement id `#osm-pdp-placement`.

To enable the Placement tag for the header, the "Header Placement Data Key" must be filled with the Data Key value and the "Header Placement Tag Enabled" must be checked. To customize the Header Placement, select a theme and/or enter custom CSS in the "Header Placement Custom Styling" attribute using the placement id `#osm-header-placement`.

To enable the Placement tag for the footer, the "Footer Placement Data Key" must be filled with the Data Key value and the "Footer Placement Tag Enabled" must be checked. To customize the Footer Placement Tag, select a theme and/or enter custom CSS in the "Footer Placement Custom Styling" attribute using the placement id `#osm-footer-placement`.

To enable the Placement tag for the Info Page, the "Info Page Placement Data Key" must be filled with the Data Key value and the "Info Page Placement Tag Enabled" must be checked. To customize the Info Page Placement Tag, select a theme and/or enter custom CSS in the "Info Page Placement Custom Styling" attribute using the placement id `#osm-info-page-placement`.

In the Library URL, input the full URL to the On-Site Messaging JavaScript Library. In On-Site Messaging Environment, select the corresponding environment (playground for test or production). In On-site Messaging Data Client ID, input the On-Site Messaging Data client ID value. For On-site Messaging Data locale, input the valid On-Site Messaging Data locale.

All placements can be customized with custom CSS using the placement component id in the brackets of the attribute name. The customization value should be wrapped in a `<style>` element.

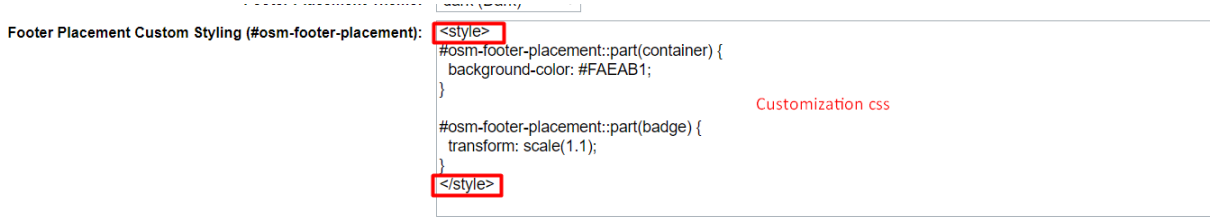


Figure 25. OSM Placements CSS customization

3.1.1.2. Configuring OSM via site preferences

Current configuration under BM is placed in Merchant Tools – Site Preferences-Custom Preference under `Klarna_OSM` group:

- To enable Klarna OSM implementation in storefront, `osm_enable` must be set to `true`.
- `osm_theme` dropdown is used to apply a theme to OSM. Options include `default`, `dark`, and `custom`.
- To enable placement in PDP, cart, header, footer, and info page, `osm_placement` is enabled in the multi-select dropdown. Placement tag ID is hardcoded from code.

Placement	TagID
Product Page	`credit-promotion-auto-size`
Cart Page	`credit-promotion-badge`
Site Wide Banner	`top-strip-promotion-badge`
FAQ	`info-page`
Footer	`footer-promotion-auto-size`

Table 2. Klarna On-site messaging placement tagID for different pages

Placement styling is taken from `osm_placement_styling`, which is a JSON. `osmLibraryURL` is hardcoded and retrieved from constants. `osmEnvironment` is used from `KlarnaPayments`.

CSS customizations are available only for the new OSM library version. Please follow Klarna guidance for placement styling - Styling On-Site Messaging with CSS.

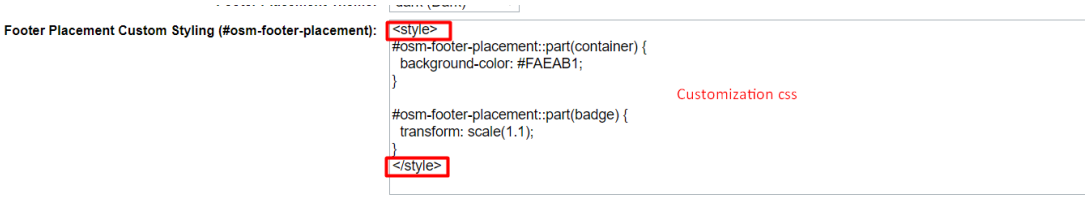


Figure 26. OSM Configuration in BM

For Canada only, update `osmDataInlineEnabled` 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`.

This functionality is deprecated as of release 24.4.0

As per version 24.4.0, `osmDataInlineEnabled` is deprecated.

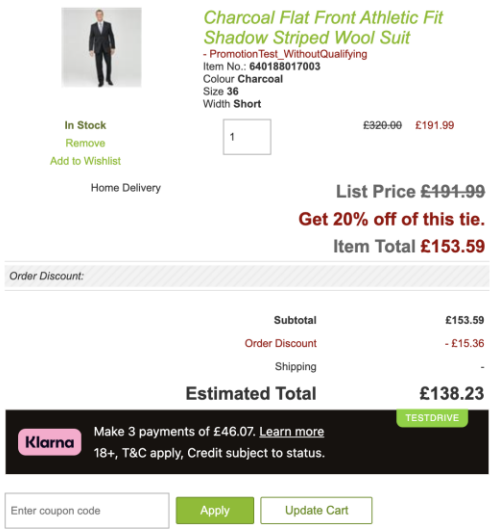


Figure 27. On-Site Messaging on Cart Page

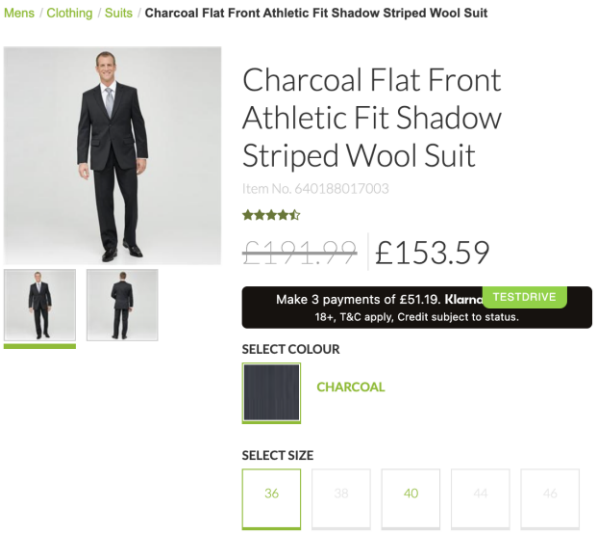


Figure 28. On-Site Messaging on PDP Page

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, update the `footer-about` content asset to include this line of code as shown below:

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

Figure 29. Footer asset update

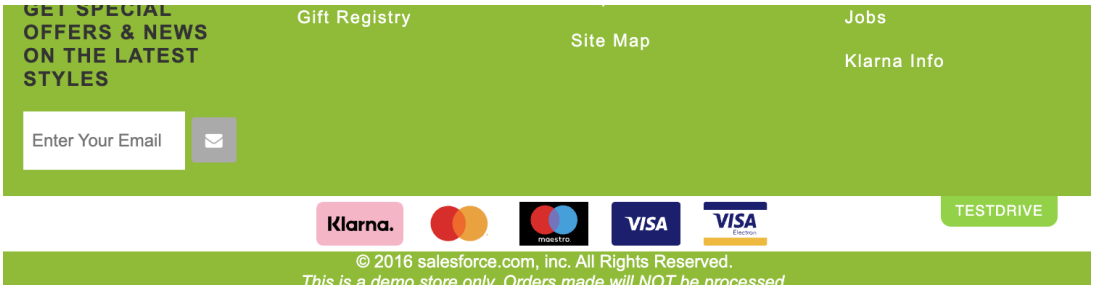


Figure 30. On-Site Messaging on footer and linked to Klarna Info Page

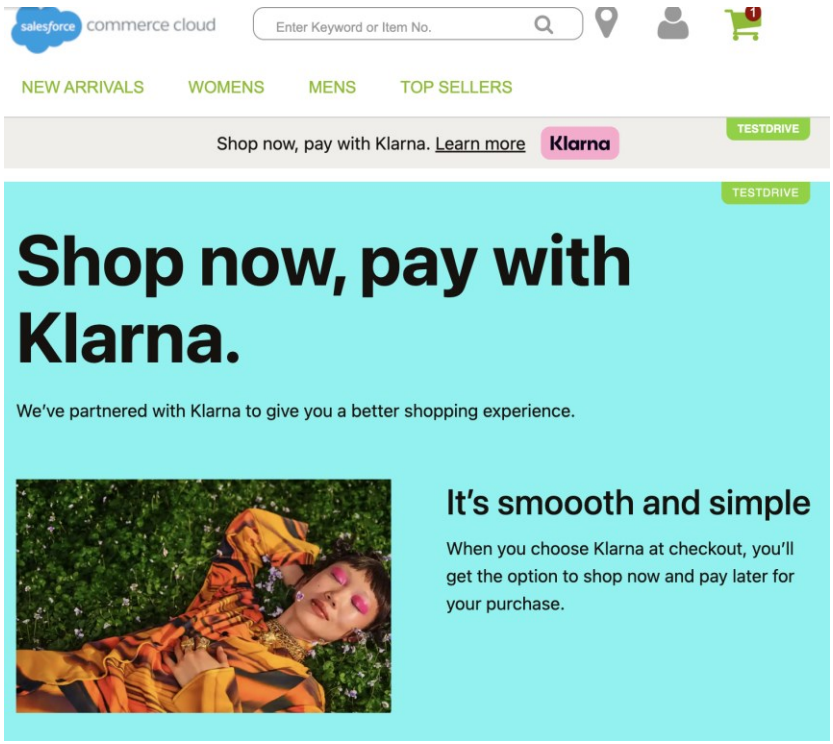


Figure 31. On-Site Messaging on header & dedicated Klarna info page

For more information regarding OSM customizations and best practices, please refer to the Klarna Developer Docs. Integration Best Practices and information about Klarna Branding and Co-marketing options can be found here.

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

3.2. Klarna Express Checkout

Klarna Express checkout (KEC) is a new feature introduced in Storefront that displays an Express Checkout Button on the Product Detail Page (PDP), Cart, and Mini Cart. Users are redirected to Klarna upon clicking the Express Checkout button. This feature enables a quick and easy checkout process where the Shipping Address, Billing Address, and Payment details are preselected, allowing the checkout to be completed in fewer clicks. For more information, refer to the Klarna Documentation. The multi-step checkout process includes a finalize call at the place order stage.

3.2.1. Configuration

The basket includes a custom attribute, `kpIsExpressCheckout`, which is set to `true` to enable the Express Checkout process.

To configure Express Checkout, navigate to **Merchant Tools > Site Preferences > Custom Site Preference Groups > Klarna Express checkout**. Here, you can select the locations within your store where you want the button to appear and customize its appearance by choosing the theme and shape.

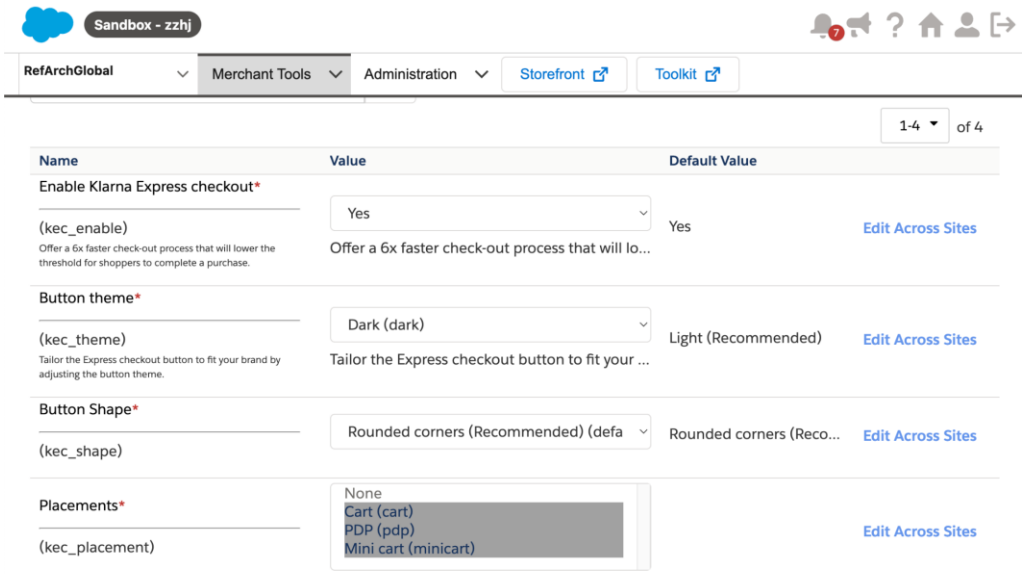


Figure 32. Klarna Express checkout configuration

3.2.2. Placements

Product Detail Page (PDP)

The Express checkout button is placed near the “Add to cart” button to offer a seamless checkout alternative. The button is not visible if the product is subscription-only.

Minicart and cart

The product is added to the cart, and the checkout starts with a new basket created exclusively for this product. Once the checkout is completed, the old basket is restored based on a session attribute.

The Express Checkout button is not displayed in the minicart if there are only subscription-only products in the cart.

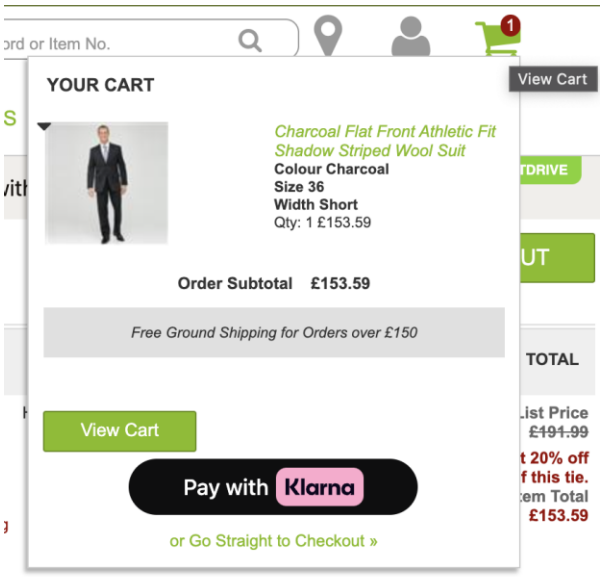


Figure 33. Klarna Express checkout on minicart

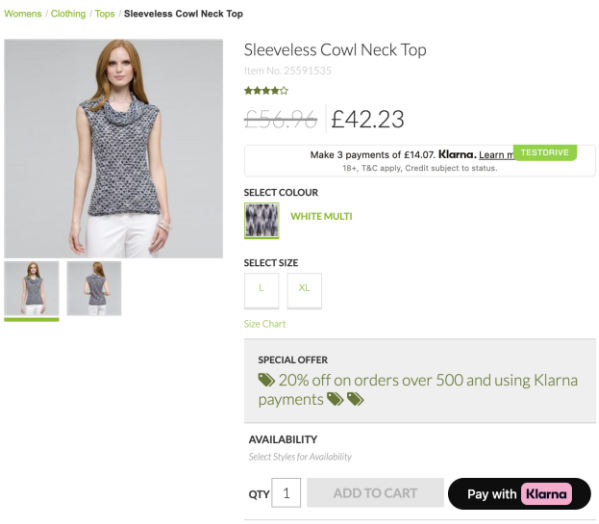


Figure 34. Klarna Express checkout on PDP

4. Implementation Guide

4.1. Setup of Business Manager

The Klarna Cartridge contains two cartridges required for full functionality. Controller and SFRA support are separated into two distinct cartridges, facilitating the installation and use of either model:

- `int_klarna_payments`: Implements the core storefront functionality.
- `int_klarna_payments_controllers`: Implements the storefront functionality with SG code.

4.1.1. Cartridge Upload & Assignment

Step-by-Step Instructions

1) **Import the `int_klarna_payments` cartridge into the SCC Studio Workspace:**
    a) Open SCC Studio.
    b) Click **File** -> **Import** -> **General** -> **Existing Projects into Workspace**.
    c) Browse to the directory where you saved the `int_klarna_payments` cartridge.
    d) Click **Finish**.
    e) Click **OK** when prompted to link the cartridge to the sandbox.

2) **Import the `int_klarna_payments_controllers` cartridge into the SCC Studio Workspace:**
    a) Open SCC Studio.
    b) Click **File** -> **Import** -> **General** -> **Existing Projects into Workspace**.
    c) Browse to the directory where you saved the `int_klarna_payments_controllers` cartridge.

d) Click **Finish**.
e) Click **OK** when prompted to link the cartridge to the sandbox.

3) **Prepend the Klarna cartridges to the effective site cartridge path:**
   a) Log into the SCC Business Manager.
   b) Click **Administration** -> **Sites** -> **Manage Sites**.
   c) Select the desired site.
   d) Click on the **Settings** tab.
   e) Prepend `int_klarna_payments_controllers:int_klarna_payments` to the **Cartridges** field.
   f) Click **Apply**.

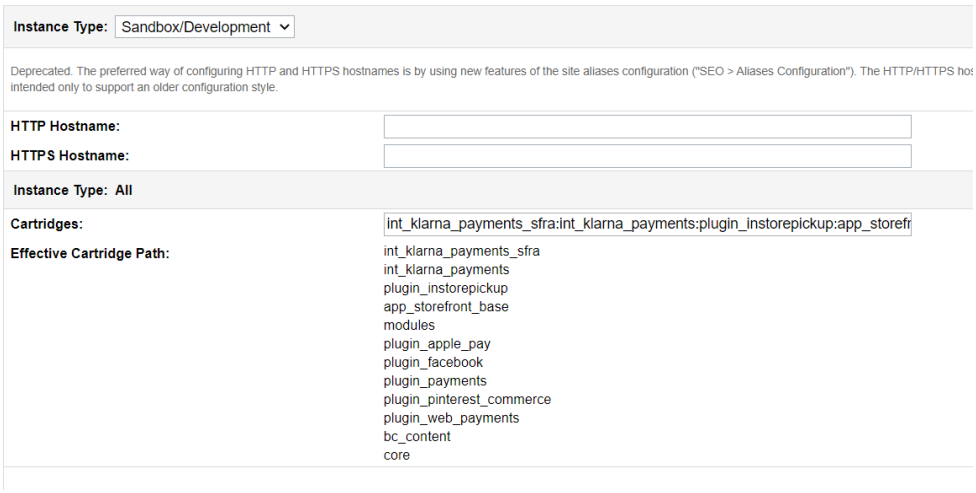


Figure 37. Effective Cartridge Path

4.1.2. Metadata Import

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

Note: Review the default `service.xml` file in the `site-template.zip` and update the configuration for Playground and Production accordingly before importing.

4.2. Configuration

4.2.1. Configure Klarna Activation Site Preferences for single Klarna API credentials per site

Steps:

1) **Log into the SCC Business Manager**.
2) **Select the Desired Site**: Choose the site from the tabs across the top of the page.
3) **Access Custom Preferences**: Click **Site Preferences** -> **Custom Preferences** -> **Klarna Activation**
4) **Update all the fields:**
    a) Enter `client_id` to activate OSM and KEC.
    b) API Username and password.
    c) Select region.
    d) Select markets on which Klarn should be available.
    e) Select environment - *yes* for playground, *no* for production.

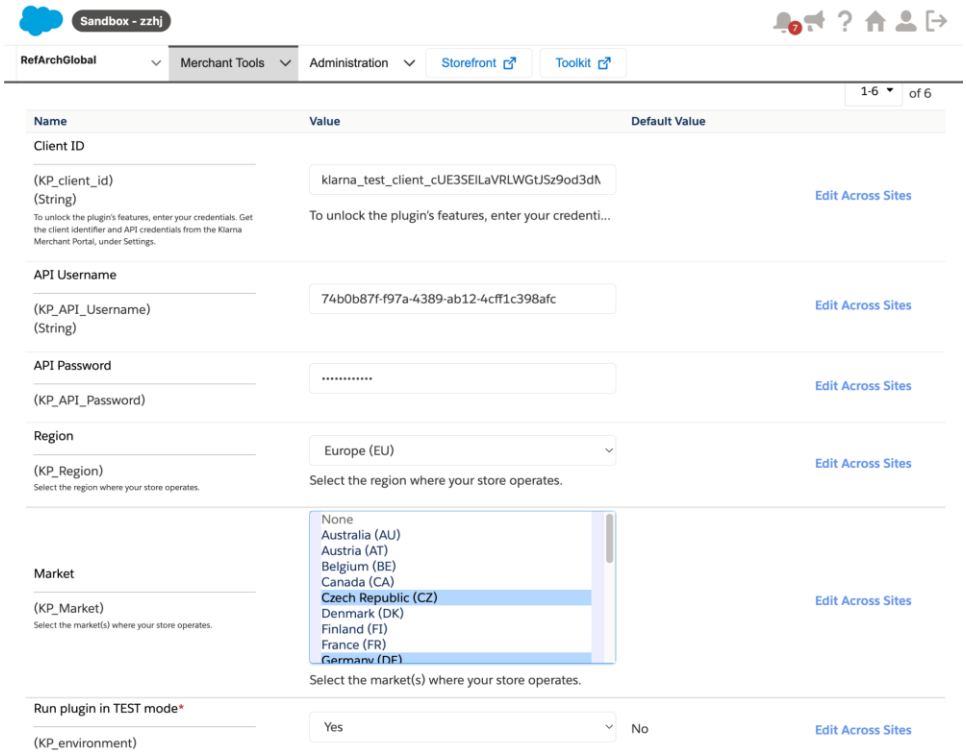


Figure 38. Klarna activation settings

4.2.2. Configure Klarna Activation Custom Object for multiple Klarna API credentials on one site

Steps:

1) **Log into the SCC Business Manager**.
2) **Select the Desired Site**: Choose the site from the tabs across the top of the page.
3) **Access Custom Objects**: Click **Custom Objects** -> **Custom Object Editor**.
4) **Change Object Type**: Set the Object Type dropdown to "KlarnaActivation".
5) **Create Object:** enter Klarna activation key (something meaningful to you).
6) **Enter Fields**: Enter the required fields as mentioned in the "KlarnaActivation" section.
7) **Repeat**: Repeat for the other regions or markets as necessary.

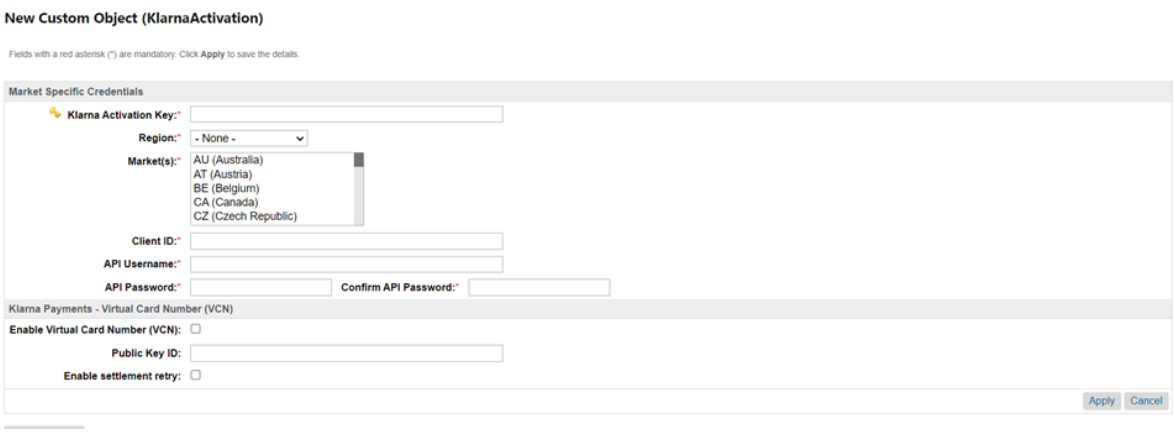


Figure 39. Klarna activation custom object settings

4.2.3. Add Account Settings to KlarnaCountries Custom Objects

This functionality is deprecated as of release 24.4.0

This configuration has been deprecated with release 24.4.0 and will be removed in the future. Use sections 4.2.1 and 4.2.2 for the updated configuration.

Steps:

1) **Log into the SCC Business Manager**.
2) **Select the Desired Site**: Choose the site from the tabs across the top of the page.
3) **Access Custom Objects**: Click **Custom Objects** -> **Custom Object Editor**.
4) **Change Object Type**: Set the Object Type dropdown to “KlarnaCountries”.
5) **Find and Edit**: Click the **Find** button. Select the desired country to edit (see screenshot below).
6) **Update Fields**: Update the required fields as mentioned in the “KlarnaCountries” section.
7) **Repeat**: Repeat for the other countries as necessary.

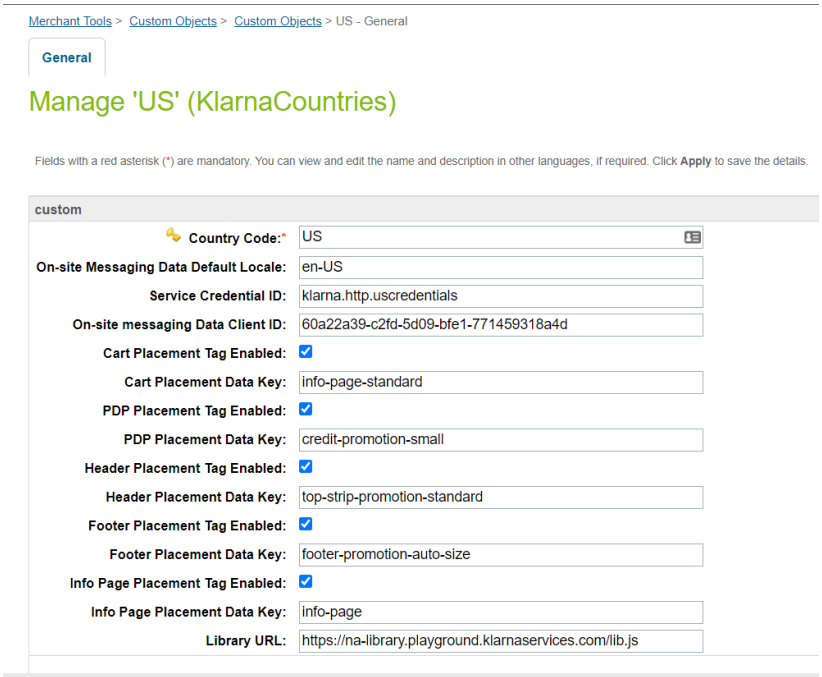


Figure 40. KlarnaCountries Settings

4.2.4. Configure Klarna Payment Custom Preferences

This functionality is deprecated as of release 24.4.0

This configuration has been deprecated with release 24.4.0 and will be removed in the future. Use sections 4.2.1 and 4.2.2 for the updated configuration.

Steps:

1) **Log into the SCC Business Manager**.
2) **Select the Desired Site**: Choose the site from the tabs across the top of the page.
3) **Access Custom Preferences**: Click **Site Preferences** -> **Custom Preferences** -> **KlarnaPayment**.
4) **Fill Out Settings**: Complete the settings as desired. Descriptions of the site preferences are in the Site Preferences section.

4.2.5. Configure Klarna Payment Service

This functionality is deprecated as of release 24.4.0

This configuration has been deprecated with release 24.4.0 and will be removed in the future. Use sections 4.2.1 and 4.2.2 for the updated configuration.

Steps:

1) **Log into the SCC Business Manager.**
2) **Navigate to Services:** Click Administration -> Operations -> Services.
3) **Access Credentials:** Click the Credentials tab. Each Klarna credential corresponds to one of the KlarnaCountries custom objects. Click on the one you want to edit.
4) **Enter Credentials:** Enter the MID API username and API password received from Klarna.
5) **Edit URL Field:** Update the URL field if using the Production environment. For Klarna API URLs, refer to Klarna API URLs.

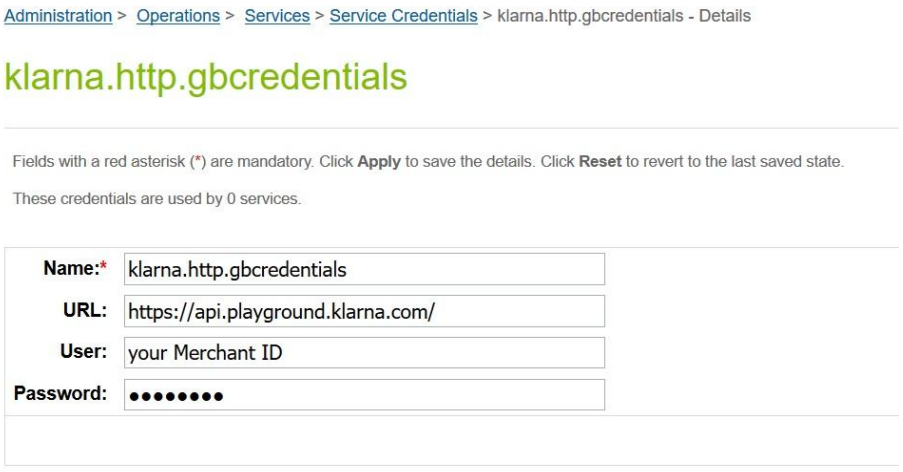


Figure 41. Service Settings

4.2.6. Configure Klarna Rate Limited Service Profile

This functionality is deprecated as of release 24.4.0

- **Rate limits are managed by Klarna.** The plugin handles the errors in case these are exceeded.
- Default service `klarna.http.defaultendpoint` will be used for all API payments calls.
- For more information refer to the Klarna technical documentation here.

Steps:

1) **Log into the SCC Business Manager**.
2) **Navigate to Klarna Payments Preferences**: Go to *Merchant Tools -> Site Preferences -> Custom Site Preferences Group -> Klarna Payments*.
3) **Enable Service Limit Configuration**: Set **Rate Limit By Operation** to **Yes**. If set to **No**, the default service will control the rate limit.

4.2.7. Configure Custom Rate Limits

This functionality is deprecated as of release 24.4.0

Rate limits are managed by Klarna. The plugin handles the errors in case these are exceeded.

Steps:

1) **Log into the SCC Business Manager**.
2) **Navigate to Services**: Go to **Administration** -> **Operations** -> **Services**.
3) **Select Required Profile**: Choose a required profile (e.g., Klarna.http.createSession).
4) **Enable Rate Limit**: Check the **Enable Rate Limit** box.
5) **Set Rate Limit**:

a) **Max Rate Limit Calls**: Set to 50 (example: Higher rate limit of 50 requests/sec agreed with Klarna).
b) **Rate Limit Interval (milliseconds)**: Set to 1000.

6) **Repeat**: Repeat for all service operations with respective agreed rate limits.

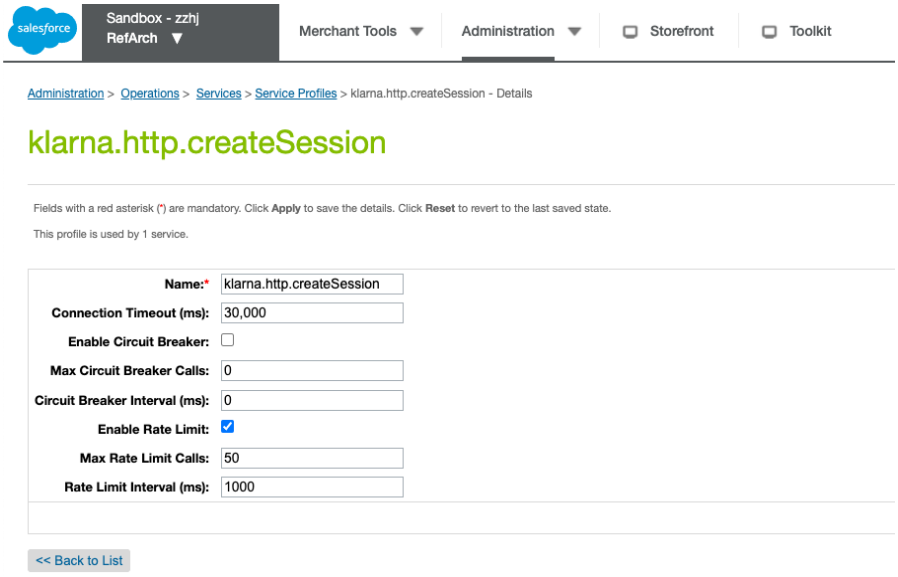


Figure 42. Example of create session rate limit profile

4.3. Extended Controllers

Controller	Start Node	Remarks
`Checkout.js`	`Begin`	Extended to call Klarna session manager
`CheckoutServices.js`	`Get`, `SubmitPayment`, `PlaceOrder`	Klarna payment method/category and totals are being stored
`CheckoutShippingServices.js`	`SubmitShipping`, `ToggleMultiShipping`	Calling the Klarna session manager
`Order.js`	`Confirm`	Extending Klarna order data to view data

Table 3. Extended Controllers List

4.4. 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.

4.5. Jobs

4.5.1. Job "OrderCleanUp" (Optional)

Note: This one-time clean-up job is applicable only to merchants integrated with Klarna Payments cartridge version earlier than 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 the attribute `custom.kpIsVCN=true` to remove sensitive details saved in fields `kpVCNPAN`, `kpVCNCSC`, `kpVCNExpirationMonth`, and `kpVCNExpirationYear` from previous releases. No parameters are passed to the script.

Upon successful run, the job logs the result of processed orders in the custom debug log located in `webdav/Sites/Logs`. You will receive a message indicating the processed orders count for each storefront or a message indicating that there are no orders needing update.

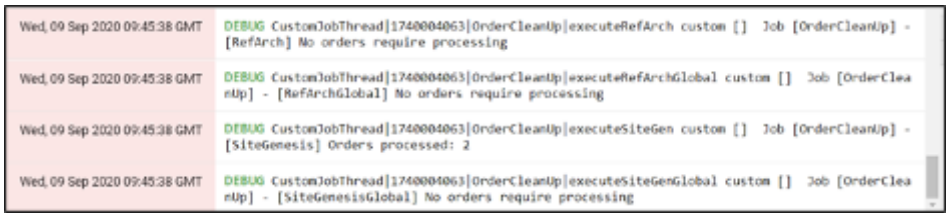


Figure 43. Job Logs

In case of an error, the cause of the failure (message and stack trace) will be logged in the standard error log.

Figure 44. Job Logs

Steps to Setup the Job

1. **Import the Job Configuration**: Navigate to **Administration > Operations > Import & Export**. Import the file `jobs.xml`.

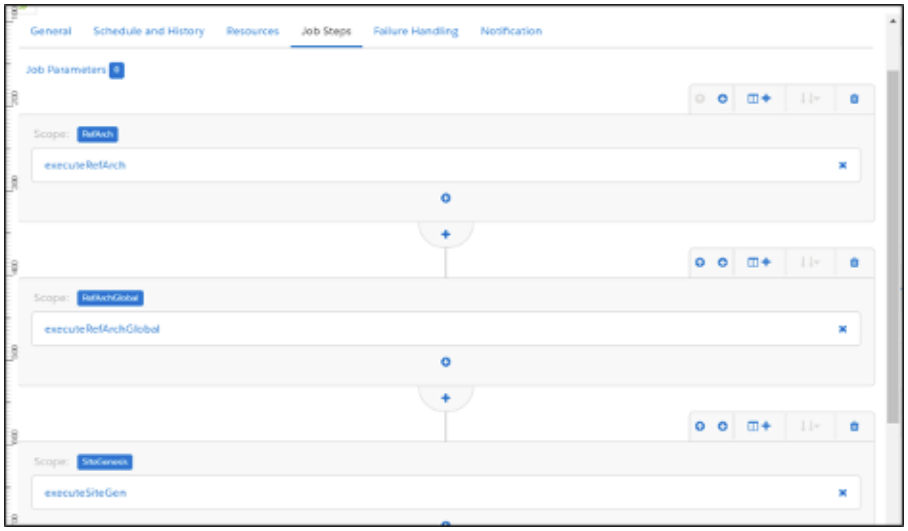


Figure 45. Job Steps

> **Note:** The XML file includes only the RefArch scope by default, but it can be configured with multiple flows if you have more than one site using this functionality. Each site should be added as a separate flow.

2. **Add a Sequential Flow**: Click on the **Add a sequential flow** button at the bottom of the current flow.

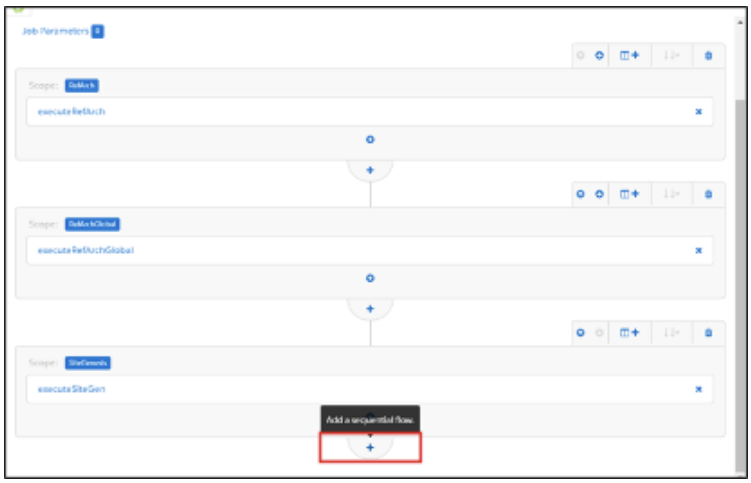


Figure 46. Add New Job Step

3. **Configure the Step**:
    - Click on the **Configure a step** button within the flow.
    - In the flyout, search for "script" and select **ExecuteScriptModule**.

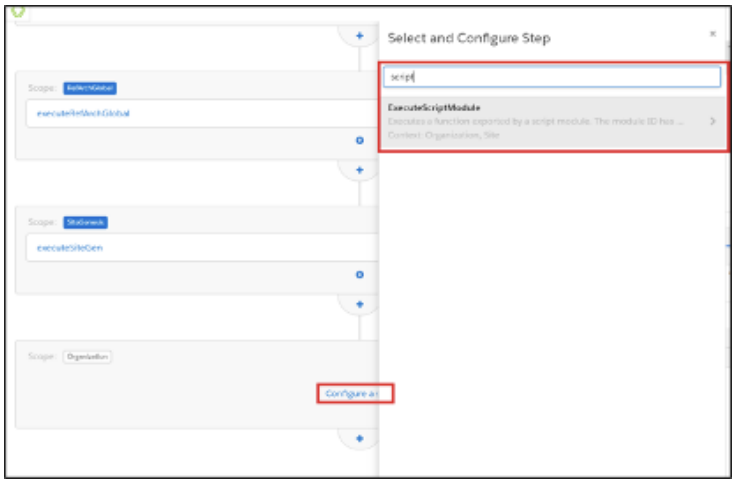


Figure 47. Configure Step

4. **Populate Step Fields**: In the flyout, populate these fields and click the **Assign** button.
    - **ID**: Enter any meaningful name. If you have multiple flows, ensure this name is unique. Duplicate names will not save, and an error message will appear.
    - **ExecuteScriptModule.Module**: Enter the location of the `OrderCleanUpJob.js` file. By default, it should be `int_klarna_payments/cartridge/scripts/job/OrderCleanUpJob.js` or the location where you have placed the script file.
    - **ExecuteScriptModule.FunctionName**: Leave this field value as `execute`.

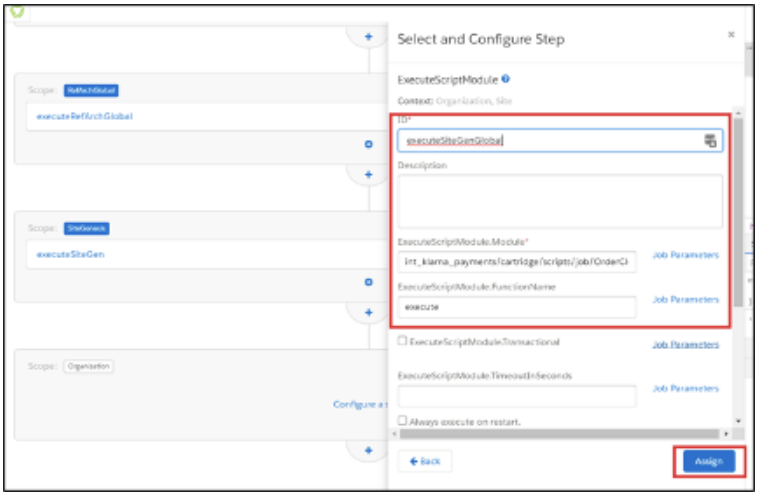


Figure 48. Configure Step (cont.)

5. **Assign to Correct Site Scope:**
    - Click on **Organization** and select **Specific Sites** from the drop-down.
    - From the list of sites, select the respective site ID (e.g., `SiteGenesisGlobal`) and click on **Assign**.

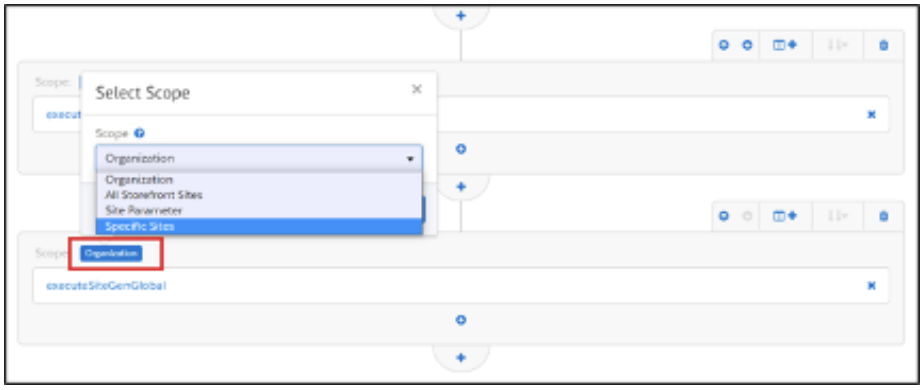


Figure 49. Job Scope

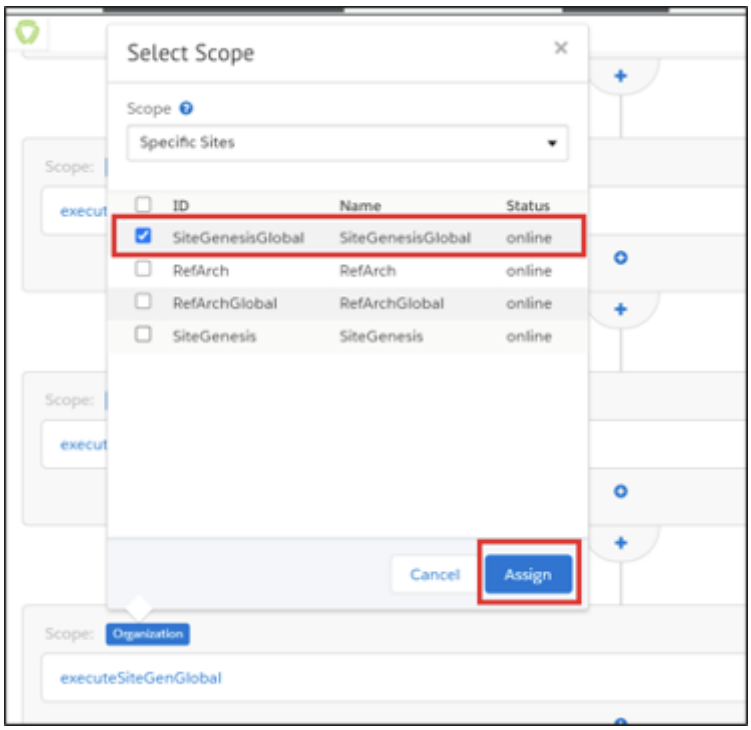


Figure 50. Job Scope (cont.)

Repeat steps 1-5 for each site/storefront that you have using Klarna VCN and need additional configuration. This ensures that the clean-up job runs for all relevant sites and removes sensitive data as required.

4.5.2. Job "RecurringOrders"

The `RecurringOrders` job is designed to process subscription entries for all customers. The job performs the following functions:

- **Eligibility Check**: It verifies that the subscription is enabled and that either the `nextChargeDate` or `nextRetryDate` matches the current date.
- **Order Creation**: It creates new Salesforce Commerce Cloud (SFCC) orders for eligible subscriptions, replacing the old ones.
- **Trial Period Handling**: It processes orders with expiring trial periods for charges.

Configuration

By default, the job is set to run on the **RefArch** site, as specified in the `jobs.xml` file. This setting can be modified either in the `jobs.xml` file or through the storefront configuration.

The job consists of a single step, `createOrder`, with the following configuration:

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

The job operates at site level.

Ensure your configuration matches these details to maintain the proper functionality of the RecurringOrders job.

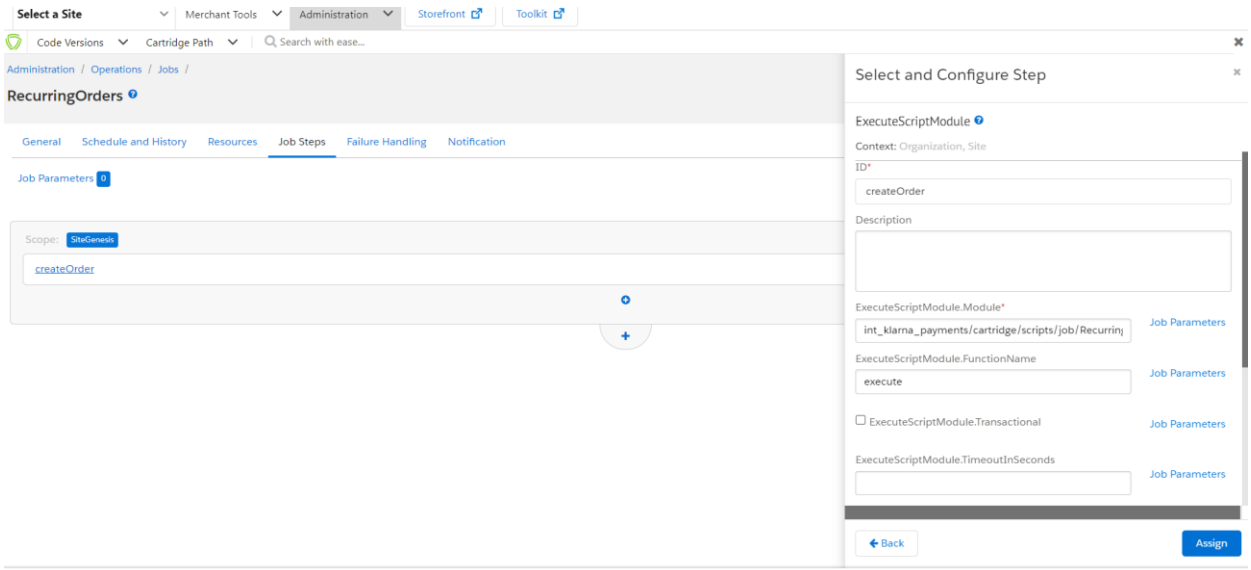


Figure 51. RecurringOrdersJobs

4.6. 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.

4.6.1. 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
- js/pages/cart.js 79
- default/checkout/shipping/singleshipping.isml 82
- scripts/cart/ValidateCartForCheckout.js 83
- scripts/util/Resource.ds 84
- js/pages/account.js 84
- default/account/orders.isml 87
- default/checkout/components/minicheckout_address.isml 87
- js/minicart.js

4.6.1.1. default/checkout/summary/summary.isml

Add the following code before the closing </isdecorate> tag at the end of the file:

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

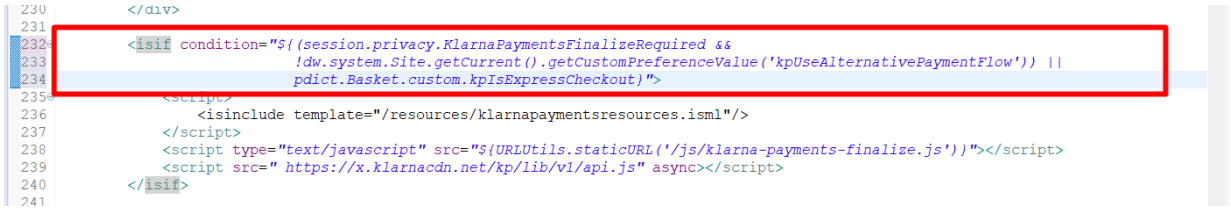
```
<isif condition="${!session.privacy.KlarnaPaymentsFinalizeRequired &&
dw.system.Site.getCurrent().getCustomPreferenceValue('kpUseAlternativePaymentFlow'
)}">
    <isinclude template="klarnapayments/modules.isml"/>
    <isset name="billingAddress" value="${pdict.Basket.billingAddress}"
scope="page"/>
    <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 defer src="${URLUtils.staticURL('/js/klarna-payments-alternative-
flow.js')}"></script>
    <script defer src="https://x.klarnacdn.net/kp/lib/v1/api.js"></script>
</isif>
```

```
226
227                 </fieldset>
228             </form>
229
230         </div>
231
232         <isif condition="${(session.privacy.KlarnaPaymentsFinalizeRequired && !dw.system.Site.getCurrent().getCustomPreferenceValue('kpUs
233             <script>
234                 <isinclude template="/resources/klarnapaymentsresources.isml"/>
235             </script>
236             <script defer src="${URLUtils.staticURL('/js/klarna-payments-finalize.js')}"></script>
237             <script defer src="https://x.klarnacdn.net/kp/lib/v1/api.js"></script>
238         </isif>
239
240         <isif condition="${!session.privacy.KlarnaPaymentsFinalizeRequired && dw.system.Site.getCurrent().getCustomPreferenceValue( 'kpUs
241             <isinclude template="klarnapayments/modules.isml"/>
242             <isset name="billingAddress" value="${pdict.Basket.billingAddress}" scope="page"/>
243             <iskpbillingaddresshelper p_address="${billingAddress}" p_email="${pdict.Basket.customerEmail}"/>
244
245             <isloop items="${JSON.parse(session.privacy.KlarnaPaymentMethods)}" var="klarnaPaymentMethod">
246                 <div class="payment-method" data-test="${session.privacy.KlarnaPaymentsFinalizeRequired}" data-method="${'klarna_payments
247                     <div id="${'klarna_payments_' + klarnaPaymentMethod.identifier + '_container'}" style="text-align: center;"></div>
248                     <isif condition="${empty(pdict.Basket.custom.kpSessionId)}">
249                         <div class="klarna_payments_error" style="text-align: center; font-weight: bold; color: red;"><isprint value="${K
250                     </isif>
251                 </div>
252             </isloop>
253
254             <script>
255                 <isinclude template="/resources/klarnapaymentsresources.isml"/>
256             </script>
257             <script defer src="${URLUtils.staticURL('/js/klarna-payments-alternative-flow.js')}"></script>
258             <script defer src="https://x.klarnacdn.net/kp/lib/v1/api.js"></script>
259         </isif>
260
261 </isdecorate>
262
```

Figure 52. Modifications in summary.isml

Update the condition on line 232 for the finalize script include:

```
<isif condition="${(session.privacy.KlarnaPaymentsFinalizeRequired &&
!dw.system.Site.getCurrent().getCustomPreferenceValue('kpUseAlternativePaymentFlow
')) || pdict.Basket.custom.kpIsExpressCheckout}">
```



```
230         </div>
231
232         <isif condition="${(session.privacy.KlarnaPaymentsFinalizeRequired &&
233                             !dw.system.Site.getCurrent().getCustomPreferenceValue('kpUseAlternativePaymentFlow')) ||
234                             pdict.Basket.custom.kpIsExpressCheckout)">
235             <script>
236                 <isinclude template="/resources/klarnapaymentsresources.isml"/>
237             </script>
238             <script type="text/javascript" src="${URLUtils.staticURL('/js/klarna-payments-finalize.js')}"></script>
239             <script src=" https://x.klarnacdn.net/kp/lib/v1/api.js" async></script>
240         </isif>
241
```

Figure 53. Modifications in summary.isml (cont.)

Note: Ensure the updated condition for the finalize script is applied correctly to handle both the finalize and express checkout scenarios.

4.6.1.2. default/checkout/billing/billing.isml

Add the following code before the closing `</isdecorate>` tag at the end of the file, :

```
<script>
    <isinclude template="/resources/klarnapaymentsresources.isml"/>
</script>
<script defer src="${URLUtils.staticURL('/js/klarna-payments.js')}"></script>
<script defer src="https://x.klarnacdn.net/kp/lib/v1/api.js"></script>
<link rel="stylesheet" href="${URLUtils.staticURL('/css/klarna-payments.css')}" />
```

```
218     var countries = ViewHelpers.getCountriesAndRegions(addressForm);
219     var json = JSON.stringify(countries);
220 </isscript>
221 <script>window.Countries = <isprint value="${json}" encoding="off"/>;</script>
222
223 <script><isinclude template="/resources/klarnapaymentsresources.isml"/></script>
224 <script defer src="${URLUtils.staticURL('/js/klarna-payments.js')}"></script>
225 <script defer src="https://x.klarnacdn.net/kp/lib/v1/api.js"></script>
226 <link rel="stylesheet" href="${URLUtils.staticURL('/css/klarna-payments.css')}" />
227
228 </isdecorate>
229
```

Figure 54. Modifications in billing.isml

4.6.1.3. default/checkout/billing/paymentmethods.isml

Add the following code after `<div class="form-row label-inline` close to line 18 as shown below:

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

```
9       <div class="dialog-required"> <span class="required-indicator">&#8226; <em>${Resource.msg('global.requiredfield','locale',null)}</em></sp
10      </legend>
11
12      <div class="payment-method-options form-indent">
13          <isloop items="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.options}" var="paymentMethodType">
14
15              <iscomment>Ignore GIFT_CERTIFICATE method, GCs are handled separately before other payment methods.</iscomment>
16              <isif condition="${paymentMethodType.value.equals(dw.order.PaymentInstrument.METHOD_GIFT_CERTIFICATE)}"><iscontinue/></isif>
17
18              <div class="form-row label-inline <isif condition="${paymentMethodType.value === 'Klarna'}">hide</isif>">
19                  <isset name="radioID" value="${paymentMethodType.value}" scope="page"/>
20                  <div class="field-wrapper">
21                      <input id="is-${radioID}" type="radio" class="input-radio" name="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentI
22                  </div>
23                  <label for="is-${radioID}"><isprint value="${Resource.msg(paymentMethodType.label,'forms',null)}"/><isif condition="${!empty(dw.
24              </div>
25
26          </isloop>
```

Figure 55. Modifications in paymentmethods.isml

Add the following code after the `</isloop>` tag, close to line 28:

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

paymentmethods.isml

```
        <legend>
            ${Resource.msg('billing.paymentheader','checkout',null)}
            <div class="dialog-required"> <span class="required-indicator">&#8226; <em>${Resource.msg('global.requiredfield','locale',null)}
        </legend>

        <div class="payment-method-options form-indent">
            <isloop items="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.options}" var="paymentMethodType">

                <iscomment>Ignore GIFT_CERTIFICATE method, GCs are handled separately before other payment methods.</iscomment>
                <isif condition="${paymentMethodType.value.equals(dw.order.PaymentInstrument.METHOD_GIFT_CERTIFICATE)}"><iscontinue/></isif>

                <div class="form-row label-inline <isif condition="${paymentMethodType.value === 'Klarna'}">hide</isif>">
                    <isset name="radioID" value="${paymentMethodType.value}" scope="page"/>
                    <div class="field-wrapper">
                        <input id="is-${radioID}" type="radio" class="input-radio" name="${pdict.CurrentForms.billing.paymentMethods.selecte
                    </div>
                    <label for="is-${radioID}"><isprint value="${Resource.msg(paymentMethodType.label,'forms',null)}"/><isif condition="${!e
                </div>

            </isloop>

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

        </div>

        <div class="form-row form-row-button">
```

Figure 56. Modifications in paymentmethods.isml (cont.)

Add the following code before the closing `</fieldset>` tag, close to line 150:

```
<iscomment>
    Klarna Payments
    ----------------------------------------------------------
</iscomment>
<isinclude template="klarnapayments/klarnapaymentblock.isml"/>
```

```
        <iscomment>
            Custom processor
            ----------------------------------------------------------
        </iscomment>

        <div class="payment-method <isif condition="${!empty(pdict.selectedPaymentID) && pdict.selectedPaymentID==
            <!-- Your custom payment method implementation goes here. -->
            ${Resource.msg('billing.custompaymentmethod','checkout',null)}
        </div>

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

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

    </fieldset>
<iselse/>
    <div class="gift-cert-used form-indent">
        <isif condition="${pdict.gcPITotal>0}">${Resource.msg('billing.giftcertnomethod','checkout',null)}<iselse/
        <input type="hidden" name="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlName}"
```

Figure 57. Modifications in paymentmethods.isml (cont.)

4.6.1.4. default/checkout/shipping/minishipments.isml

To integrate Klarna Payments into the `minishipments.isml` file, follow these steps:

Add the following code at the beginning of the `minishipments.isml` file. This step includes the necessary Klarna Payments modules for the mini shipments template.

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

minishipments.isml

```
1  <iscontent type="text/html" charset="UTF-8" compact="true"/>
2  <isinclude template="util/modules.isml"/>
3  <isinclude template="klarnapayments/modules.isml"/>
4
5  <iscomment>
6      This template renders a summary of all shipments of the basket which is
7      used below the order summary at the right hand side in the checkout
8      process.
9  </iscomment>
10 <isset name="Shipments" value="${pdict.Basket.shipments}" scope="page"/>
11
```

Figure 58. Modifications in minishipments.isml

After the line containing `<isminicheckout_address p_address="${shipment.shippingAddress}"/>`, add the code below. This step adds the Klarna Payment Address Helper, which manages the shipment and shipping address.

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

```
44             <div class="details">
45                 <iscomment>
46                     render the detail section depending on whether this is a physical shipment with products
47                     (shipped to an address) or if this is a gift certificate (send via email)
48                 </iscomment>
49                 <isif condition="${shipment.giftCertificateLineItems.size() > 0}">
50                     <isloop items="${shipment.giftCertificateLineItems}" var="giftCertLI">
51                         <div><isprint value="${giftCertLI.recipientName}"/></div>
52                         <div>(<isprint value="${giftCertLI.recipientEmail}"/>)</div>
53                     </isloop>
54                 <iselseif condition="${shipment.shippingAddress != null && pdict.Basket.productLineItems.size() > 0}">
55                     <isminicheckout_address p_address="${shipment.shippingAddress}"/>
56                     <iskpaddresshelper p_shipment="${shipment}" p_address="${shipment.shippingAddress}"/>
57
58                     <isif condition="${!empty(shipment.shippingMethod)}">
59                         <div class="minishipments-method">
60                             <span>${Resource.msg('order.orderdetails.shippingmethod','order',null)}</span>
61                             <span><isprint value="${shipment.shippingMethod.displayName}"/></span>
62                         </div>
63                     </isif>
```

Figure 59. Modifications in minishipments.isml (cont.)

4.6.1.5. default/components/header/header.isml

Add the following code before the `<script src="${URLUtils.staticURL('/js/app.js')}"></script>` script tag:

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

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

Figure 60. Modifications in footer_UI.isml

4.6.1.6. default/components/footer/footer.isml

Add the following code right after the `</footer>` end tag and before the `<iscontentasset aid="footer-copy"/>` tag:

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

This includes the Klarna OSM footer and KEB form in the footer template.

Figure 61. Modifications in footer.isml

4.6.1.7. default/components/footer/footer_UI.isml

Add the following code right under the pricing template include:

```
<isinclude template="klarnapayments/modules.isml"/>
<iskosmpdp p_product="${psProduct}" />
```

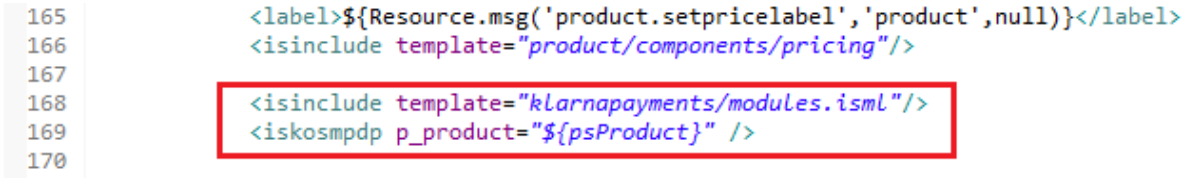


Figure 62. Modifications in producttopcontentPS.isml

4.6.1.8. default/product/producttopcontentPS.isml

Add the following code right under the pricing template include:

```
<isif condition="${!isQuickView}">
    <isinclude template="klarnapayments/modules.isml"/>
    <iskosmpdpp_product="${pdict.Product}" />
</isif>
```

```
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 63. Modifications in producttopcontent.isml

Add the following JavaScript code on line 24 within the `<isscript>` element:

```
var isStandardProduct = !empty(product.custom.kpIsStandardProduct) ?
product.custom.kpIsStandardProduct : true;
var isSubscriptionProduct = !empty(product.custom.kpIsSubscriptionProduct) ?
product.custom.kpIsSubscriptionProduct : false;
var isSubscriptionOnly = (isSubscriptionProduct && !isStandardProduct);
```

```
19
20          let availableCount = "0";
21          if (pdict.isProductAvailable && !empty(avm.inventoryRecord)) {
22              availableCount = avm.inventoryRecord.perpetual ? "999" : avm.inventoryRecord.ATS.value.toFixed().toString();
23          }
24          var isStandardProduct = !empty(product.custom.kpIsStandardProduct) ? product.custom.kpIsStandardProduct : true;
25          var isSubscriptionProduct = !empty(product.custom.kpIsSubscriptionProduct) ? product.custom.kpIsSubscriptionProduct : false;
26          var isSubscriptionOnly = (isSubscriptionProduct && !isStandardProduct);
27      </isscript>
28      <iscomment>
29          primary details
```

Figure 64. Modifications in productcontent.isml

Add the following code on line 256, right after the "Add to Cart" button:

```
<isif condition="${!isSubscriptionOnly}">
    <iskecpdp />
</isif>
```

```
252                     var buttonTitleDisabledSelectVariation = StringUtils.format(Resource.ms
253                 </isscript>
254                 <button id="add-to-cart" type="button" title="${buttonTitleDisabledSelectVa
255             </isif>
256             <isif condition="${!isSubscriptionOnly}">
257                 <iskecpdp />
258             </isif>
259             </div><!-- end details block -->
260         </fieldset>
```

Figure 65. Modifications in poductcontent.isml (cont.)

4.6.1.9. default/product/productcontent.isml

Add the following code in the variation update callback on line 35:

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

```
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 66. Modifications in variant.js

Add the following code in the variation update callback on line 37:

```
if (window.klarnaExpressCheckout) {
window.klarnaExpressCheckout.klarnaExpressCheckoutPDP(); }
```

```
            }
            image.replaceImages();
            tooltip.init();
            if (window.Klarna && window.Klarna.OnsiteMessaging) {
                window.Klarna.OnsiteMessaging.refresh();
            }
            if (window.klarnaExpressCheckout) {
                window.klarnaExpressCheckout.klarnaExpressCheckoutPDP();
            }
        }
    });
};

module.exports = function () {
```

Figure 67. Modifications in variant.js (cont.)

4.6.1.10. default/product/productcontent.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 68. Modifications in cart.isml

Add the following code on line 823, right after the `<isordertotals>` tag:

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

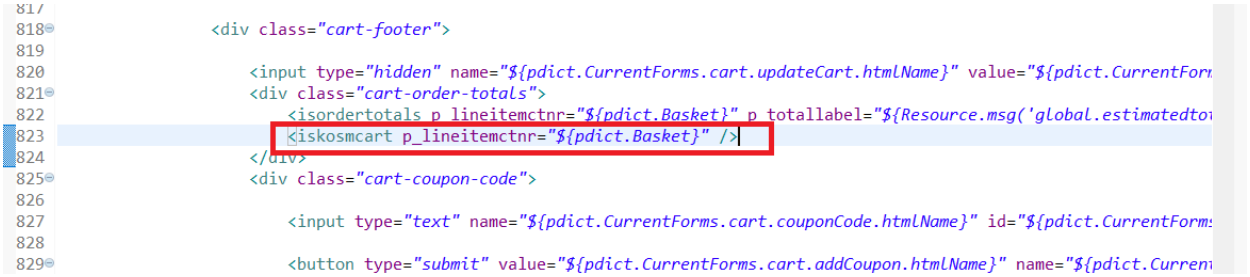


Figure 69. Modifications in cart.isml (cont.)

Add the following code on lines 33 and 864, right after the `<isif condition="${enableCheckout}">` tag:

```
<iskebcart />
```

Figure 70. Modifications in cart.isml (cont.) line 33

Figure 71. Modifications in cart.isml (cont.) line 864

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 the `deliveryoptions` include:

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

Figure 72. Modifications in cart.isml (cont.) line 182

Add the following code on line 65, inside the div with class `error-form`:

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

```
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 73. Modifications in cart.isml (cont.) line 65

Add the following code on line 862, at the end of the div with class `cart-footer`:

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

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

Figure 74. Modifications in cart.isml (cont.) line 862

Add the following code on line 34, before the checkout button:

```
<iskeccart container_id="klarnaExpressCheckout"/>
```

```
30          <form class="cart-action-checkout" action="${URLUtils.httpsContinue()}" method="pos
31              <fieldset>
32                  <isif condition="${enableCheckout}">
33                      <iskebcart />
34                      <iskeccart container_id="klarnaExpressCheckout"/>
35                      <button class="button-fancy-large" type="submit" value="${Resource.msg
36                          ${Resource.msg('global.checkout','locale',null)}
37                      </button>
38                      <isapplepay></isapplepay>
39                  <iselse/>
40                      <button class="button-fancy-large" disabled="disabled" type="submit" va
41                          ${Resource.msg('global.checkout','locale',null)}
42                      </button>
43                  </isif>
44              </fieldset>
45          </form>
```

Figure 75. Modifications in cart.isml (cont.) line 34

Add the following code on line 879, in the `enabledCheckout` case before the checkout button:

```
874            <iscomment>continue shop url is a non-secure but checkout needs a secure and that is why s
875            <form class="cart-action-checkout" action="${URLUtils.httpsContinue()}" method="post" name
876                <fieldset>
877                    <isif condition="${enableCheckout}">
878                        <iskebcart />
879                        <iskeccart container_id="klarnaExpressCheckoutBottom"/>
880                        <button class="button-fancy-large" type="submit" value="${Resource.msg('global
881                            ${Resource.msg('global.checkout','locale',null)}
882                        </button>
883                        <isapplepay></isapplepay>
884                    <iselse/>
885                        <button class="button-fancy-large" disabled="disabled" type="submit" value="$
886                            ${Resource.msg('global.checkout','locale',null)}
887                        </button>
```

Figure 76. Modifications in cart.isml(cont.) line 879

This includes the Klarna Express Checkout component for the enabled checkout case.

4.6.1.11. default/checkout/cart/cart.isml

Add the following code at the end of the file:

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

```
41          <iscomment>Country Selector</iscomment>
42          <isinclude template="components/header/countryselector"/>
43
44      </nav>
45
46      <iscomment>INCLUDE: Mini-cart, do not cache</iscomment>
47      <div id="mini-cart">
48          <isinclude url="${URLUtils.url('Cart-MiniCart')}"/>
49      </div>
50
51  </div><!-- /header -->
52
53  <!-- Klarna OSM header -->
54  <isinclude template="klarnapayments/modules.isml"/>
55  <iskosmheader />
56  <!-- /Klarna OSM header -->
57
```

Figure 77. Modifications in header.isml

4.6.1.12. default/mail/orderconfirmation.isml

Add the following code before the closing `</table>` tag:

```
<tr>
    <td style="font-size:12px;font-family:arial;padding:20px 10px;vertical-
align:top;">
        <isset name="confirmationAsset"
value="${require('*/cartridge/scripts/util/klarnaHelper').getConfirmationEmailAsse
t()}" scope="page" />
        <isprint value="${confirmationAsset}" encoding="off" />
    </td>
</tr>
```

This includes the Klarna confirmation email asset in the order confirmation email.

Figure 78. Modifications in orderconfirmation.isml

4.6.1.13. default/components/order/ordrdetailsemail.isml

Add the following code before the closing `</isif>` tag on line 51:

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

This conditionally includes the Klarna order reference in the order details email if the payment method is Klarna.

Figure 79. Modifications in orderdetailsemail.isml

4.6.1.14. default/checkout/cart/minicart.isml

Add the following code before the `<a class="mini-cart-link-checkout"...` tag on line 74:

```
<isset name="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>
```

This sets the Klarna OSM variable, checks if the mini cart express button is enabled, and includes the necessary Klarna templates and scripts.

Figure 80. Modifications in minicart.isml

Add the following code before the `<a class="mini-cart-link-checkout"...` tag on line 81:

```
<isif condition="${KlarnaOSM.isKlarnExpressCheckoutEnabled() &&
KlarnaOSM.showExpressCheckoutButton().miniCart}">
    <isset name="SubscriptionHelper"
value="${require('*/cartridge/scripts/subscription/subscriptionHelper')}"
scope="page" />
    <isif condition="${!SubscriptionHelper.hasSubscriptionOnly()}">
        <isinclude template="klarnapayments/modules.isml" />
        <iskecminicart />
    </isif>
</isif>
```

This conditionally includes the Klarna Express Checkout button in the mini cart based on specific conditions.

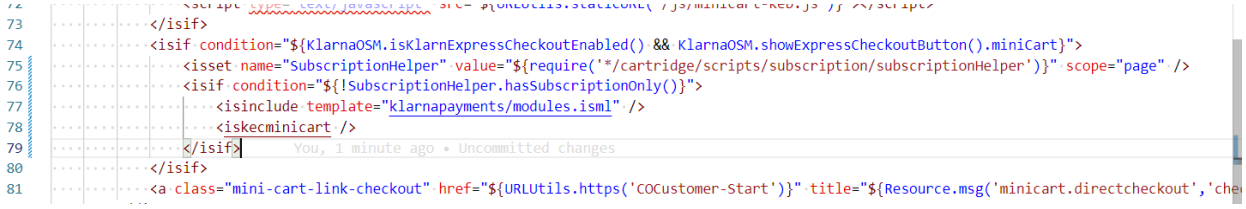


Figure 81. Modifications in minicart.isml (cont.)

4.6.1.15. js/pages/cart.js

Add the following code after line 6:

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

This imports the `util` module for use in the cart script.

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

Figure 82. Modifications in cart.js

Add the following code at the end of the `initializeEvents()` function:

```
$('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;
            }
```

```
        }
    });
});
```

This handles changes to the subscription checkbox.

```
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 83. 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;
            }
        }
    });
});
```

This handles changes to the subscription period and frequency dropdowns.

```
75    $('body').on('change', '.subscription-period, .subscription-frequency', function () {
76        var selectedValue = $('option:selected', this).val();
77        console.log(selectedValue);
78        var url = $(this).data('url');
79        var subscriptionField = $(this).data('field');
80
81        var urlParams = {
82            selectedValue: selectedValue,
83            subscriptionField: subscriptionField
84        };
85
86        url = util.appendParamsToUrl(url, urlParams);
87
88
89        $.ajax({
90            url: url,
91            type: 'get',
92            context: this,
93            dataType: 'json',
94            success: function (data) {
95                if (data.error) {
96                    console.error(data.errorMessage);
97                }
98            },
99            error: function (err) {
100               if (err.responseJSON.redirectUrl) {
101                   window.location.href = err.responseJSON.redirectUrl;
102               }
103           }
104       });
105   });
```

Figure 84. Modifications in cart.js (cont.)

4.6.1.16. default/checkout/shipping/singleshipping.isml

Add the following code after line 98, after the shipping method list:

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

This includes the subscription details template within a fieldset after the shipping method list.

```
96              <div id="shipping-method-list">
97                  <isinclude url="${URLUtils.https('COShipping-UpdateShippingMethodList')}"/>
98              </div>
99
100             <fieldset>
101                 <isinclude template="checkout/shipping/subscriptionDetails" />
102             </fieldset>
103         </isif>
104         <fieldset>
```

Figure 85. Modifications in singleshipping.isml

Add the following code on line 145, before the closing `</isdecorate>` tag:

```
<isif condition="${KlarnaOSM.isKlarnExpressCheckoutEnabled() &&
KlarnaOSM.getKlarnExpressCheckoutClientKey()}">
    <script src="https://x.klarnacdn.net/kp/lib/v1/api.js" async></script>
</isif>
```

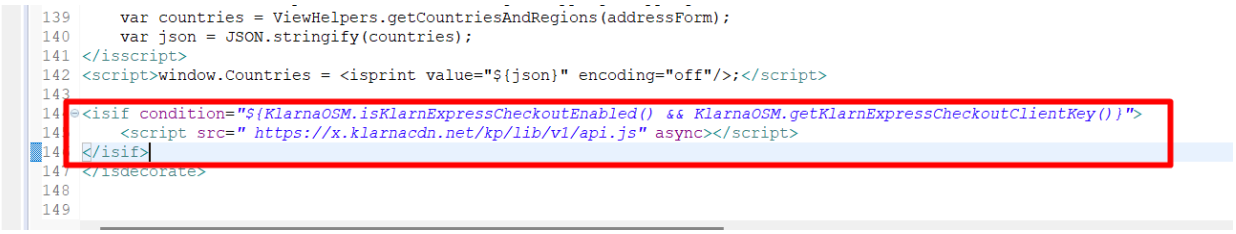


Figure 86. Modifications in singleshipping.isml (cont.)

4.6.1.17. scripts/cart/ValidateCartForCheckout.js

Add the following code block in the `validate` function before the `DONE` section on line 115:

```
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
    };
}
```

This code block performs validation on the cart products for subscriptions and handles any subscription-related errors.

```
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 87. Modifications in ValidateCartForCheckout.js

4.6.1.18. scripts/util/Resource.ds

Add the following code block before the validation messages section:

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

This includes resource messages for cancelling subscriptions and a global cancel action.

```
107     QUICK_VIEW_POPUP                : Resource.msg('product.quickview.popup', 'product', null),
108     TLS_WARNING                     : Resource.msg('global.browsertoolscheck.tls', 'locale', null),
109     CSRF_TOKEN_MISMATCH             : Resource.msg('global.csrf.failed.error', 'locale', null),
110     CANCEL_SUBSCRIPTION             : Resource.msg('heading.cancel.subscriptions', 'subscription', null),
111     CANCEL                          : Resource.msg('global.cancel', 'locale', null),
112
113     // Validation messages
114     VALIDATE_REQUIRED               : Resource.msg('validate.required', 'forms', null),
115     VALIDATE_REMOTE                 : Resource.msg('validate.remote', 'forms', null),
```

Figure 88. Modifications in Resource.de

4.6.1.19. js/pages/account.js

Add the following code block after the `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();
                            }
                        });
                    }
                }
            ]
        }
    });
});
}
```

This function initializes events for handling subscription cancellation.

```
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 89. Modifications in account.js

Add the following code in the `initializeEvents()` function to call the `initSubscriptionEvents()` function:

```
initSubscriptionEvents();
```

This ensures that the subscription events are initialized when the account page is loaded.

```
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 90. Modifications in account.js (cont.)

4.6.1.20. default/account/orders.isml

Add the following code within the `order-history-header` div:

```
<isif condition="${order.object.custom.kpCustomerToken}">
    <div class="order-number">
        <span class="label">${Resource.msg('label.subscriptions.subsciptionId',
'subscription', null)}:</span>
        <span class="value"><isprint
value="${order.object.custom.kpCustomerToken}" /></span>
    </div>
</isif>
```

This conditionally displays the subscription ID if the `kpCustomerToken` custom attribute is present on the order.

```
37
38              <div class="order-number">
39                  <span class="label">${Resource.msg('account.orders.numberlabel','account',null)}</span>
40                  <span class="value"><isprint value="${order.object.orderNo}" /></span>
41              </div>
42
43              <isif condition="${order.object.custom.kpCustomerToken}">
44                  <div class="order-number">
45                      <span class="label">${Resource.msg('label.subscriptions.subsciptionId', 'subscription', null)}:</span>
46                      <span class="value"><isprint value="${order.object.custom.kpCustomerToken}" /></span>
47                  </div>
48              </isif>
49
50          </div>
51
52          <div class="order-history-items">
```

Figure 91. Modifications in orders.isml

4.6.1.21. default/checkout/components/minicheckout_address.isml

Add the following code on line 16:

```
<div><isprint value="${Resource.msg('country.' +
pdict.p_address.countryCode.value.toLowerCase(), 'forms', null)}"/></div>
```

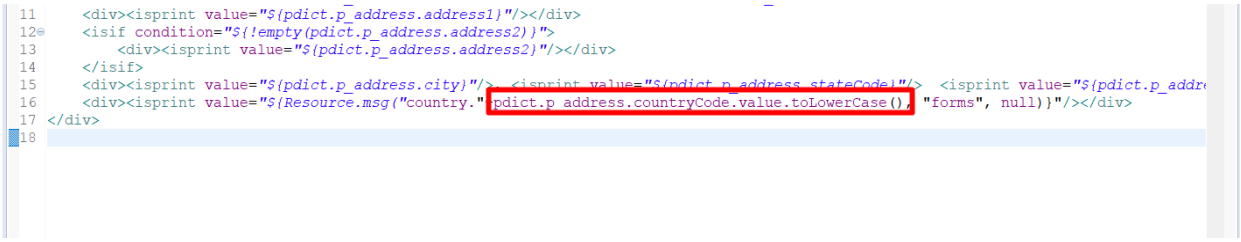


Figure 92. Modifications in minicheckout_address.isml

4.6.1.22. js/minicart.js

Add the following code on line 38 in the `mouseenter` event:

```
if (window.klarnaExpressCheckout) {
    window.klarnaExpressCheckout.klarnaExpressCheckoutMiniCart();
}
```

This code initializes Klarna Express Checkout in the mini cart when the mouse enters the mini cart area.

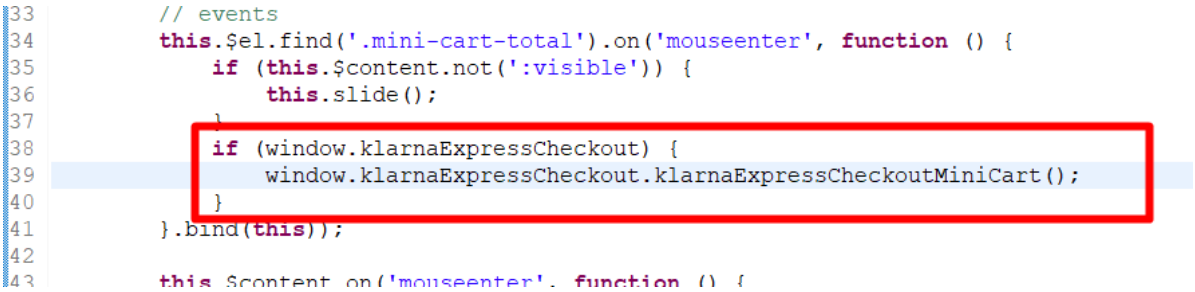


Figure 93. Modifications in minicart.js

Add the following code on line 60 in the `show minicart` event:

```
if (window.klarnaExpressCheckout) {
    window.klarnaExpressCheckout.klarnaExpressCheckoutMiniCart();
}
```

This code initializes Klarna Express Checkout in the mini cart when the mini cart is shown.

```
54          */
55          show: function (html) {
56              this.$el.html(html);
57              util.scrollBrowser(0);
58              this.init();
59              this.slide();
60              if (window.klarnaExpressCheckout) {
61                  window.klarnaExpressCheckout.klarnaExpressCheckoutMiniCart();
62              }
63              bonusProductsView.loadBonusOption();
64          },
65          /**
```

Figure 94. Modifications in minicart.js (cont.)

4.6.2. Controller modification

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

4.6.2.1. COBilling.js

In the `returnToForm()` method, add the following code block after the `pageMeta.update()` function:

```
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);
}
```

This code attempts to create or update a Klarna session and logs an error if it fails.

```
82    // if the payment method is set to gift certificate get the gift certificate code from the form
83    if (!empty(cart.getPaymentInstrument()) && cart.getPaymentInstrument().getPaymentMethod() === PaymentInstrument.METHO
84        app.getForm('billing').copyFrom({
85            giftCertCode: cart.getPaymentInstrument().getGiftCertificateCode()
86        });
87    }
88
89    pageMeta.update({
90        pageTitle: Resource.msg('billing.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')
91    });
92
93    try {
94        require('int_klarna_payments_controllers/cartridge/controllers/KlarnaPayments.js').CreateOrUpdateSession();
95    } catch (e) {
96        require('dw/system/Logger' ).getLogger( 'COBilling.js' ).error( 'Klarna Create Session Error: {0}', e);
97    }
98
99    if (params) {
100       app.getView(require('~/cartridge/scripts/object').extend(params, {
101           Basket: cart.object,
102           ContinueURL: URLUtils.https('COBilling-Billing')
103       })).render('checkout/billing/billing');
```

Figure 95. Modifications in COBilling.js

In the `resetPaymentForms()` method, add the command to remove Klarna payment instruments for the three conditions:

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

This ensures that any Klarna payment instruments are removed when resetting the payment forms.

```
354    var status = Transaction.wrap(function () {
355        if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals('PayPal')) {
356            app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();
357            app.getForm('billing').object.paymentMethods.bml.clearFormElement();
358
359            cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD_CREDIT_CARD));
360            cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD_BML));
361            cart.removePaymentInstruments(cart.getPaymentInstruments('Klarna'));
362        } else if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD_CRE
363            app.getForm('billing').object.paymentMethods.bml.clearFormElement();
364
365            cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD_BML));
366            cart.removePaymentInstruments(cart.getPaymentInstruments('PayPal'));
367            cart.removePaymentInstruments(cart.getPaymentInstruments('Klarna'));
368        } else if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD_BML
369            app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();
370
371            if (!app.getForm('billing').object.paymentMethods.bml.ssn.valid) {
372                return false;
373            }
374
375            cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD_CREDIT_CARD));
376            cart.removePaymentInstruments(cart.getPaymentInstruments('PayPal'));
377            cart.removePaymentInstruments(cart.getPaymentInstruments('Klarna'));
378        }
379        return true;
380    });
381
```

Figure 96. Modifications in COBilling.js (cont.)

In the `handlePaymentSelection(cart)` method, 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/KlarnaPayments.js')
.CancelAuthorization();
}
```

This cancels any Klarna authorizations if the selected payment method is not Klarna.

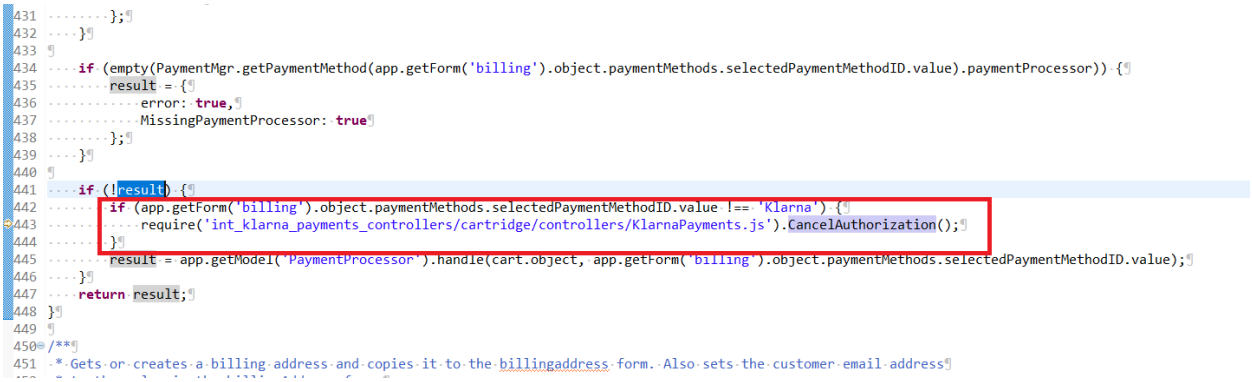


Figure 97. Modifications in COBilling.js (cont.)

4.6.2.2. COSummary.js

In the `submit()` method, add the following code block before the `showConfirmation(placeOrderResult.Order)` function:

```
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);
```

```
}
```

This code attempts to handle the Klarna redirect and logs an error if it fails.

```
65    var placeOrderResult = app.getController('COPlaceOrder').Start();
66    if (placeOrderResult.error) {
67        start({
68            PlaceOrderError: placeOrderResult.PlaceOrderError
69        });
70    } else if (placeOrderResult.order_created) {
71
72        try {
73            require('int_klarna_payments_controllers/cartridge/controllers/KlarnaPayments.js').Redirect();
74        } catch (e) {
75            require('dw/system/Logger' ).getLogger( 'COSummary.js' ).error( 'Klarna Redirect Error: {0}', e);
76        }
77
78        showConfirmation(placeOrderResult.Order);
79    }
80}
```

Figure 98. Modifications in COSummary.js

4.6.2.3. OrderModel.js

In the `placeOrder` method, add the following code block before the `var placeOrderStatus = OrderMgr.placeOrder(order);` line:

```
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;
}
```

This code handles orders with a pending Klarna fraud status by calling the `PendingOrder` method and logs an error if it fails.

```
/**
 * 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 99. Modifications in OrderModel.js

In the `placeOrder` method, add the following code at the end:

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

This code updates the customer subscription data if a customer token is present in the session.

```
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 100. Modifications in OrderModel.js (cont.)

4.6.2.4. CartModel.js

In the `addProductListItem` function, add the following code block after the shipment is assigned on line 153:

```
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;
    }
}
```

This code checks if the product is a subscription product and updates the product line item accordingly.

```
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 101. Modifications in CartModel.js

In the `addProductItem` function, add the following code block on line 220:

```
var isSubscriptionProduct = product.custom.kpIsSubscriptionProduct;
var isStandardProduct = !empty(product.custom.kpIsStandardProduct) ?
product.custom.kpIsStandardProduct : true;
if (isSubscriptionProduct && !isStandardProduct) {
```

```
    productLineItem.custom.kpSubscription = true;
}
```

This code checks if the product is a subscription product and updates the product line item accordingly.

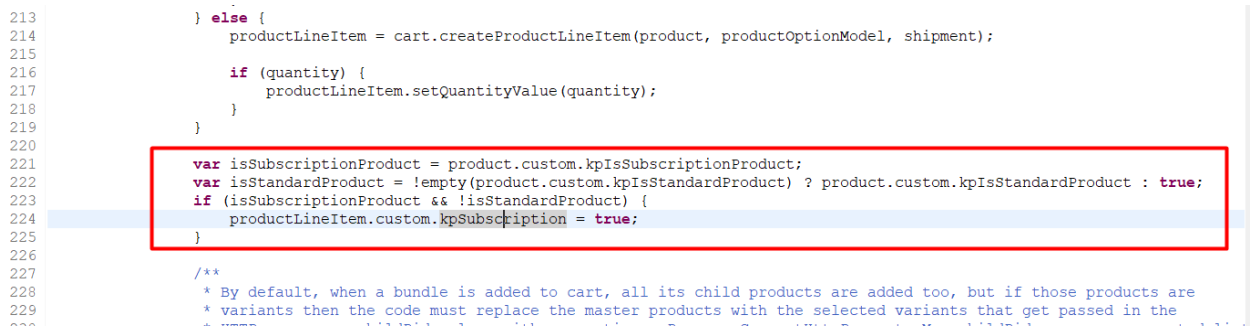


Figure 102. Modifications in Cartmodel.js (cont.)

4.6.2.5. Cart.js

Add the following code before the 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)
        });
    }
}
```

This function handles AJAX requests to update the subscription status of a product in the cart.

Add the following code before the 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,
        });
    }
}
```

This function handles AJAX requests to update subscription details in the cart.

Add the following code at the end of the 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);
```

This exposes the newly added methods to handle subscription updates.

```
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 103. Modifications in Cart.js

4.6.2.6. COShipping.js

Add the following code after the 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;
}
```

```
32   */
33  function prepareShipments(param) {
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 104. COShipping.js modifications (cont.)

4.6.2.7. Order.js

Add the following `subscriptions` function at the end of the 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');
}
```

This function retrieves the customer's subscriptions from their profile and renders the subscription history page.

```
101 function subscriptions() {
102     var subscriptions = [];
103     var profile = customer.profile;
104     if (profile.custom.kpSubscriptions) {
105         subscriptions = JSON.parse(profile.custom.kpSubscriptions);
106     }
107     app.getView({
108         subscriptions: subscriptions
109     }).render('klarnapayments/subscription/account/subscriptionHistory');
110 }
111
112 /*
113  * Module exports
114  */
115
116 /*
117  * Web exposed methods
118  */
119 /** Renders a page with the order history of the current logged in customer.
120  * @see module:controllers/Order~history */
121 exports.History = guard.ensure(['get', 'https', 'loggedIn'], history);
```

Figure 105. Order.js modifications

Add the following module export for the `subscriptions` function:

```
/** 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);
```

This ensures that the `subscriptions` function is accessible and secured.

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

Figure 106. Order.js modifications (cont.)

4.6.2.8. COCustomer.js

Add the following code on line 38, after the `removeAllPayments()` call:

```
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);
```

This code validates the subscription products in the cart and updates the cart subscription details. If there's an error, it redirects to the cart page.

```
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 107. Modifications in COCustomer.js

4.6.2.9. COPlaceOrder.js

Update the `start()` function with a parameter in attributes and pass it to the cart model:

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

This modification ensures that the `start()` function accepts a parameter and uses it to get the cart.

```
function start(param) {
    var cart = Cart.get(param);

    if (!cart) {
        app.getController('Cart').Show();
        return {};
    }

    var COShipping = app.getController('COShipping');

    // Clean shipments.
    COShipping.PrepareShipments();

    // Make sure there is a valid shipping address, accounting for gift certificates that do not have one.
    if (cart.getProductLineItems().size() > 0 && cart.getDefaultShipment().getShippingAddress() === null) {
        COShipping.Start();
        return {};
```

Figure 108. COPlaceOrder.js modifications

4.7. External interfaces

All requests to Klarna are performed through Klarna's REST API and are encrypted using SHA-256 with the shared secret provided by Klarna. To ensure security, only HTTPS is allowed for these communications. JSON is the format used for all requests and responses, facilitating consistent and structured data exchange.

For detailed information, including the resource structure for requests and responses, please refer to the full reference guide available in Klarna's developer portal: Klarna Payments API Documentation.

5. Testing

Klarna provides a set of testing credentials and triggers to facilitate the testing process. These resources ensure that integrations with Klarna Payments can be thoroughly tested before going live.

For detailed information and to obtain the necessary testing credentials, please refer to the following URL: Klarna Testing Environment.

6. Operations and Maintenance

6.1. Data Storage

6.1.1. System Object Extensions

6.1.1.1. 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 (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
Klarna Is Express Checkout	`kpIsExpressCheckout`	Determines if the basket is an express checkout
Klarna Session ID	`kpSessionId`	The Klarna session ID returned after “Create Session” API endpoint is called (Applicable since version 21.2.0)

Table 4. Basket Attributes

6.1.1.2. 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

Table 5. Order Attributes

6.1.1.3. 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 6. Order Payment Instrument Attributes

6.1.1.4. Payment Transaction

Parameter Name	Attribute ID	Description
Fraud Status	`kpFraudStatus`	Klarna Payments order fraud status
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 7. Payment Transaction Attributes

6.1.1.5. Site Preferences

Klarna Payments

This functionality is deprecated as of release 24.4.0

As of version 24.4.0, the **Klarna_Payments** preferences group is deprecated. The attributes have been reorganized as follows:

- All Klarna Payments related attributes have been moved to the new **Klarna_KP** preferences group.
- Attributes related to Klarna On-site Messaging have been moved to the **Klarna_OSM** custom preferences group.
- Klarna Express Checkout attributes have been moved to the **Klarna_KEC** preferences 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. 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 results 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 selected to NO, the default service profile will be utilized. The standard API rate limit for the Klarna service is as mentioned listed on docs.klarna.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.
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 mentioned below and public key shared in advance with Klarna
VCN Public Key ID	`kpVCNkeyId`	UUIDv4 value corresponding to the key pair. Shared with Klarna representative for Production & Playground (test) env configuration
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 >

		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 grayed 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 the Checkout Review page when a 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 an issue occurs. 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”
Klarna Express checkout (Klarna_ExpressCheckout)		
Klarna Express checkout Enabled	kpECEnabled	Boolean flag to enable/disable Klarna Express checkout
Klarna Express checkout Button Theme	kpECButtonTheme	The theme of the button. Options include default, light & dark
Klarna Express checkout Button Shape	kpECButtonShape	The shape of the button. Options include default, rect & pill
Placement	kec_placement	Multiselect attribute to choose where to display express checkout buttons. Values: cart, pdp, minicart

Table 8. Klarna Payments Site Preferences

Klarna Payments (Klarna_KP)

Parameter Name	Attribute ID	Description

Enable Klarna Payments	`kp_enable`	Enable/Disable Klarna Payments.Boolean attribute with Default value as true
Color Customization	`kpColorCustomization`	JSON attribute to customize color
Enable Extra Merchant Data	`kpEMD`	Enable this option to include customer_account_info and other_delivery_address as attachments when creating an order.Boolean with default value false
VCN - Enable settlement retry	`kpVCNRetry`	When set to “Yes”, SFCC will automatically retry the VCN settlement in the event of a service error. Boolean with default value false
Log Extra debug data	`kpLogExtraData`	Attribute to log debug data.Boolean with default value true
Merchant Reference 2	`merchant_reference2`	Enter the attribute from the SCC order (basket) object that you would like to forward as an additional merchant reference to Klarna. This attribute must be a class attribute of the SCC LineItemCtnr. String attribute
Agent username for subscriptions	`kpSubsUsername`	Agent username used to login on behalf of customers to create subscription orders.String attribute
Agent password for subscriptions	`kpSubsPassword`	Agent password to login on behalf of customers to create subscription orders.Password Attribute
Enable retry for subscriptions	`kpSubsRetryEnable`	Boolean with default value false

Number of Retries	`kpSubsRetryNumber`	Number attribute
Recurring retry frequency	`kpSubsRetryFrequency`	Number attribute

Klarna Express checkout (Klarna_KEC)

Parameter Name	Attribute ID	Description
Enable Express Checkout	`kec_enable`	Boolean flag to enable/disable Klarna Express checkout
Theme	`kec_theme`	The theme of the button. Options include default, light & dark
Button Shape	`kec_shape`	The shape of the button. Options include default, rect & pill
Placement	`kec_placement`	Multiselect attribute to choose where to display express checkout buttons. Values: cart, PDP, minicart

Table 9. Klarna Express checkout Site Preferences

Klarna On-site messaging

Parameter Name	Attribute	Description
Enable On-Site Messaging	`osm_enable`	Enable/Disable OSM functionality
Theme	`osm_theme`	Theme for on site messaging .Options include default,dark and custom
Placement	`osm_placement`	Multiselect dropdown which includes cart,pdp,header,footer and info. Placement Tag IDs are hardcoded in code.
Custom styling	`osm_custom_styling`	JSON to store custom styling

Table 11. Klarna On-site messaging Site Preferences

6.1.1.6. Product

Parameter Name	Attribute	Description
Is Klarna Standard Product	`kpIsStandardProduct`	Boolean attribute to define if the product is standard.
s 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.

Table 12. Product parameters for Klarna subscriptions

6.1.1.7. 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.

Table 13. ProductLineItem parameters

6.1.1.8. Profile

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

Table 14. Klarna Subscriptions profile

6.1.2. Custom Objects

6.1.2.1. Klarna Express Button

This functionality is deprecated as of release 24.4.0

Klarna Express Button has been deprecated and replaced with Klarna Express Checkout.

The Klarna Express Button (KEB) can be configured and adapted to fit your storefront's needs. Follow these steps to enable the KEB button:

1) **White-listing MID and Domains**:
    a) Your Klarna Merchant ID (MID) and domains must be white-listed. Contact your Klarna delivery manager for support with this process before testing and going live, as the white-listed domains may vary.
    b) Ensure that the MID matches the one configured in the Business Manager (BM) service credentials for your storefront.
2) **Configuring KEB Settings**:
    a) Navigate to "Merchant Tools – Custom Object Editor" and search for the `KlarnaCountries` custom object.
    b) Select the respective country key (e.g., "US") and provide the following details:
        i) **Express Button Merchant ID**: The allow-listed Merchant ID.
        ii) **Express Button Environment**: Choose the Klarna environment (Production/Playground).
        iii) **Express Button Cart Enabled**: Check this box to display the button on the cart page.
        iv) **MiniCart Express Button Enabled**: Check this box to display the button on the minicart page.
        v) **MiniCart Express Button Theme/Shape**: Set the theme/shape for the button on the minicart page.
        vi) **Express Button Library URL**: The URL for the button library.

vii) **Express Button Category**: The Klarna category to be pre-selected Refer to Figure 109 for a visual guide on completing the configuration.at checkout (e.g., “pay_over_time”).

3) **Shopper Experience**:
   a) The shopper lands on the cart page where the KEB is displayed.
   b) The shopper clicks on the KEB button.
   c) The shopper is prompted to share their Klarna registered credentials to complete authentication.
   d) The shopper is redirected to checkout with the Klarna Payment Method pre-selected.

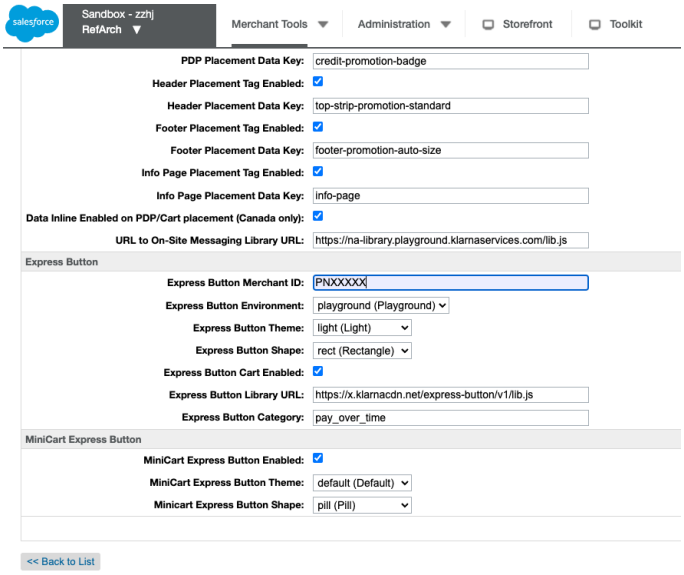


Figure 109. Klarna Express Button completed configuration

More customization options for the button can be found here.

Note: The Klarna Express Button is currently available in specific markets, with more to follow.

6.1.2.2. KlarnaCountries

This functionality is deprecated as of release 24.4.0

This object is deprecated from version 24.4.0, and replaced with Klarna Activation object.

The respective object is dynamically selected based on the requested locale country, e.g., SFCC site with locale `de_DE` or `en_DE` will use the `DE` custom object. In cases when the requested 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.

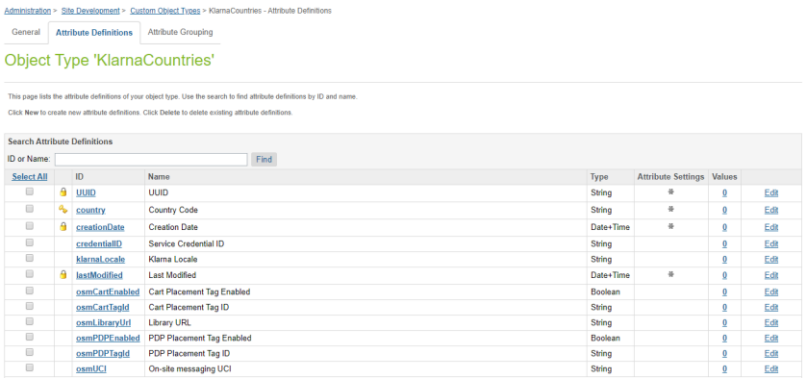


Figure 110. KlarnaCountries Attributes

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

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.
Cart Placement Theme	`osmCartTheme`	The theme for cart placement tag. It could be default or dark.
Cart Placement Custom Styling (#osm-cart-placement)	`osmCartCustomStyling`	Custom styling for cart placement. The css should be wrapped in <style>...customCss...</style> element.
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 the Product Display page for a given locale.
PDP Placement Theme	`osmPDPTheme`	The theme for PDP placement tag. It could be default or dark.
PDP Placement Custom Styling (#osm-pdp-placement)	`osmPDPCustomStyling`	Custom styling for pdp placement. The css should be wrapped in <style>...customCss...</style> element.
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
Header Placement Theme	`osmHeaderTheme`	The theme for header placement tag. It could be default or dark.
Header Placement Custom Styling (#osm-header-placement)	`osmHeaderCustomStyling`	Custom styling for header placement. The css should be wrapped in <style>...customCss...</style> element.
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
Footer Placement Theme	`osmFooterTheme`	The theme for footer placement tag. It could be default or dark.

Footer Placement Custom Styling (#osm-footer-placement)	`osmFooterCustomStyling`	Custom styling for footer placement. The css should be wrapped in <style>...customCss...</style> element.
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
Info Page Placement Theme	`osmInfoPageTheme`	The theme for info page placement tag. It could be default or dark.
Info Page Placement Custom Styling (#osm-info-page-placement)	`osmInfoPageCustomStyling`	Custom styling for info page placement. The css should be wrapped in <style>...customCss...</style> element.
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 “production”. E.g: Old library Test URL>

		https://na-library.playground.klarnaservices.com/lib.js , e.g: Live URL>https://na-library.production.klarnaservices.com/lib.js New library - https://js.klarna.com/web-sdk/v1/klarna.js
On-Site Messaging Environment	`osmEnvironment`	Environment on which the OSM is used. It could be playground or production.
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
MiniCart Express Button Enabled	`kebMCEnabled`	To enable the Klarna Express Button on minicart
MiniCart Express Button Theme	`kebMCTheme`	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”

Klarna Express checkout Client Key	`expressCheckoutClientKey`	The Klarna Express client key used for express checkout

Table15. KlarnaCountries Attributes

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.

6.1.2.3. Klarna Activation

KlarnaActivation custom object gives an opportunity to configure multiple MIDs per site. Merchants could have different credentials, clientIDs and VCN keys per country or group of countries. It has a similar structure to the Klarna Activation Site preferences group. Klarna Activation Key is the unique key for each Klarna Activation CO entry. It should be defined by the merchant and could have any unique string value.

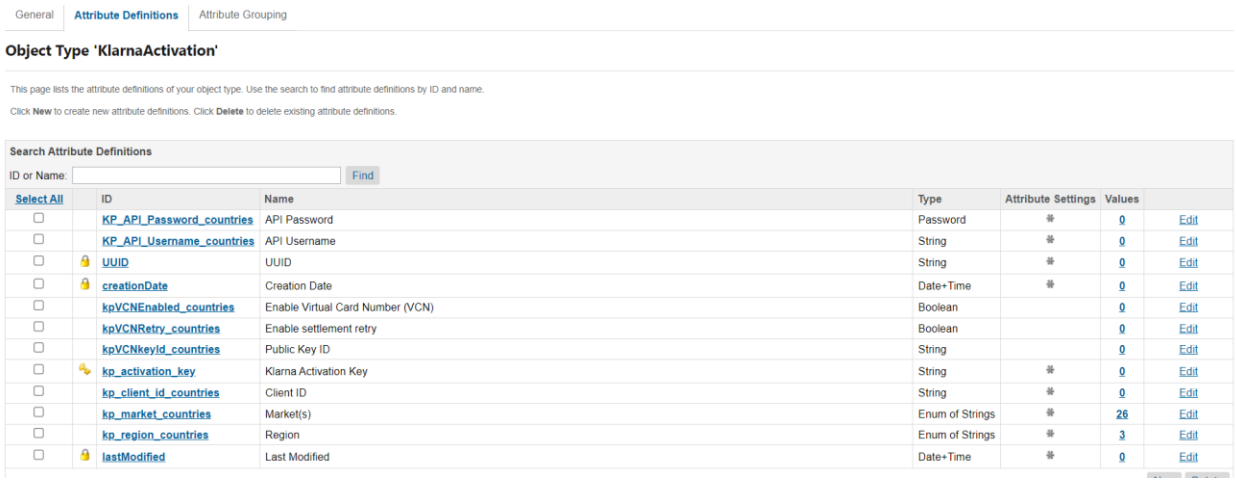


Figure 111. Klarna Activation attributes

Attribute Name	Attribute ID	Description
Klarna Activation Key	`kp_activation_key`	Unique identifier for a single KlarnaActivation entry. It is a free text entered by the merchant.

Attribute Name	Attribute ID	Description
Region	`kp_region_countries`	Klarna regions (Europe, North America, Oceania)
Market(s)	`kp_market_countries`	Klarna available countries - List of Klarna Countries
Client ID	`kp_client_id_countries`	Client ID generated by Klarna
API Username	`KP_API_Username_countries`	Username for Klarna services
API Password	`KP_API_Password_countries`	Password for Klarna services
Enable Virtual Card Number (VCN)	`kpVCNEnabled_countries`	Flag to enable virtual card number for selected countries. This will override the globally selected in Klarna Payments.
Public Key ID	`kpVCNkeyId_countries`	Countries specific public key for VCN
Enable settlement retry	`kpVCNRetry_countries`	Flag to enable settlement retry for selected countries. This will override the globally selected in Klarna Payments.

Table 16. Klarna Activation site preference

6.1.3. 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 since version 21.2.0)**	Client token returned by "Create Session" API endpoint and used to initialize the JS SDK
`KlarnaPaymentMethods`	The available payment method categories for the respective Klarna session; Saved in JSON format
`KlarnaPaymentsAuthorizationToken`	The authorization token returned by JS SDK "Authorize" call
`KPAuthInfo`	Whether finalization is required for the payment method; Returned by JS SDK "Authorize" call; Saved in JSON format
`KlarnaExpressCategory`	The KEB payment category; Currently applicable for US and defaults to "pay_over_time"
`KlarnaPaymentsRedirectURL`	The URL to redirect the customer to after placing the order; Returned by the "Create Order" API call
`'kpActive_' + countryCode`	Flag to indicate if Klarna is enabled for current site country
`'kpActivationSource_' + countryCode`	Klarna Activation source per country - Custom Object or Site Preferences. It is empty for old Klarna Countries config
`'kpActivationKey_' + countryCode`	Klarna Activation key in case of Custom object usage.

Table 17 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 18. Klarna Cookies

6.1.4. Library

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

- `footer-about`: An updated Out-of-the-Box (OOTB) asset that includes a link to the Klarna On-site messaging (OSM) dedicated page in the footer.
- `klarna-email-info`: An asset containing links to review Klarna Payment information, which is used in the confirmation email sent to customers.

6.1.5. Services

An HTTP service, `klarna.http.defaultendpoint`, has been added with the `klarna.http.service` profile.

Deprecation notice

- **Deprecation Notice for Version 24.4.0:** Service credentials and the KlarnaCountries custom object have been deprecated as of version 24.4.0. Please use the Klarna Activation Site Preferences or the Klarna Activation custom object to enter API credentials.
- **Replication Guidance for Version 21.2.0:** Prior to version 21.2.0, the KlarnaCountries custom object was replicable. To avoid issues with service credentials during replication, merchants should use the same service credential name across staging, development, and production environments.

For more details on updating the `KlarnaCountries` definition in your instances, please review Section Update KlarnaCountries Definition.

6.2. Logs

The integration includes the following types of logs:

- **Service Communication Logs**: These logs start with `service-klarna-***` and contain every request and response to the Klarna endpoints. Personal information, such as emails and names required for the Klarna API calls, is masked in these logs.
- **Custom Errors and Debug Info**: Depending on the case, custom errors and debug information are logged under `customerror-***`, `custodebug-***`, and `custominfo-***` files.

6.3. Availability

Cartridge functionality is dependent on the availability of the Klarna API service. The current operational status of Klarna can be viewed at Klarna Status.

6.4. Failover/Recovery Process

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

6.5. Support

A customer service workshop can be conducted during the implementation process before going live to align operational processes and ensure customer satisfaction. Klarna provides all customers with access to the Klarna App via the website Klarna App Login or by downloading the Klarna App (available for free on Android/iOS). Through the app, customers can contact support, view their statements, pay for purchases, track delivery updates, and extend due dates if they choose to pay after delivery.

6.5.1. Merchant Support

For reporting core SFCC functionality issues in the Klarna cartridge technical integration, please contact: commercecloud@klarna.com.

For production issues related to Klarna API availability, merchant representatives should reach out to their Klarna Account Manager after reviewing the current operational status at Klarna Status. If there are suspicions about degraded performance or issues with Klarna's service, report the problem in production (Post Go-live). The Klarna contact will then report this internally to the incident management team, which has established routines to handle and resolve reported incidents. The Klarna contact may request additional information from the individual reporting the problem to help the internal team ascertain and identify the issue. The KAM may also advise the merchant to follow updates on the status page if it is a known incident with ongoing updates.

Pre-requisite Information to Provide When Reporting an Incident:

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

This information helps speed up the investigation and resolution process.

7. User Guide

7.1. Cartridge Upgrade

Regular updates to our cartridge code include bug fixes, performance enhancements, security patches, and new features. Staying up to date with these changes is essential for maintaining compatibility, security, and optimal functionality of your integration.

7.1.1. Upgrade Process

Follow these steps to upgrade your custom code with the latest version of our cartridge:

1) Review Release Notes

Start by reviewing the release notes for the latest version of the cartridge. The release notes outline changes, improvements, and any potential compatibility considerations associated with the upgrade.

2) Assess Custom Code Changes

Identify any customizations or modifications you have made to the SFCC cartridge code in your custom files. These may include storefront customizations, controller adjustments, or custom business logic built on top of our cartridge.

3) Backup Custom Files

Before proceeding with the upgrade, ensure that you have a backup of your custom files, including any modifications made to the SFCC cartridge code. This backup will serve as a safety net in case any issues arise during the upgrade process.

4) Compare Code Differences

Use a version control system or a file comparison tool to compare the differences between your custom code and the latest version of the SFCC cartridge code. Pay close attention to areas where changes have been made to ensure compatibility and maintain functionality.

5) **Update Integration Code**

Integrate the latest version of the SFCC cartridge code into your custom files, replacing any outdated or deprecated code with the new implementations. Follow the migration guides and best practices provided to ensure a smooth transition.

6) **Test and Validate**

After updating your custom code, thoroughly test the integration to ensure that all functionality works as expected. Test various scenarios, including user interactions, data processing, and third-party integrations, to identify any potential issues or regressions.

7) **Address Compatibility Issues**

If you encounter any compatibility issues or conflicts with your existing custom code, troubleshoot and resolve them accordingly. Consult our support resources or reach out to our team for assistance in addressing compatibility concerns.

8) **Deploy Changes**

Once you are satisfied with the upgrade and have validated its functionality, deploy the changes to your production environment. Monitor the integration closely following deployment to ensure ongoing stability and performance.

7.2. Roles and Responsibilities

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

7.3. Storefront Functionality

When Klarna is set up, Klarna Payment options and iframe widgets will be shown on the billing step. All SFCC out-of-the-box (OOTB) checkout functionality remains in place, such as:

- 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 a bonus product

To use Klarna's payment options on the billing step of the checkout process:

1) Select one of Klarna's payment options as the payment method.
2) Click the “Next: Place Order” button:

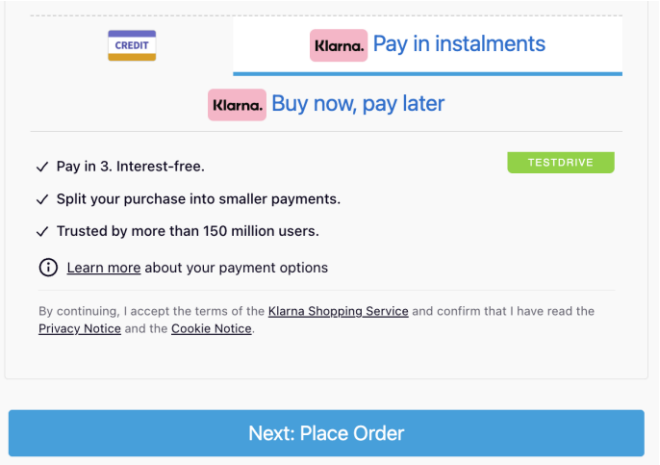


Figure 112. Payment Options on Checkout

3) Depending on the payment method selected and the region, a Klarna popup window will appear. Follow the steps on the screen:

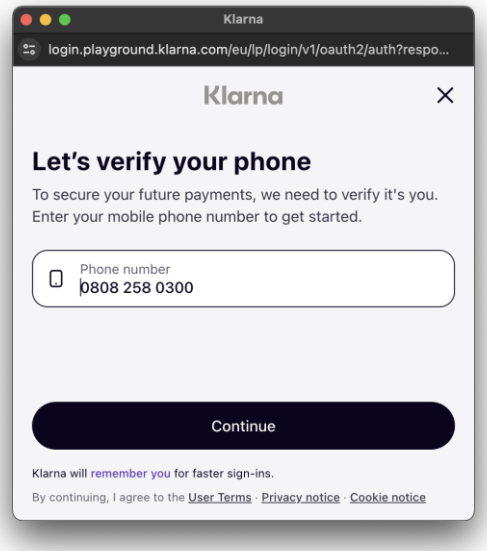


Figure 113. Klarna Pop Up Screen

4) On the Review step click on “Place Order” button:

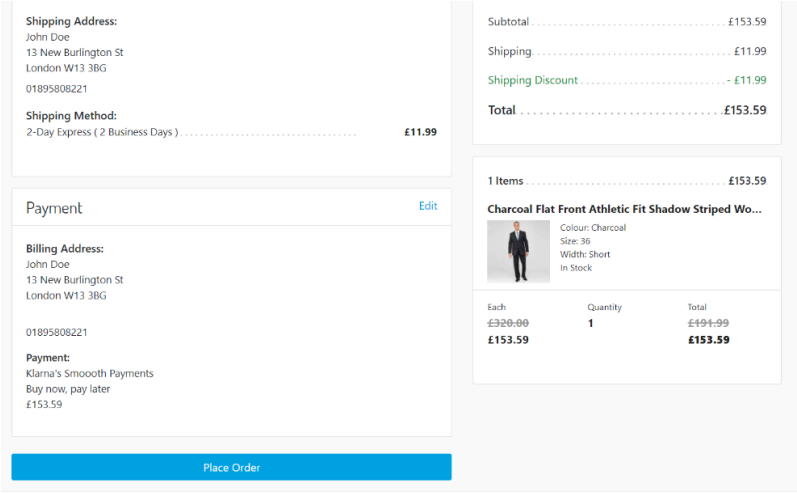


Figure 114. Payment Review Screen

5) The customer’s browser is sent to the `redirect_url` and immediately thereafter shown the Commerce Cloud Order Confirmation page:

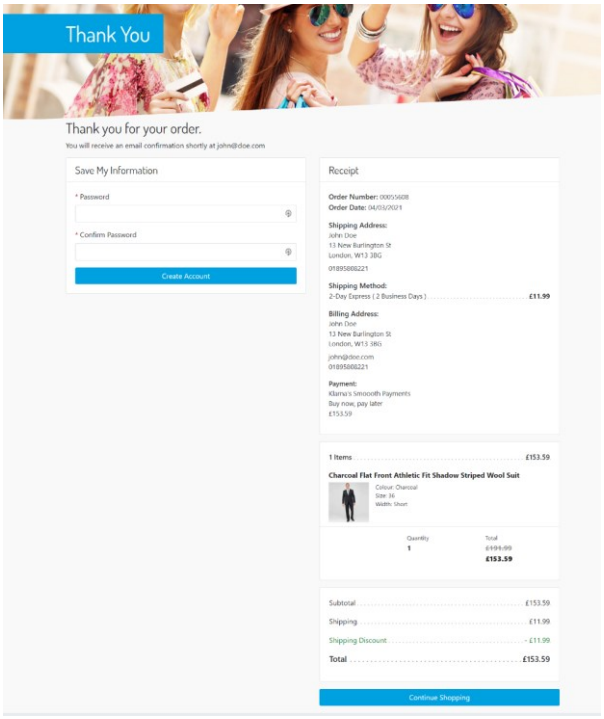


Figure 115. Order Confirmation Page

6) The newly created order can be inspected in Business Manager:

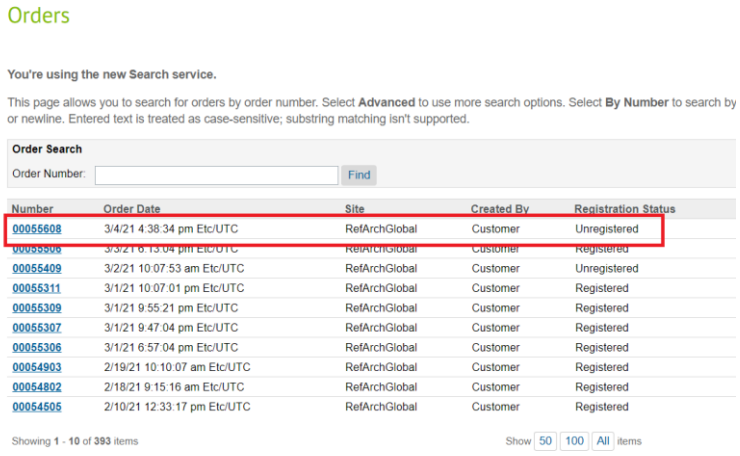


Figure 116. Orders List in BM

a) Klarna Payments order ID can be inspected in the Attributes tab of the order:

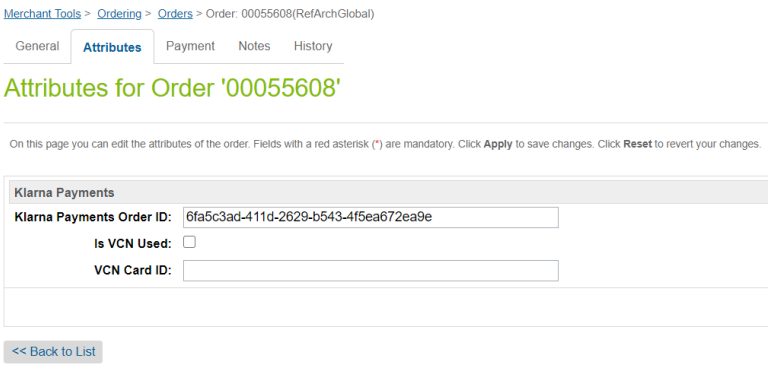


Figure 117. Order Attributes

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

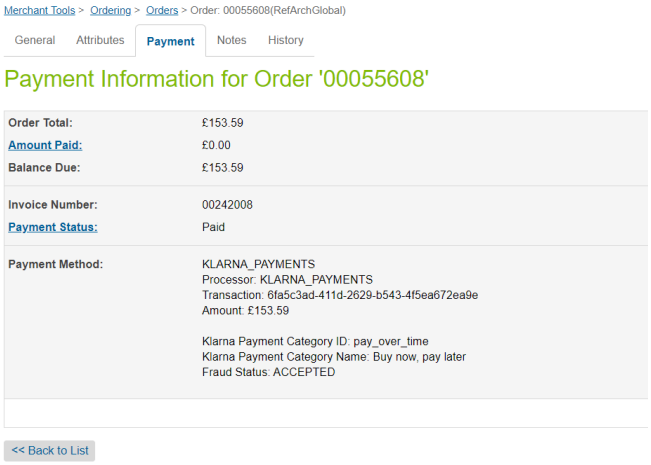


Figure 118. Order Payment Detail

c) The order can be further inspected in the Klarna Merchant Portal:
   i) EU: eu.portal.klarna.com
   ii) US: us.portal.klarna.com
   iii) OC: us.portal.klarna.com

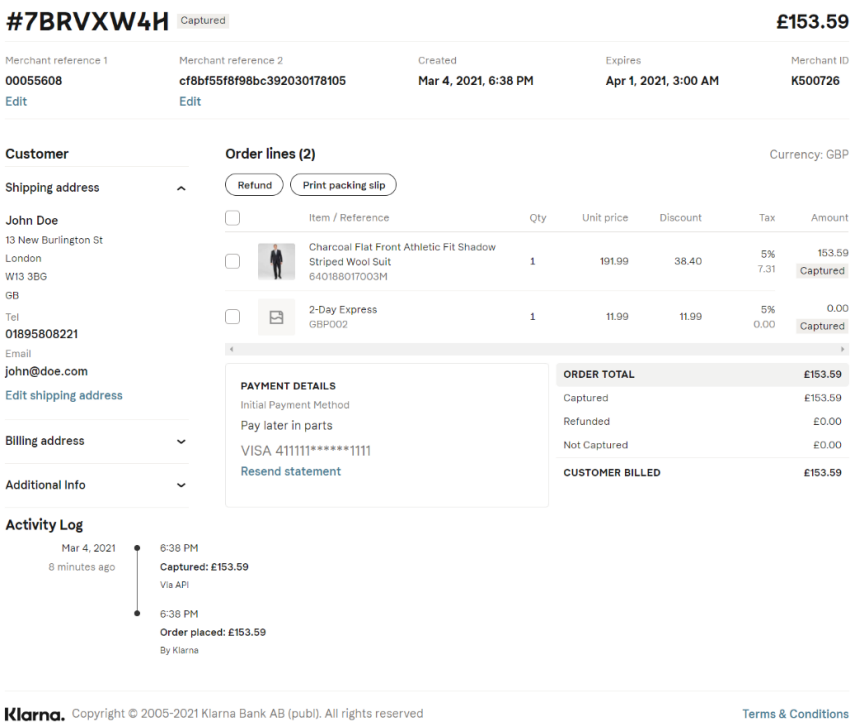


Figure 119. Klarna Portal Order View

8. Release History

Version	Date	Changes
18.1.0		Initial release of Klarna Payments SFRA.
19.1.0	June 2019	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.

Version	Date	Changes
19.1.6		New country locales added. Updated VCN to store encrypted card details
21.1.0	March 2021	Fixes around discounts taxation & VCN error handling. Added VCN improvements, additional OSM placements, BOPIS support. New IT, CA, FR & NZ country locales. Removed acknowledge call. Documentation updates.
21.1.1	March 2021	New On-Site Messaging attribute for Canada. Remove not required locale templates for SG Spain & Belgium. Documentation updates.
21.1.2	April 2021	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	June 2021	Added Klarna Express Button. Moved Klarna session ID & client token from SFCC session privacy to Basket attributes.

Version	Date	Changes
		Changed KlarnaCountries definition to not replicable. Code cleanup. Documentation updates.
21.3.0	November 2021	Improvements for create_session errors Expired user session issues related to empty shipment.shippingMethod Additional locale (PL) included in config files
21.3.1	November 2021	Documentation updates.
22.1.0	February 2022	Improvements for create and update session errors Added Klarna Express Button in minicart. Support for long running basket Rate-limits by operations
22.2.0	March 2022	One Klarna Optimisation Mexico locale support
22.2.1	April 2022	Rollback of One Klarna Optimisation

Version	Date	Changes
22.3.0	May 2022	SFRA ver. 6.0.0 support Rollback hide VAT from Checkout functionality
22.3.1	May 2022	Fix User-agent version sent to Klarna services
22.4.0	July 2022	Intent field addition in Klarna Payment session creation Combine Klarna Authorization and Create Order in Checkout Review Step
22.5.0	January 2023	OMS support
23.1.0	July 2023	Fix issue with incorrect values for EMD Improvement Klarna Auto Capture and error handling Logging information for troubleshooting bugs Add Auto_finalise=True to the review checkout flow
23.1.1	September 2023	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.

Version	Date	Changes
23.2.0	December 2023	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.
24.1.0	January 2024	Klarna Express checkout: a new feature introduced in Storefront where Express Checkout Button will be displayed in PDP, Cart and Mini Cart and the user will be redirected to Klarna after clicking the Express Checkout button. Klarna Express checkout enabled quick and easy checkout where Shipping Address, Billing Address and Payment details will be preselected so that Checkout can be completed in fewer clicks. Fix for creating order service calls with 500 status responses: now orders are not created and error is thrown for these cases.
24.2.0	March 2024	Update of OSM functionality that is more aligned with current web standards, provides a consistent identifier system as used in KEC, and offers enhanced customization options to our merchants. This will not only improve the user experience but also reinforce Klarna's commitment to providing versatile and state-of-the-art e-commerce solutions. CSS customizations are available only in the new library version. Documentation update - new section added for cartridge upgrade process.
24.3.0	May 2024	Update of Klarna Express checkout configuration. The display of Klarna Express checkout buttons is now customizable by selecting preferred placements. By default, none is selected.

Version	Date	Changes
		Documentation update – new section added "Klarna Display Conditions and Authorization Handling".
24.4.0	June 2024	**New features** ● **RO and CZ** countries support **Configuration Structure Updates** ● **New Settings Points** ○ Klarna Activation Custom Object ○ Site Preferences: ■ Klarna Activation (Klarna_Activation) ■ Klarna Payments (Klarna_KP) ■ Klarna Express Checkout (Klarna_KEC) ■ Klarna On-site Messaging (Klarna_OSM) **Deprecations** ● Klarna Countries Custom Object ● Site Preferences: ○ Klarna Payments (Klarna_Payments) ○ Klarna Recurring Payments (Klarna_RecurringPayments) ○ Klarna Express Checkout (Klarna_ExpressCheckout) ● Removed Site Preferences Attributes: ○ kpServiceName ○ kpBankTransferCallback ○ kpRejectedMethodDisplay

Version	Date	Changes
		○ kpNotAvailableMessage ○ vcnPrivateKey ○ vcnPublicKey ○ KpRateLimitByOperation ○ kpCreateNewSessionWhenExpires ○ sendProductAndImageURLs **Note:** Site preferences and service credentials for the deprecated items are now obsolete and must be migrated to the new site preferences or activation custom object.

8.1. Known issues

There are no known issues in version 24.4.0.

9. Additional Information

9.1. Klarna API Information

The Klarna Payments API is accessible through different endpoints 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

To access the Klarna API in the production environment, use the following URLs based on your region:

- **Europe**: https://api.klarna.com/
- **North America**: https://api-na.klarna.com/
- **Oceania**: https://api-oc.klarna.com/

9.1.2. Testing Environment

To access the Klarna API in the testing environment, use the following URLs based on your region:

- **Europe**: https://api.playground.klarna.com/
- **North America**: https://api-na.playground.klarna.com/
- **Oceania**: https://api-oc.playground.klarna.com/

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

To generate an RSA key pair with a 4096-bit private key, follow these steps:

Generate RSA Key Pair: Use the following OpenSSL command to generate a 4096-bit private key:

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:4096
```

Extract Public Key: Use the following OpenSSL command to extract the public key from the RSA key pair:

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

File Output: These commands will create two files in the folder where you executed the commands:

- `public_key.pem`
- `private_key.pem`

File Contents: The contents of the files should look something like this:

Public_key.pem:

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9wOBAQEFAAOCAg8AMIICCgKCAgEAoNYG7l2G8nZa+22oBYZk
tV228lw3UE9WO4oxfknJtKEdHn84x55ULt8KQTh9NVtdeKC8nTfTgyvMt/GNCa18
xuZV/lGYDftKt85hbV5EjOum+StAIufEXvlBX7nMOMc1KyWm9kp2kbqd88mFIX63
KV94OoNEXcNatRDFYR+qz53+ifadDQtQ1slVNStdroCZDJ1+LxtBy9V+BdmsBK1E
RLsKh/JLXyWE24FJKV+zOOs7TQkdWW/5ET12OGQYZsWo1yqgi9HplNvrisve8vWP
xaL4m8iZ3I/9yYdg7yANQbTxSJcbbRCgaaagPo3OCNxeqU6qafY5g8vY3E52CoXH
DdO4UslX1qcuYIDhqaDzey6W+b8m755xLi+rqQyM4PBWLOJOdM3FVid8+4YKILex
3AKBFciqRCMHSOGaEeyrXKTjlAsghr9RS8PifvQRrL44OcHzqw2vXODvpjSWcmUJ
tW4wUq5RNSsobrxnVmoV6fj1z67Q/1P+l5Ie+oowdahR5ztVqJlO+2PNoX4I5VDs
/Pkz3f8wWVc3Mp2oNT244o+/NIiyRfPFaJJx7JAgrcvZt2nFAmY4QApXLFJCpgEM
wYucE4AH4gJKsh3KZbxRERrrO72bL2rxvWqBp/Oh7DcMsV9sQs4BvxxIl6CF5O6F
```

```
ThzmclaKLBAyd5LALiXiPfkCAwEAAQ==
-----END PUBLIC KEY-----
```

Private_key.pem:

```
-----BEGIN PRIVATE KEY-----
MIIJQQIBADANBgkqhkiG9wOBAQEFAASCCSswggknAgEAAoICAQCg1gbuXYbydlr7
bagFhmS1XbbyXDdQT1Y7ijF+ScmOoROefzjHnlQu3wpBOHO1W114oLydN9ODK8y3
8YOJrXzG5lX+UZgN+Oq3zmFtXkSM66b5KOAi58Re+UFfucw4xzUrJab2SnaRup3z
yYUhfrcpX3g6gORdw1q1EMVhH6rPnf6J9pONC1DWyVU1K12ugJkMnX4vGOHL1X4F
2awErUREuwqH8ktfJYTbgUkpX7PTSztNCR1Zb/kRPXY4ZBhmxajXKqCLOemU2+uK
y97y9Y/FovibyJncj/3Jh2DvIA1BtPFIlxttEKBppqA+jfQI3F6pTqpp9jmDy9jc
TnYKhccNO7hSyVfWpy5ggOGpoPN7Lpb5vybvnnEuL6upDIzg8FYvQnROzcVWJ3z7
hgogt7HcAoEVyKpEIwdI4ZoR7KtcpOOUCyCGv1FLw+J+9BGsvjjRwfOrDa9fQO+m
NJZyZQm1bjBSrlE1KyhuvGdWahXp+PXPrtD/U/6Xkh76ijB1qFHnO1WomU77Y82h
fgjlUOz8+TPd/zBZVzcynag1Pbjij78OiLJF88VoknHskCCty9m3acUCZjhAClcs
UkKmAQzBi5wTgAfiAkqyHcplvFERGus7vZsvavG9aoGn/SHsNwyxX2xCzgG/HEiX
oIXnToVOHOZyVoosEDJ3ksAuJeI9+QIDAQABAoICACRkaUsUNI22RB3yEPu3DiCP
pO6v+QAeA4gTW+GUdqR9dCZLaSCZ7bhxVVOuoX4qPzslO6hjUmOyzG6upFgVPk+P
HNQfyEUZoC148Eib9OziAXUN2URMpv1KbwVm+BO814X8zguai7uruOPHTG1oy677
4Ct1OknxAxxHQDIaxT6XJFo5SA4EinUfNz2Bo3/xry/QjxW/mCKOGwDd4PNp9TGM
FPTv2SgdSDOWzGQlOH5N3owuzMpI8NV6z74wv+i5Ptv41Dzu8WhyXpiYSskOOSRK
HPC68j2bAzTPghp5aSZ9976SGm2SPonJXyboXdiHbI/osdyqDxeIT3iB9GmrHX/i
kHPGJCh7fRZvqj39Hc+IxYjabwW3rDeDIPB7ab9z1KLF4z1D6AZOKCPyTaDRdQ1Q
eDi7LwDmk7NHEPrmF/nIcguQdqbIbmFO2zEsOTOe6y4uBMndRsbQprTNSMUdBkrA
lNaYVSTQ1ZOY/8DZDpGcyS1OnJv74F15uDjKN6/ov991mZ1JrZ+V2sdS3EDUlmvP
6thQKwI7Ln6h+ApHtWUG1NmvQe5gJEOqAeJ9b45clUzIRUwhVmEp8NoIJhOkAjaN
d4lk7xy9ZRDUY5yekPeYrJPShjsHAyEoktJIjRufI2UUq3uxNjjICoQcOVGfNDIS
YTTPwpu1pmCOC+rh2fgBAoIBAQDRultRArvtc2JKhVOUyZk88zd9kvrI6fNiyKmi
HgiWf7qkTPD9xhOQWDw3iwRFQAD+YkgV5MCBO8wp8oO8GEsOCI+XZWExOcPTOVfj
PZHiQrTFnlfG/+fAO14xLf3j3ED4YQXdHOKI3xoLknQx/EydLoctxgkkpgWLrsA7
DwdSAg1/OsBvaHY27ogAfdimHdaKZ5OAe4a9k1qP3xVZBuOe8Sd65unBavUJLDuv
ikeNmkSVgW1sm55/729JIr63USHF76It+vE1cdZ+vKg5vYotsQgPzvNBmUO/E8Gj
zMXQRfqfvEDlNXEXOrCupTkw1G6AGTwQc/NPzyr/LTpLe6UBAoIBAQDEUjTiG11V
hf7WjdG3gctRlr+mYapQHgXdVLx2QSaqUYid+OQXK11YfJlsRB6nwa+OED83RfPO
lIFqxpzudSLPmoDuIBT7Dl5c/aleyKs/siUusP8QVDXk6OAR84XSytC35sIRV7pE
VMuBL91jfkQOLf/PreslK/kI6Yvwwp4qrHK6/f9TgciHclYtf+/oti4ky6GJgfmP
fmuCqjxmUKbXXFPd5RbL2THGOowilb8zDLjf3RlbjlQFqogAk6H9hp2VOVZLiJHp
UWM3z3zxDWeDaqJO8sHuk/rA9QpsVTu8IGTQsxdj8JwluN1Q+YZiOuPiSENBqPzT
V3exexzo3sD5AoIBAGU3qEyPojz1+9D1SaI8LW2CABzlq4z9g84ABAZOslxX5q7W
x1PinZyDSQSRXg1B13jt29ZdIR79ygnQlg1YOBjcvtgVQHPuafk3RlBQbbCh+vaI
9dn/tUxMGqhnhunKaby1rovJHfdqnPpKwzNAjYUqaGkJ822xhmmke/fEyAanIPa4
stDRvIPEWPTLx5xcOCdx13khpKSnkgRvaLEfpwkVX7Vr7hK/2OSFaYTNmrzXYBQ7
c6D/9d3Oo4nLb/mu+Tq67S19t53Qg/GEgTfkpuRoVPiOKyhUnKKCGWlBMZLTwyIG
S9eTFDKoJOcSTGipjW7bPua93wZ8eEbRABpf4QECggEANNhQBeEJOaCdBVHtdrEI
```

```
crDaa8XOW1aJi5dol4hYCRajaKsfHAF/QfdgMQVxHwUC5YG4En/Q+DAVWhGWYpXD
RhC3zeFy5FVszykOsx/fAOlKGvRn5BRW4YRR9GMRzbjsT+RcruBnckdE9ERXGpX9
c/JB3rxZBIt+oIiFM8yfWKtMwsrmNKtFuDftvJeok4KejycFF4eWDqsf828xjPT+
xA/FP4CQD1UqkcpmuFSIwAwXo6LXVY7NTSOnKMiUnTLkLlTIHtLnO9+9jmNapWRP
Tc+hZUuHKlpI8DHFmX2j87LgkFDO5eD5lynY4RgZtU1W1C1RdVYwoA72WB7knEaB
uQKCAQAH9s67P/7fFX9dfEans3PHU4nGjD8dJ8eoNQ6DhBMydZpGWI5ZUeEBZDRk
OcBOeRs5BOcS43Em9kETpzawyCwxmnwzl+CzoPzMQcTw9tXomF9HG6RJ9XBdJfGA
ALAwCd4bASxmFM6guSP5GKnZ9aY3tR3tWWDfr7f9z8wOewzzpPclwRhOO9fPe4TC
NXoEm1MELJVeUieDSLKZgjgCw8WHGqLItONpAO/fwSM2gIcxETVV7qx3aPuJzCVh
LQZoBLQk3UMKsWDdpzeBdiERe66NAgVk92Xe7SY9EY2vymaq761i1x1vlprT27qp
24OLDJawqMOIraKmdCvWjofWSaOU
-----END PRIVATE KEY-----
```

Key Pair Association: This key pair must be associated with a `key_id` (UUIDv4). The public key must be shared in JWK format with your Klarna contact. Note that for both production and playground environments, different `key_id` and key pair combinations must be configured before testing or going live with the virtual card product.

9.3. Decrypt VCN Card Details

To decrypt the virtual card details stored at the order level and authorize the credit card processor, you can use the following code snippet. This script utilizes the Salesforce Commerce Cloud (SFCC) APIs and classes for cryptographic operations.

```
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];
```

Steps:

1) **Retrieve the Order**: Use `OrderMgr.getOrder("order_id")` to fetch the order by its ID.
2) **Store the VCN private key in SFCC:** Create custom attribute to store the private key:

    a) **Site Preference for single site credentials**
        i) Go to **Administration** > **System Object Types**.
        ii) Select **Site Preferences** from the list of system object types.
        iii) Click on **Attribute Definitions**.
        iv) Click on **New** to create a new custom attribute.
        v) Define the Attribute:
            (1) **ID**: Enter ***'vcnPrivateKey'*** (this is the unique identifier for the attribute).
            (2) **Display Name**: Enter '***VCN Private Key***'.
            (3) **Description**: Enter '***SSL private key used only to decode Virtual Card information (used with kpVCNEnabled)'***.
            (4) **Type**: Select ***'Text'*** from the dropdown.
            (5) **Mandatory**: Ensure the mandatory flag is set to '***false'***.
        vi) Save the attribute by click on **Apply**
        vii) Click on **Attribute Grouping**
        viii) Select **Klarna_KP** group
        ix) Add the new attribute ***'vcnPrivateKey'*** and save
        x) Open **Merchant Tools > Site Preferences > Custom Site Preference Groups > Klarna Payments** and update the **vcnPrivateKey** value

b) **Custom Object attribute for multiple Klarna credentials per site**
   i) in Business Manager, go to **Administration** > **Site Development** > **Custom Object Types**.
   ii) Locate the custom object type named **KlarnaActivation** and click to edit
   iii) Navigate to the **Attributes** tab.
   iv) Click on **New** to create a new attribute.
   v) Fill in the following details:
      (1) **Attribute ID**: *vcnPrivateKey*
      (2) **Display Name**: *VCN Private Key*
      (3) **Description**: *SSL private key used only to decode Virtual Card information (used with kpVCNEnabled).*
      (4) **Value Type**: Select ***Text***
      (5) **Mandatory**: Ensure this is ***unchecked***, as *mandatory-flag* is *false*.
   vi) Still within the Custom Object **KlarnaActivation** , navigate to **Attribute Groups** tab.
   vii) Locate and click on Edit of **'vcn'** attributes group
   viii) Find the attribute ***vcnPrivateKey*** and add it to the group
   ix) Go to **Merchant Tools > Custom Objects > Manage Custom Objects**
   x) Select **KlarnaActivation** ObjectType and search
   xi) Enter the ***vcnPrivateKey*** for each entry

3) **Get the VCN Private Key**: Retrieve the VCN private key from the custom site preferences using:
   a) From Site Preferences (single credentials) `Site.getCurrent().getCustomPreferenceValue('vcnPrivateKey')`.
   b) From Custom Object - please take a look at `KlarnaHelper.getVCNKeyId()` and create similar function to get the private key using its id ***vcnPrivateKey*** **.**
4) **Decrypt the AES Key**:
   a) Convert the encrypted AES key from Base64 to bytes using `Encoding.fromBase64`.
   b) Decrypt the AES key using `cipher.decryptBytes` with the RSA algorithm.
5) **Decrypt the Card Data**:
   a) Convert the encrypted card data from Base64 to bytes using `Encoding.fromBase64`.
   b) Decrypt the card data using `cipher.decryptBytes` with the AES algorithm.
6) **Parse and Extract Card Details**:

a) Decode the decrypted card data from URL encoding to UTF-8 format.
b) Parse the JSON string to get the card details object.
c) Extract the card PAN, CVV, and expiry date.

7) **Authorize the Credit Card Processor**:
   a) Use the decrypted card details to proceed with the credit card authorization process.

Important!

- Ensure that sensitive data such as card details are handled securely and comply with PCI DSS standards.
- Remove or secure any logging of sensitive information in a production environment.
- For more information about the decryption process, refer to the Klarna documentation.

9.4. Update KlarnaCountries Definition

This functionality is deprecated as of release 24.4.0

This object is deprecated from version 24.4.0, and replaced with the Klarna Activation object.

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:

1) Back-up `KlarnaCountries` configurations per country for each site as changing the definition will remove all configurations on your environment.

a) **For merchants with one site** - Go to "**Merchant Tools > Custom Objects > Import & Export**", select `KlarnaCountries` & export the data.

b) **For merchants 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 the Klarna ones.

2) Export custom object definitions from "**Admin > Site Development > Import & Export**". This action will export all custom object definitions that you have.

3) 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>`"

4) Import the updated `KlarnaCountries.xml` definition in "**Admin > Site Development > Import & Export**".

5) 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`