CCV PWA Link Cartridge

# 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 architecture of the Salesforce PWA kit.

CCV uses a hosted payment page approach – the customer is redirected to a webpage, hosted by CCV, where they can complete their payment.

The payment methods supported by this link cartridge are:

* Credit cards – VISA, Mastercard, Maestro, Amex, Bancontact
* PayPal
* GiroPay
* iDEAL
* Sofort
* Payconic
* EPS
* Apple
* SDD (Sepa direct debit)

The default credit card payment is using a hosted payment page approach.

We have also added an inline credit card component – however**, in order to use the inline credit card form, the merchant needs to have a PCI compliant site**, because sensitive payment information passes through the SFCC backend in that case.

Manual refunds are supported via a Customer Service Center action.

# Compatibility

Node .v 14.17.6

PWA kit app – v.2.6.0

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.

# 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

1. 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**

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:

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. Copy the files from pwa-new-files to your pwa-kit folder
3. Make the following changes to the base PWA files:

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

// extend the useBsket#self#loaded()

get **loaded**() {

return basket && (basket.basketId || basket.c\_order\_status\_pending)

},

### 3.2 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**

}

### 3.3 app/pages/checkout/index.jsx

// 1. Add at top level

import CCVPayment from ' ../../../ccv-pwa-plugin/pages/checkout/partials/payment-ccv'

import useCCVApi from ' ../../../ccv-pwa-plugin/pages/checkout/util/useCCVApi'

import {CCVPaymentProvider, useCCVPayment} from ' ../../../ccv-pwa-plugin/pages/checkout/util/ccv-context '

// 2. Add to the **Checkout**() function scope

const ccv = **useCCVApi**()

const {paymentError, setPaymentError} = **useCCVPayment**()

// 3. Replace the submitOrder function with this:

const **submitOrder** = async () => ccv.**submitOrderCCV**(**setIsLoading**, setPaymentError)

// 4. Change the useEffect condition in Checkout() to this:

if (globalError || (step === 4 && !paymentError)) {...}

// 5. Replace <Payment> component with <CCVPayment /> in the **Checkout()** return JSX

// <Payment />

<**CCVPayment** />

// 6. Add <CCVPaymentProvider> around <Checkout /> in **CheckoutContainer()** but inside CheckoutProvider:

<**CheckoutProvider**>

<**CCVPaymentProvider**>

<**Checkout** />

</**CCVPaymentProvider**>

</**CheckoutProvider**>

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

// 1. Add onClick function to the “Remove” button in **CCRadioGroup –** only used in inline credit card component

<**Button** *variant*="link" *size*="sm" *colorScheme*="red" *onClick*={() => customer.**removeSavedPaymentInstrument**(payment.paymentInstrumentId)}>

### 3.5 pwa-kit-src/app/routes.jsx

// 1. Add at top level

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/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'

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'

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})

# 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*.

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 payment methods’ **ID** values should **never** be changed!

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

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.

## Jobs

1. **CCVPayment-CheckOrderTransactionStatuses**

This job attempts processes orders in status “Created”. It attempts to authorize the CCV payment and move the orders to status “New” or “Failed” depending on the transaction status. It also fails orders which have a currency or price mismatch between SFCC and CCV. 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.

1. **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.

## 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” |
| ccvStoreCardsInVaultEnabled | Yes/No | If set to “Yes”, the credit card data of the customer will be tokenized in the CCV vault, and the token will 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 one in CCV. |

## Custom 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 are “pending” |
| ccvManualInterventionRefund | Indicates some refund object in this order has “manuialintervention” status |
| 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 |

# Refunds and reversals

Refunds and reversals can be created either manually or automatically on price/currency mismatch.

Automatic refunds/reversals can be enabled or disabled via a custom site preference – “**Auto Refund Enabled”**.

Automatic refunds/reversals are created by the **CCVPayment-CheckOrderTransactionStatuses** job.

A refund can be created for payments with type “capture”.

A reversal can be created for payments with type “authorise”.

Refunds can be partial or for the full amount, while reversals can only be for the full amount.

## Manual refunds

Manual refunds can be created via Customer Service Center order action.

Manual refunds are not available for orders with status “Cancelled”, “Failed”, or “Created”.

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 refund

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** Graphical user interface, text, application

   Description automatically generated
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)

Graphical user interface, application

Description automatically generated

1. If the refund status is “pending”, the status of the refund will 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”.

# 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 pwa-kit-src

## Payment lifecycle flowchart

