CCV Cartridge Technical Documentation

Background and LINK cartridge documentation

This page is about the CCV LINK cartridge, which provides an integration of the CCV payment provider with the SF PWA-kit react app. This page will cover some of the more technical aspects.

The CCV API documentation can be found here:

CCV Pay

Payment lifecycle overview

1. When the user reaches the payment step in the checkout, they are presented with the available payment methods. Some payment methods (Giropay, Ideal) have additional options, which are added dynamically to the GET payment_methods response via the `payment_methods.modifyGETResponse_v2` hook extension.
2. The user selects a payment method and clicks “Review Order” – the selected payment instrument with the required data to their basket.
3. When the user clicks “Place order”, an order is created in SFCC, and a CCV payment is created in the order `afterPOST` extension hook. In the payment request, we set a webhookUrl, which CCV will call when the payment status changes in their system.
   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.
4. When the user has completed or cancelled their payment, CCV notifies SFCC that the payment status has updated via the webhook. The webhook for payments is handled in the SFCC controller endpoint `CCV-WebhookStatus`. SFCC gets the passed transaction reference ID 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.
5. 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 `GET /orders` in one-second intervals to check the order status:

- 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 the 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 requires that a different page should be shown if the order status is still “Pending”.

Installation

The installation instructions can be found in the Functional Cartridge Documentation.

Code structure

The implementation uses controllers for the webhooks handlers and OCAPI extension hooks for creating payment requests, enriching payment methods, and some utilities.

At the time of writing, there are 3 main subfolders in the repository:

```
|-- "link_ccv"
|   |-- "cartridges"
|   |-- "site-import-data"
|   |-- "test"
|   |-- ...
|-- "pwa-kit-src"
|   |-- "ccv-pwa-plugin"
|   |-- ...
|-- "sfra"
|   |-- "cartridges"
|   |-- ...
`-- "tree.txt"
```

`link_ccv` contains:

- `cartridges` - the custom cartridges related to CCV containing all controllers, OCAPI hook extensions, and server-side scripts related to the CCV services.
- `site-import-data` - site import metadata required for the CCV customizations
- `test` - unit tests for the server-side scripts

`pwa-kit-src` contains:

- `ccv-pwa-plugin` - CCV react components and scripts used for rendering checkout components, e.g. payment-selection, payment-return page, etc. This directory acts as an override directory to a base pwa-kit v3 application.

Main scripts (server-side)

Ordered by checkout flow:

Payment methods

`link_ccv/cartridges/int_ccv/cartridge/scripts/hooks/paymentMethodHooks.js`

used in the `OCAPI payment_methods modifyGETResponse_v2` extension hook to add options, retrieved from CCV `/method` API to some payment methods (iDEAL, Giropay). The CCV response is stored in a custom cache for 10 minutes.

CCV Payment creation

`link_ccv/cartridges/int_ccv/cartridge/scripts/hooks/orderHooksCCV.js`

Adds logic to `afterPOST` ocapi /orders hook, where we create a new payment in the CCV system, via a service call to the CCV `/payment` API.

Inside this request body, we set the webhookUrl to be called when the transaction’s status changes.

`link_ccv/cartridges/int_ccv/cartridge/scripts/services/CCVPaymentHelpers.js`

Contains helper functions for calling the CCV APIs. We only use one service (CCVPayment), and set the PATH in the helper functions.

Webhook status update

`link_ccv/cartridges/int_ccv/cartridge/controllers/CCV.js`

Controller with our webhook handlers and an extra handler for Apple Pay.

- `CCV-WebhookStatus` - for payments,
- `CCV-WebhookRefund` - for refunds/reversals

- `SubmitApplePayToken` - needed for an additional step in the Apple Pay flow, where we need to submit the authorized Apple Pay token to CCV. Called from our client side.

CCV will keep calling the webhooks until they return a 200 status.

`link_ccv/link_ccv/cartridges/int_ccv/cartridge/scripts/authorizeCCV.js`

Contains 2 functions:

- `authorizeCCV` makes a call to CCV `/transaction` endpoint to check the status of the transaction and returns an object with the authorization result
- `handleAuthorizationResult` - handles the authorization result. We have handlers for failed / success/ priceOrCurrenctMismatch / manualIntervention, updating the SFCC order status.

These 2 functions are only used in the webhooks and in the job CCV-UpdateTransactionStatuses - which should do the job of the webhook for stuck orders.

Cancellations / Refunds / Reversals

A payment can be canceled/refunded/reversed via a Customer Service Center (CSC) action. The difference between the 3 is explained in the LINK documentation.

`link_ccv/cartridges/bm_ccv/cartridge/controllers/CSCOrderPaymentRefund.js`

This is the controller used to extend the CSC. Provides logic for the 3 actions. In the CSC there is only one button and the logic determines which action is possible for the given order.

`link_ccv/cartridges/bm_ccv/cartridge/bm_extensions.xml`

Registers the custom action to the CSC.

Main scripts (client-side)

Customizations of base files

Some customizations of the base PWA app files are necessary (installation steps in the LINK documentation). Their main purposes are:

- to prevent a new basket from being created immediately on the payment return page
- adding some icons to the main sprite sheet
- adding phoneCountry field to the address forms, needed for 3DS flow (if scaReady site pref is enabled)

ccc-pwa-plugin

The custom CCV react components are placed in the `pwa-kit-src/ccv-pwa-plugin/app` directory.

checkout

`pwa-kit-src/ccv-pwa-plugin/app/pages/checkout/index.jsx`

The CCV checkout page - based on the checkout page from the base kit.

Adds a `CCVPaymentProvider` around the checkout component.

`pwa-kit-src/ccv-pwa-plugin/app/pages/checkout/confirmation.jsx`

The confirmation page is overwritten in order to display the CCV-used payment method in the summary section.

checkout-redirect

`pwa-kit-src/ccv-pwa-plugin/app/pages/checkout-redirect/index.jsx`

The payment return page the user is redirected back to after returning to the CCV-hosted payment page.

Utilities

`payment-ccv` and `payment-selection-ccv` are based on the same components in the base kit (without the ccv suffix).

`pwa-kit-src/ccv-pwa-plugin/pages/checkout/util/ocapi-ccv.js`

Creates an API proxy to access some OCAPI and controller endpoints, needed for the CCV checkout.

`pwa-kit-src/ccv-pwa-plugin/pages/checkout/util/useCCVApi.js`

Helper functions for order submission, mostly using the ocapi-ccv.

Services

`CCVPayment` - used in all calls to CCV payment APIs. The path to each resource is set in the code.

`ccv.ocapi.service` - OCAPI service - we reuse the customer JWT auth token, passed from client side, so no credentials are needed. We only use it in `CCV-SubmitApplePayToken` to try and retrieve the customer’s order, as an extra authorization steps - because the endpoint is not protected otherwise.

Jobs

Ideally, all orders should be updated by the webhook.

However, if some orders or refunds get stuck (in case the webhook failed/wasn’t called), we also have two jobs that will check the payment status of Created orders, or the refund status of orders with pending refunds:

`CCVPayment-CheckOrderTransactionStatuses`

`CCVPayment-ProcessRefunds`

Refund/ reverse / cancel

Refunds are created via a CSC action - more about this in the LINK documentation.

Note: To see the button you need to have the `Refund CCV payment ` role in BM.

The refunds are stored on Order level, in a custom attribute `order.custom.ccvRefunds` - it’s a JSON array of refund objects.

When a refund/reversal for an order is created, an object representing the refund is added to this array and the order is marked as `hasPendingRefunds`.

The refund status is updated later via the `CCV-WebhookRefund `handler, or via the job- `CCVPayment-ProcessRefunds`

The CCV api allows refunds for an amount larger than the order total (configurable in CCV), but in our implementation we limit it to the order total.

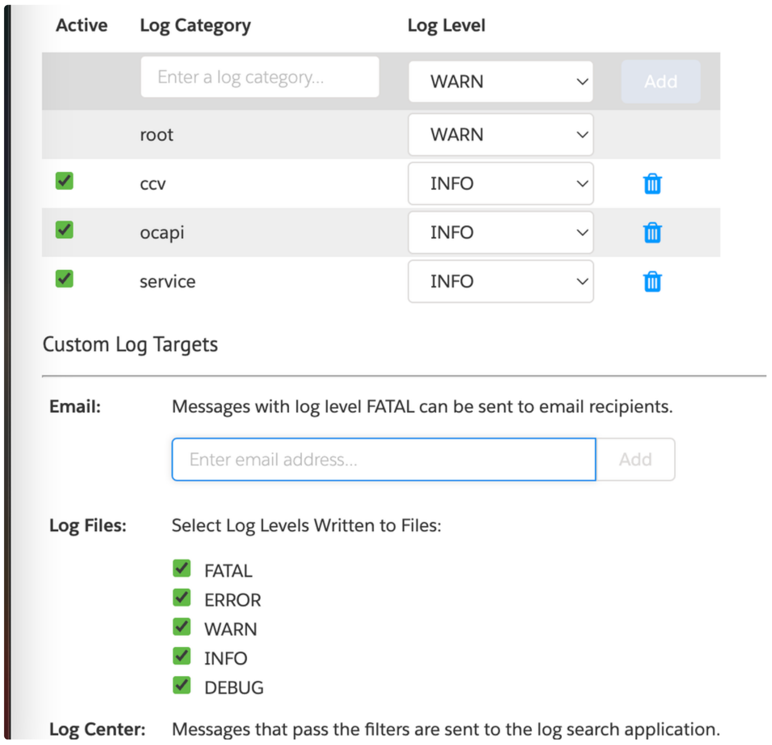
Apple Pay

The setup of Apple Pay requires some extra steps:

1. You need an Apple device, which can make payments - e.g. a MacBook, iPhone, iPad.
2. We only need an Apple Developer account to create a sandbox tester account - all the steps about merchant registration, certificates, etc are handled by CCV support. More about creating the tester account here: Sandbox Testing - Apple Pay
3. A public https URL is needed - CCV support needs this to register the domain name on their Apple Pay account. We can't use localhost. During development, I used the paid version of ngrok - it lets you register a persistent domain name to tunnel your localhost server.
   CCV registered these URLs as an example:
   https://demo-client-ccv.mobify-storefront.com (MRT)
   https://test-sfcc-pwa.ngrok.app (custom ngrok domain)
   If development is needed on a different domain, CCV support needs to be asked to register it.

Logging

To enable the custom logger, add a log category `ccv` (lowercase) on level info, and make sure info messages are written to files.



Active	Log Category	Log Level	
	Enter a log category...	WARN	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

Log Center: Messages that pass the filters are sent to the log search application.