CCV PWA Link Cartridge Documentation

CCV Link Cartridge

Table of Contents

Functional Overview

CCV is a payment processor providing a wide range of payment methods.

This cartridge provides an integration of the CCV payment processor with the headless Salesforce PWA kit and Salesforce Commerce Cloud platform.

For most payment methods in this cartridge use a hosted payment page approach – upon placing the order the customer is redirected to a webpage, hosted by CCV, where they can complete their payment. They are then redirected back to the merchant’s site, where the payment transaction’s status is checked and handled appropriately.

For Apple Pay, a payment sheet pops up on the checkout page, so they can complete their payment without leaving the merchant’s site.

The payment methods supported by this link cartridge are:

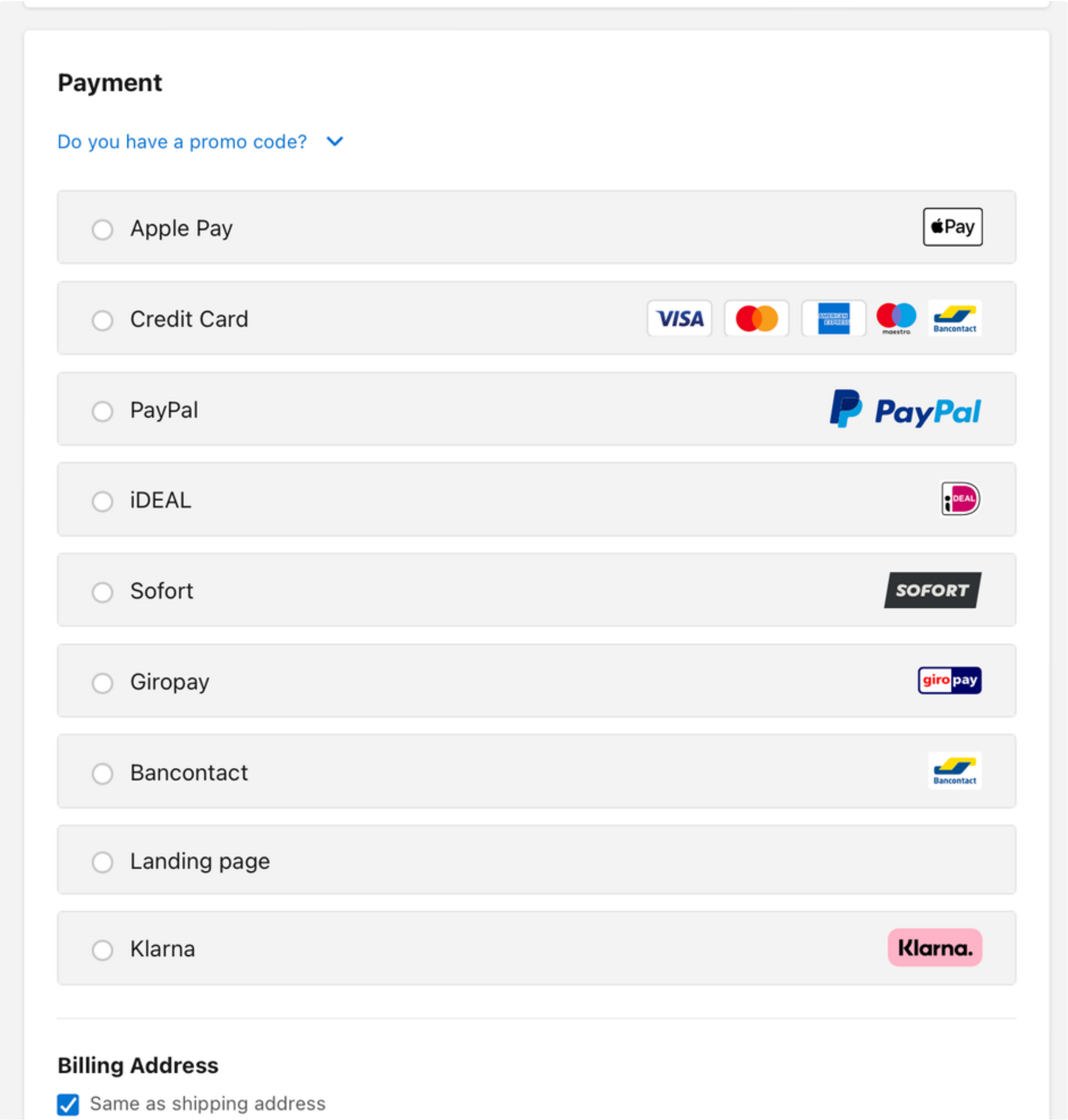
- Credit cards – VISA, Mastercard, Maestro, Amex, Bancontact
- PayPal
- GiroPay
- iDEAL
- Sofort
- Klarna
- Landing page
- Apple Pay (using hybrid approach)

Manual cancellation, reversals and refunds are supported via a Customer Service Center action.

Credit cards “remember card” functionality is supported via the CCV vault token API.

Use Cases

- As a customer, complete a checkout processes using the CCV payment methods with the ability to remember your card data when logged in.
- As the merchant, view the CCV payment details in the Business Manager
- As the merchant, perform custom actions regarding the payment (cancel payment, refund payment, ship order)
- As the merchant, configure multiple settings, controlling the behavior of the CCV payment logic, in the custom site preferences



Payment

Do you have a promo code?

○ Apple Pay

○ Credit Card

○ PayPal

○ iDEAL

○ Sofort

○ Giropay

○ Bancontact

○ Landing page

○ Klarna

Billing Address

☑ Same as shipping address

Limitations & Contraints

The plugin was built to work with the SFCC PWA kit app, using data from the demo RefArch site. The following CCV API's are used throughout the code:

- /payment
- /method
- /cancel
- /refund
- /reversal
- Endpoints submitted via cardDataUrl / cancelUrl (e.g. /applepay/*/encryptedtoken)

Updates to SFCC orders are implemented via webhooks.

Installing this link cartridge requires a developer with some experience of SFCC and the react PWA kit app.

Compatibility

Node .v 14.17.6

PWA kit app – v.2.6.0

SFRA v.6.3.0

SFCC Compatibility mode: 22.7

The PWA components created for this implementation require that only payment methods using the CCV payment processor are enabled - they are not compatible with payment methods using a different payment processor.

Privacy

Sensitive credit card data is never processed on the Commerce Cloud side. Payment data is entered and processed on the CCV Payment page.

For the “remember card” functionality a token is stored on Commerce Cloud side, but the sensitive payment data related to the token is only stored in the CCV side.

Installation

BM setup

1. Create a new code version and upload the cartridges **int_ccv** and **bm_ccv** to BM.
2. Run **npm install** in the root of the **link_ccv** folder.
3. Add the `int_ccv` cartridge to the **site cartridges path**:

BM > Administration > select your site > Settings tab > Cartridges

Add `bm_ccv:int_ccv` to **BM cartridges** path.

BM > Administration > Manage the Business Manager site > Cartridges

> Note: **bm_ccv** always needs to be in front of **int_ccv**
> **bm_ccv** and **int_ccv** need to be in front of **bm_app_storefront_base, app_storefront_base,** and **modules**

1. In **site-import-data/sites** rename the *RefArch* folder to the name of your site (site name in BM).
2. Import the **site-import-data** folder in BM via Site Import.
3. Validate there are no errors with the site import. Validate the following got imported correctly:

> ⚠ You need to manually set the CCVPayment.credential password using the credentials in 1Password CCV vault! The Site Import will warn you that it cannot decrypt the password.

⚠ If you get a DATAERROR with invalid jobstepId verify that:

1. The cartridges are uploaded to the **active** code version
2. The cartridges are assigned to the site **and** to the Business Manager

If there’s still an error, it might be a caching issue. Switch back and forth between code versions to invalidate the cache and import again.

Administration > Operations > Services:

CCVPayment service, CCVPayment.profile, CCVPayment.credential

Merchant Tools > Ordering > Payment Processors

CCV_DEFAULT payment processor

Merchant Tools > Ordering > Payment Methods

- CCV_CREDT_CARD
- CCV_PAYPAL
- CCV_BANCONTACT
- …

PWA setup

1. Install the Salesforce PWA kit app v.2.6.0 (if you don’t already have it) https://developer.salesforce.com/docs/commerce/pwa-kit-managed-runtime/guide/quick-start.html
2. Place the ccv-pwa-plugin folder in the root of the PWA kit app (on same level as the App folder)
3. Make the following changes to the base PWA files:

3.1 Set the ocapiVersion

Add the `ocapiVersion` to the appConfig files of yout pwa-kit project.

```
module.exports = {
    app: {
        commerceAPI: {
            ...
            ocapiVersion: 'v21_3',
            ...
        }
    }
}
```

3.2 app/commerce-api/hooks/useBasket.js

```
// extend the useBasket#self#loaded()

get loaded() {
  return basket && (basket.basketId || basket.c_order_status_pending)
},
```

3.3 app/commerce-api/index.js

```
// 1. Add at top level

import OcapiCCV from ' ../../ccv-pwa-plugin/pages/checkout/util/ocapi-ccv'
```

```
// 2. Add to CommerceAPI#apiConfigs (on same level as shopperSearch)

ccvPayment: {
    api: OcapiCCV
}
```

3.4 app/pages/checkout/partials/cc-radio-group.jsx

```
// 1. Add onClick function to the “Remove” button in CCRadioGroup

<Button variant="link" size="sm" colorScheme="red" onClick={() => customer.removeSavedPaymentInstrument(payment.p
```

3.5 app/routes.jsx

```
// 1. Add at top level (replace the base Checkout component with the ccv one)
const Checkout = loadable(() => import('../ccv-pwa-plugin/pages/checkout'), {fallback})
const CheckoutRedirect = loadable(() => import(' ../ccv-pwa-plugin/pages/checkout-redirect'), {fallback})
```

```
// 2. Add this route before the {*} route
{
        path: '/:locale/checkout/handleShopperRedirect',
        component: CheckoutRedirect,
        exact: true
},
```

3.6 app/commerce-api/hooks/useShopper.js

```
// Add the following code snippet after the hasBasket declaration in the useShopper useEfect
/* existing code */
// Handle basket init/updates in response to customer/basket changes.
useEffect(() => {
  const hasBasket = basket?.loaded
  /* existing code */
  // if c_order_status_pending exists, the order payment is being checked
  // and we don't want to create a new basket so we return early
  if (hasBasket && basket.c_order_status_pending) {
      return
  }
}
```

3.7 app/pages/checkout/confirmation.jsx

```
// on top level add:
import React, {useEffect, useContext, useState, Fragment} from 'react'
import {BasketContext} from '../../commerce-api/contexts'
// in CheckoutConfirmation component add this at the top:
const {setBasket} = useContext(BasketContext)
// . . . and this at the end of the component
// we remove c_order_status_pending to unblock the creation of a new basket
useEffect(() => {
    setBasket({
        ...order,
        c_order_status_pending: false
    })
}, [])
```

3.8 app/ssr.js

```
// Add the apple pay url to the script-src directive in helment:
/* ... */
  'script-src': ["'self'", "'unsafe-eval'", 'storage.googleapis.com', "applepay.cdn-apple.com"],
/* ... */
```

3.9 app/components/icons/index.jsx

```
// Add the following code at top level

// ==================== CCV ICONS ====================

import IdealSymbol from '../../../ccv-pwa-plugin/assets/svg/ideal-logo.svg'

import BancontactSymbol from '../../../ccv-pwa-plugin/assets/svg/bancontact-logo.svg'

import GiropaySymbol from '../../../ccv-pwa-plugin/assets/svg/giropay-logo.svg'

import SofortSymbol from '../../../ccv-pwa-plugin/assets/svg/sofort-logo.svg'

import EPSSymbol from '../../../ccv-pwa-plugin/assets/svg/eps-logo.svg'

import PayconiqSymbol from '../../../ccv-pwa-plugin/assets/svg/payconiq-logo.svg'

import MaestroSymbol from '../../../ccv-pwa-plugin/assets/svg/maestro-logo.svg'

import KlarnaSymbol from '../../../ccv-pwa-plugin/assets/svg/klarna-logo.svg'

import AppleSymbol from '../../../ccv-pwa-plugin/assets/svg/apple-pay-logo.svg'



AmexSymbol.viewBox = AmexSymbol.viewBox || '0 0 38 22'

MastercardSymbol.viewBox = MastercardSymbol.viewBox || '0 0 38 22'

PaypalSymbol.viewBox = PaypalSymbol.viewBox || '0 0 80 20'

VisaSymbol.viewBox = VisaSymbol.viewBox || '0 0 38 22'

IdealSymbol.viewBox = IdealSymbol.viewBox || '0 0 306.1 269.8'

BancontactSymbol.viewBox = BancontactSymbol.viewBox || '0 0 326.1 230.5'

GiropaySymbol.viewBox = GiropaySymbol.viewBox || '0 0 38 22'

SofortSymbol.viewBox = SofortSymbol.viewBox || '0 0 746.1 286.2'

```

```
EPSSymbol.viewBox = EPSSymbol.viewBox || '0 0 889 577'

PayconiqSymbol.viewBox = PayconiqSymbol.viewBox || '0 0 326 230.5'

MaestroSymbol.viewBox = MaestroSymbol.viewBox || '0 0 125 120'

KlarnaSymbol.viewBox = KlarnaSymbol.viewBox || '0 0 100 40.4494'

AppleSymbol.viewBox = AppleSymbol.viewBox || '0 0 165.52107 105.9651'



// Export Chakra icon components that use our SVG sprite symbol internally

// For non-square SVGs, we can use the symbol data from the import to set the

// proper viewBox attribute on the Icon wrapper.

export const IdealIcon = icon('ideal-logo', {viewBox: IdealSymbol.viewBox})

export const BanContactIcon = icon('bancontact-logo', {viewBox: BancontactSymbol.viewBox})

export const GiropayIcon = icon('giropay-logo', {viewBox: GiropaySymbol.viewBox})

export const SofortIcon = icon('sofort-logo', {viewBox: SofortSymbol.viewBox})

export const EPSIcon = icon('eps-logo', {viewBox: EPSSymbol.viewBox})

export const PayconiqIcon = icon('payconiq-logo', {viewBox: PayconiqSymbol.viewBox})

export const MaestroIcon = icon('maestro-logo', {viewBox: MaestroSymbol.viewBox})

export const KlarnaIcon = icon('klarna-logo', {viewBox: KlarnaSymbol.viewBox})

export const ApplePayIcon = icon('apple-pay-logo', {viewBox: AppleSymbol.viewBox})
```

Optional changes

If the ***ccvScaReadyEnabled*** site preference is enabled the customer data fields sent in the payment creation request become mandatory instead of optional. In this case, CCV will require **a phone country code** in the request. It is not present in the base PWA address forms, so we need to add it to the address fields component.

If the merchant already has such a field in their address form, you will need to make sure it is sent in the OCAPI request to/addresses as "*c_phone_country*"

If the field is not already present, this is how it can be added:

3.10 app/components/forms/useAddressFields.jsx

```
// Add phoneCountry to messages. Remember to also add the translations.
lastName: {defaultMessage: 'Last Name', id: 'use_address_fields.label.last_name'},
phoneCountry: {defaultMessage: 'Phone Country', id: 'use_address_fields.label.phone_country'},
phone: {defaultMessage: 'Phone', id: 'use_address_fields.label.phone'},
```

```
// Add phoneCountry to useAddressFields#fields

```

```
phoneCountry: {

           name: `${prefix}c_phone_country`,

           label: formatMessage(messages.phoneCountry),

           defaultValue: '',

           type: 'tel',

           inputProps: ({onChange}) => ({

               inputMode: 'numeric',

               maxLength: 4,

               onChange(evt) {

                   onChange(evt.target.value.replace(/[^0-9 ]+/, ''))

               }

           }),

           rules: {

               required: formatMessage({

                   defaultMessage: 'Please enter your phone country code.',

                   id: 'use_address_fields.error.please_select_your_phone_country'

               })

           },

           error: errors[`${prefix}c_phone_country`],

           control

       },
```

3.11 app/components/forms/address-fields.jsx

```
// Add phoneCountry to the AddressFields component.

<SimpleGrid templateColumns="1fr 4fr" gap={5}>
    <Field {...fields.phoneCountry} />
    <Field {...fields.phone} />
</SimpleGrid>
```

3.12 Unit tests need to be adjusted to account for the new field:

3.13.a /app/commerce-api/mock-data.js

```
// add to mockedRegisteredCustomer#addresses[0]
c_phone_country: '123',
```

3.13.b /app/pages/checkout/index.test.js

```
In the “Can edit address during checkout as a registered customer”  test, add c_phone_country to the rest.put('*

rest.put('*/shipping_address', (req, res, ctx) => {

            const shippingBillingAddress = {

                /* ... */

                c_phone_country: '123',

                /* ... */

            }



In the same test, add this at the // Add addres section

    user.type(screen.getByRole('textbox', {name: /phone country/i}), '123')



In the “Can proceed through checkout steps as guest”  test add this at the // Fill out shipping address form and

    user.type(screen.getByLabelText(/phone country/i), '123')
```

Payment lifecycle overview

When the user reaches the payment step in the checkout, they are presented with the available payment methods.

The user selects a payment method and clicks “Review Order” – the selected payment instrument with the required data to their basket.

When the user clicks “Place order”, an order is created in SFCC, and a CCV payment is created in the order afterPOST extension hook. If the payment creation request fails, the order creation is rolled back and no order is created in SFCC and an error message is shown to the user.

If the payment creation request is successful, the customer gets redirected to the CCV-hosted pay URL, where they can authorize the payment.

When the user has completed or cancelled their payment, CCV notifies SFCC that the payment status has updated via a webhook. The webhook is handled in the SFCC controller endpoint CCV-WebhookStatus. SFCC gets the passed reference from CCV makes a service call to CCV to check the new status of the transaction, and updates the SFCC order status and payment attributes accordingly.

In the meantime, the customer is redirected back to the payment-return page in the PWA app. On this page, up to 10 service calls are made to SFCC in one second intervals to check the order status.

The number of calls or interval duration can be adjusted by a developer according to business requirements via a simple code change.

If the order status has changed to “New” – the customer is redirected to the order confirmation page.

If the order status has changed to “Failed” – the customer is redirected back to checkout payment step, the basket is reopened and an error is shown.

If the status has not changed and is still “Created” after the 10 service calls, the customer is still redirected to the order confirmation page. This can also be customized by a developer if the business require that a different page should be shown if the order status is still “Pending”.

Information about the payment transaction and reasons for a failed order can be found on the ***Payment*** and ***Notes*** tabs of the order in Business Manager.

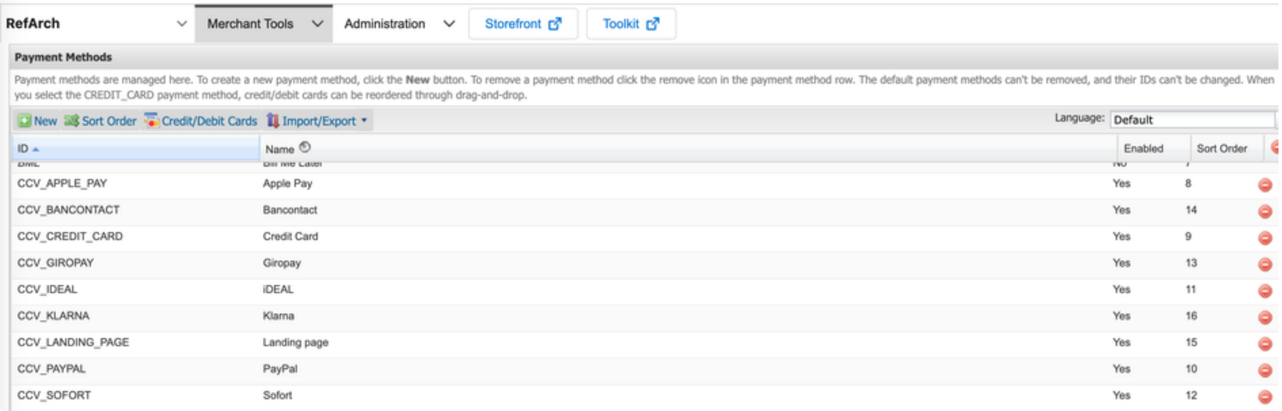
Configuration

API Key

In *Administration* > *Operations* > *Services* open the ***CCVPayment* credential** and in put your CCV API key the “**Password**” field.

Payment methods

Payment methods can be configured at *Merchant Tools* > *Ordering* > *Payment Methods*.



Payment Methods

Payment methods are managed here. To create a new payment method, click the **New** button. To remove a payment method click the remove icon in the payment method row. The default payment methods can't be removed, and their IDs can't be changed. When you select the CREDIT_CARD payment method, credit/debit cards can be reordered through drag-and-drop.

ID	Name	Enabled	Sort Order
BML	Bill Me Later	No	7
CCV_APPLE_PAY	Apple Pay	Yes	8
CCV_BANCONTACT	Bancontact	Yes	14
CCV_CREDIT_CARD	Credit Card	Yes	9
CCV_GIROPAY	Giropay	Yes	13
CCV_IDEAL	iDEAL	Yes	11
CCV_KLARNA	Klarna	Yes	16
CCV_LANDING_PAGE	Landing page	Yes	15
CCV_PAYPAL	PayPal	Yes	10
CCV_SOFORT	Sofort	Yes	12

All CCV payment methods have the CCV_ prefix. Payment methods can be enabled or disabled via the “Enabled” checkbox.

The “Name” value can be customized by the merchant – this is the name that appears on the storefront in the payment form component. It is localizable so the value can differ for each locale.

The order of payment methods can be sorted via the “Sort Order” button.

The payment methods’ **ID** values and the the custom attribute **ccvMethodId** should **never** be changed!

When enabling CCV payment methods, make sure all non-CCV payment methods are disabled – methods which don't have the *CCV_* prefix are not compatible with this implementation.

Apple Pay

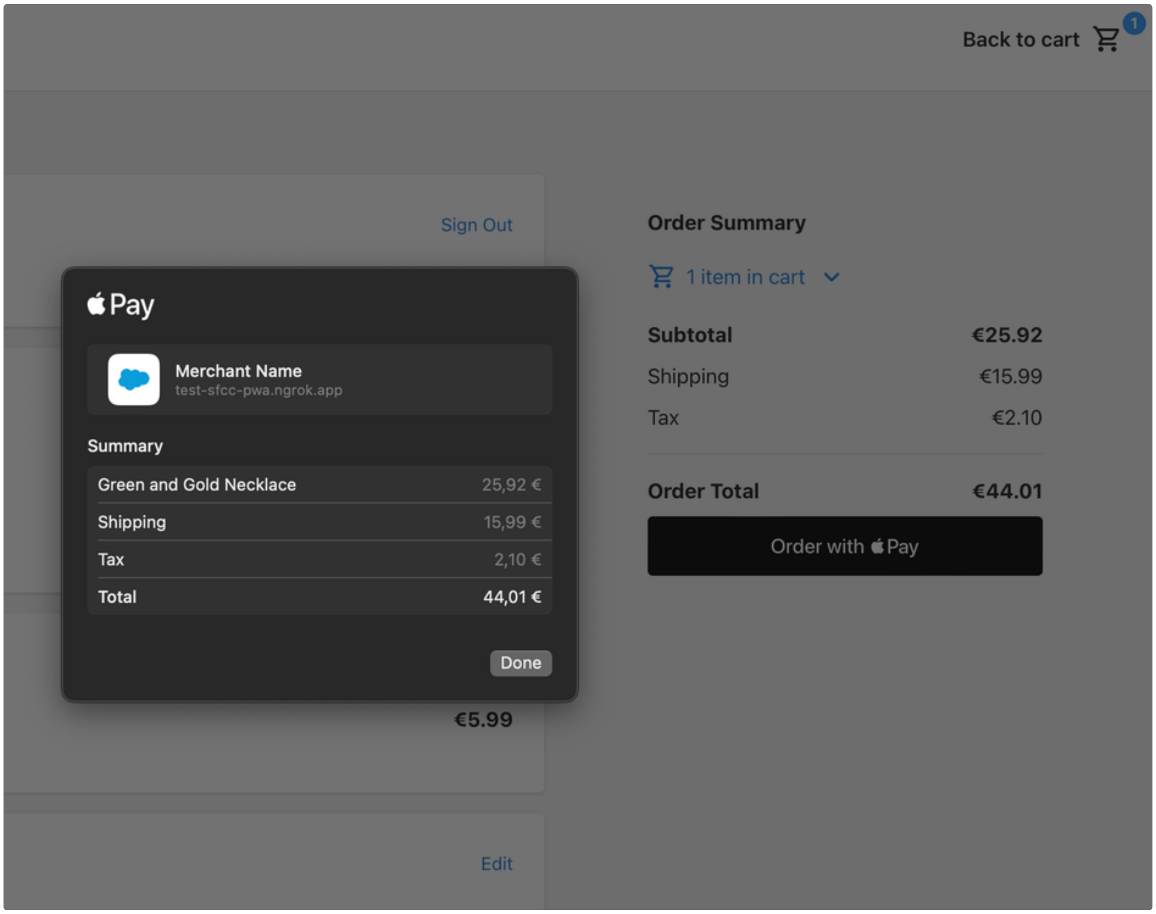
Apple Pay is available on Apple devices running Safari or iOS.

The Apple Pay implementation in this cartridge uses the hybrid approach – the Apple developer account, including apple merchant IDs and certificates are configured by CCV.

CCV customer support can configure and enable Apple Pay for their account.

The merchant usually needs to:

1. Provide a domain name where the Apple Pay button will be hosted – needs to be public and hosted over HTTPS
2. Create an Apple Pay tester account, in order to test the configuration on their test environment.
3. Add their merchant name to the /ccv-pwa-plugin/ccvConfig.js.- **applePayMerchantLabel.** This is the label that will appear in the apple pay sheet:

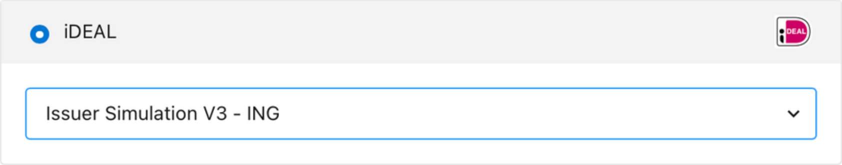


Green and Gold Necklace	25,92 €
Shipping	15,99 €
Tax	2,10 €
Total	44,01 €

Subtotal	€25.92
Shipping	€15.99
Tax	€2.10
Order Total	€44.01

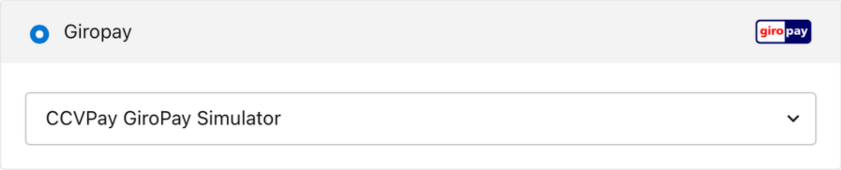
Giropay and iDEAL

Giropay and iDEAL are payment methods, where the customer needs to select some additional options:



○ iDEAL

Issuer Simulation V3 - ING



○ Giropay

CCVPay GiroPay Simulator

The available options are retrieved dynamically from CCV api via the /method API.

In order to improve performance, the response of this web service call is cached for 10 minutes via a custom cache (if caching is enabled in Business Manager for the current site).

If the merchant wishes to change this cache duration, a developer needs to adjust the code and edit the expiration of the **ccvPaymentMethodsCache** custom cache, in link_ccv/cartridges/int_ccv/caches.json.

The cache can also be refreshed by invalidating the site cache in Buisness Manager.

Operation

Jobs

CCVPayment-CheckOrderTransactionStatuses

This job processes orders in status "Created", older than the threshold passed via the job parameter **cutoffPeriodInDays**, and not flagged as needing manual intervention (order.custom. ccvManualIntervention).
It attempts to authorize the CCV payment on each of the orders passing the condition and move the orders to status "New" or "Failed" using the same logic that the webhook endpoint uses.
If the "Auto Refund Enabled" site preference is enabled, these orders will be automatically marked for refund.
It is recommended to be ran often, for example every 10 minutes.

CCVPayment-ProcessRefunds

This job processes orders with pending refunds (order.custom.ccvHasPendingRefunds)

It checks the refund transaction status in CCV and updates the refund object on the SFCC order (order.custom.ccvRefunds) accordingly. This job needs to be configured to run regularly.

Custom preferences

The following custom preferences are available:

Preference name	Possible values	Description
ccvCardsAuthoriseEnabled	Yes/No	If set to "Yes" – credit card payments will be created with type

		“authorise”. If set to “No” – the type will be “sale”. When payments with type “sale” reach status “success”, the order payment status in SFCC will be set to “PAID”.
ccvStoreCardsInVaultEnabled	Yes/No	If set to “Yes”, the customer can choose to use the “remember card” functionality - the credit card data of the customer will be tokenized in the CCV vault, and the token can be used on subsequent payments
Auto Refund Enabled	Yes/No	If set to “Yes” automatic refunds will be triggered for orders where the payment amount or currency in SFCC is not matching the ones in CCV.
ccvScaReadyEnabled	Yes/No	
ccv3DSExemption	TRANSACTION_RISK_ANALYSIS LOW_VALUE SECURE_CORPORATE TRUSTED_BENEFICIARY	If a value is selected, it will be added as an exemption to each payment request. This may shift the liability on the merchant.

Custom order attributes

The following new order custom attributes are added - these attributes are “read-only”, and are only supposed to be updated programmatically.

Attribute name	Description
ccvTransactionReference	CCV reference of the payment transaction
ccvRefunds	Array containing refund objects. When a refund request is created, a new object with the refund data is added here
ccvHasPendingRefunds	Indicates if the order has some refunds which have status “pending”
ccvManualIntervention	Indicates the payment for this order has reached “manuialintervention” status. Payments with “manualintervention” status need to be handled in the CCV control panel.
ccvPriceOrCurrencyMismatch	Indicates there is price or currency mismatch between the SFCC order total amount, and CCV

	payment amount
ccvPayUrl	CCV-hosted URL the customer needs to go to to complete their payment
ccvManualInterventionRefund	Set to true if some refund requires manual intervention
ccvCardDataUrl	URL used during checkout to send apple pay token data
ccvCancelUrl	URL used during checkout to cancel a payment

Payment cancellations, refunds and reversals

Payments for an order can be canceled, refunded, or reversed via a Customer Service Center action.

A payment can be cancelled up to a certain point in the payment lifecycle – usually before the customer has authorized it. If the payment has already been authorised in the CCV system, but the webhook still hasn't updated the order status, the option to cancel will still appear but it won't have any effect. The customer service agent will need to wait for the webhook to update the order before attempting to refund or reverse the payment.

Payments with type "**authorise**" can be reversed – this reverses the authorization for the whole amount.

Payments with type "**sale**" can be refunded – a refund can be partial or full.

Refunds and reversals can also be created automatically on price/currency mismatch between SFCC and CCV if the custom site preference – "**Auto Refund Enabled"** is enabled.

Automatic refunds/reversals are created by the webhook endpoint *CCV-WebhookStatus* or by the job *CCVPayment-CheckOrderTransactionStatuses*.

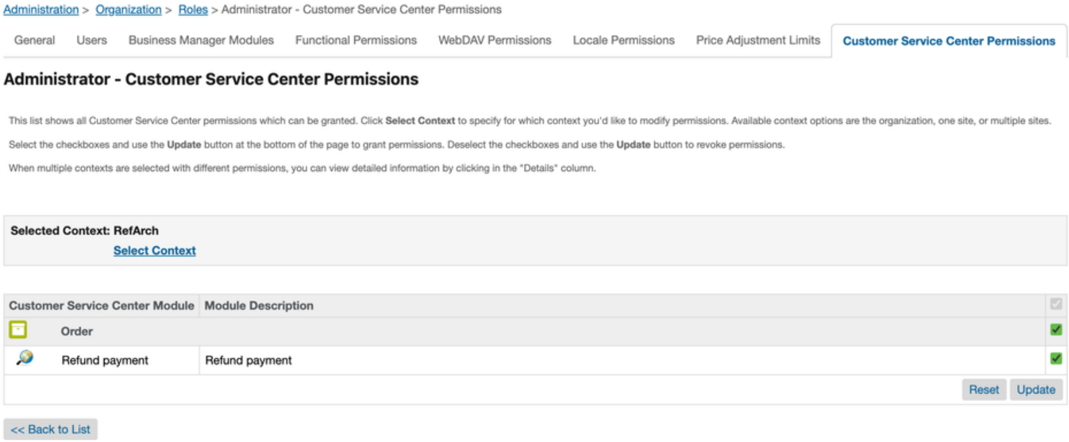
After a refund is created, it is added as an object to the order's **ccvRefunds** custom attribute. The statuses of all refunds is usually updated later via the *CCVPayment-ProcessRefunds* job.

Manual cancel / refund / reverse

- Manual refunds can be created via Customer Service Center order action.
- Manual refunds are not available for orders with status "Cancelled" or "Failed".
- Orders with status "Created" can only be cancelled.
- The order total is the maximum refundable amount.
- Multiple refunds can be created for payments with type "capture" and the amount can be chosen by the CSC agent.

Steps for creating a manual cancel / refund / reverse

1. **In order the see the refund action, the BM user needs to have the “Refund CCV payment” role**. The role can be added by an administrator at:
2. **Administration > Organization > Roles > Customer Service Center Permissions**



Administration > Organization > Roles > Administrator - Customer Service Center Permissions

Administrator - Customer Service Center Permissions

This list shows all Customer Service Center permissions which can be granted. Click **Select Context** to specify for which context you'd like to modify permissions. Available context options are the organization, one site, or multiple sites.

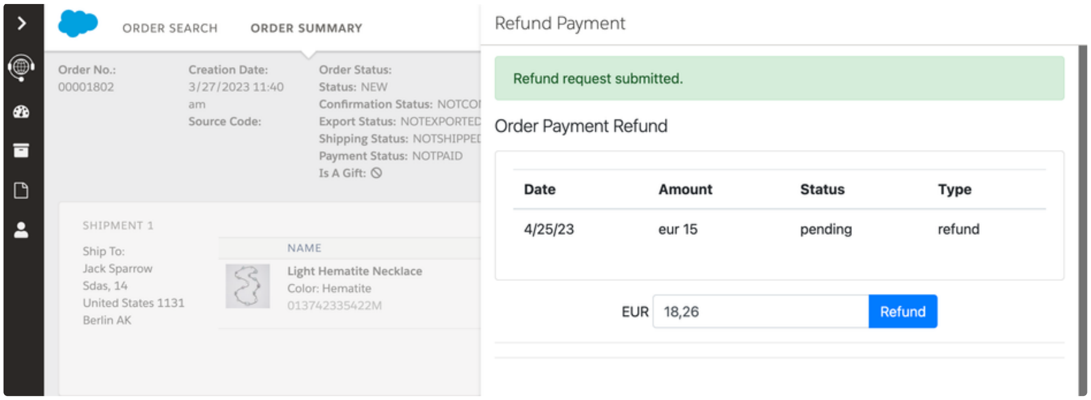
Select the checkboxes and use the **Update** button at the bottom of the page to grant permissions. Deselect the checkboxes and use the **Update** button to revoke permissions.

When multiple contexts are selected with different permissions, you can view detailed information by clicking in the "Details" column.

Selected Context: RefArch
Select Context

Customer Service Center Module	Module Description	☐
Order		☑
Refund payment	Refund payment	☑

3. Go to CSC:
   Merchant tools > Ordering > Customer Service Center
4. Find and open the order you want to refund via Order Search.
5. Click on “More” in upper right corner.
   Note: If you don’t see the “More” button, or the “Refund payment” after clicking “More”, you either don’t have the role from step.1 or the BM cartridge is not added to the cartridge path (see Installation > BM Setup)
6. Click on “Refund payment”
7. Select an amount and click “Refund” (captured payments are eligible for full or partial refund, while authorized payments are only eligible for full refund)
8. A new refund will be created on the order (stored in order.custom.ccvRefunds)



Order No.: 00001802
Creation Date: 3/27/2023 11:40 am
Source Code:
Order Status:
Status: NEW
Confirmation Status: NOTCO
Export Status: NOTEXPORTED
Shipping Status: NOTSHIPPED
Payment Status: NOTPAID
Is A Gift: ⃠

SHIPMENT 1

Ship To:
Jack Sparrow
Sdas, 14
United States 1131
Berlin AK

NAME
Light Hematite Necklace
Color: Hematite
013742335422M

Refund Payment

Refund request submitted.

Order Payment Refund

Date	Amount	Status	Type
4/25/23	eur 15	pending	refund

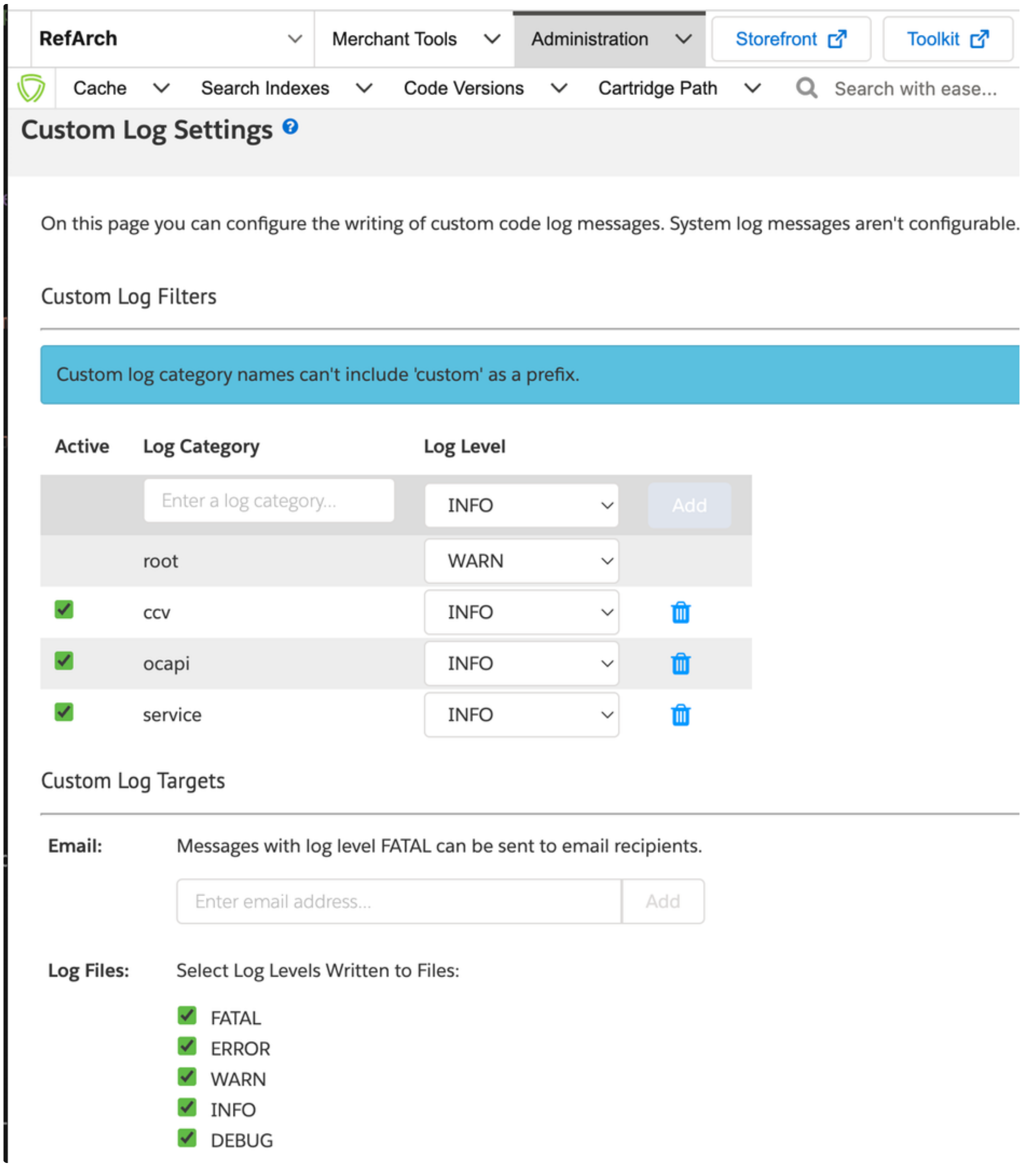
EUR 18,26

1. The refund status should be updated after a few seconds by the webhook trigerring our CCV-WebhookRefund controller.
2. Refunds can also be updated by the **CCVPayment-ProcessRefunds** job when it runs.

Logging

Custom logging for some CCV jobs and services can be found in logs with “custom-CCV-“ prefix. The lowest log level in these logs is “Info”. These logs can be enabled in Buisness Manager, via

Administration > Custom log settings by adding a log category “ccv” (lowercase). The log level can be set to “warn”, or “info” – if more detailed logs are needed.



Custom Log Settings

On this page you can configure the writing of custom code log messages. System log messages aren't configurable.

Custom Log Filters

Custom log category names can't include 'custom' as a prefix.

Active	Log Category	Log Level	
	Enter a log category...	INFO	Add
	root	WARN	
☑	ccv	INFO	🗑
☑	ocapi	INFO	🗑
☑	service	INFO	🗑

Custom Log Targets

Email: Messages with log level FATAL can be sent to email recipients.

Enter email address... Add

Log Files: Select Log Levels Written to Files:

☑ FATAL
☑ ERROR
☑ WARN
☑ INFO
☑ DEBUG

Unit testing

There are 2 sets of unit tests – one for SFCC server-side code, and one for the PWA kit.

SFCC tests:

‘npm run test’ in link_ccv

PWA kit tests:

‘npm run test’ in root folder of the PWA kit

Known issues

During the guest checkout the user can proceed up to the order review step with an invalid email address without getting an error. After clicking the “Place order” button the order will not get placed, an a general error message will be shown, which might not make it clear to the user what the problem is.

The merchant should make sure to add appropriate validation to the email form field during checkout.