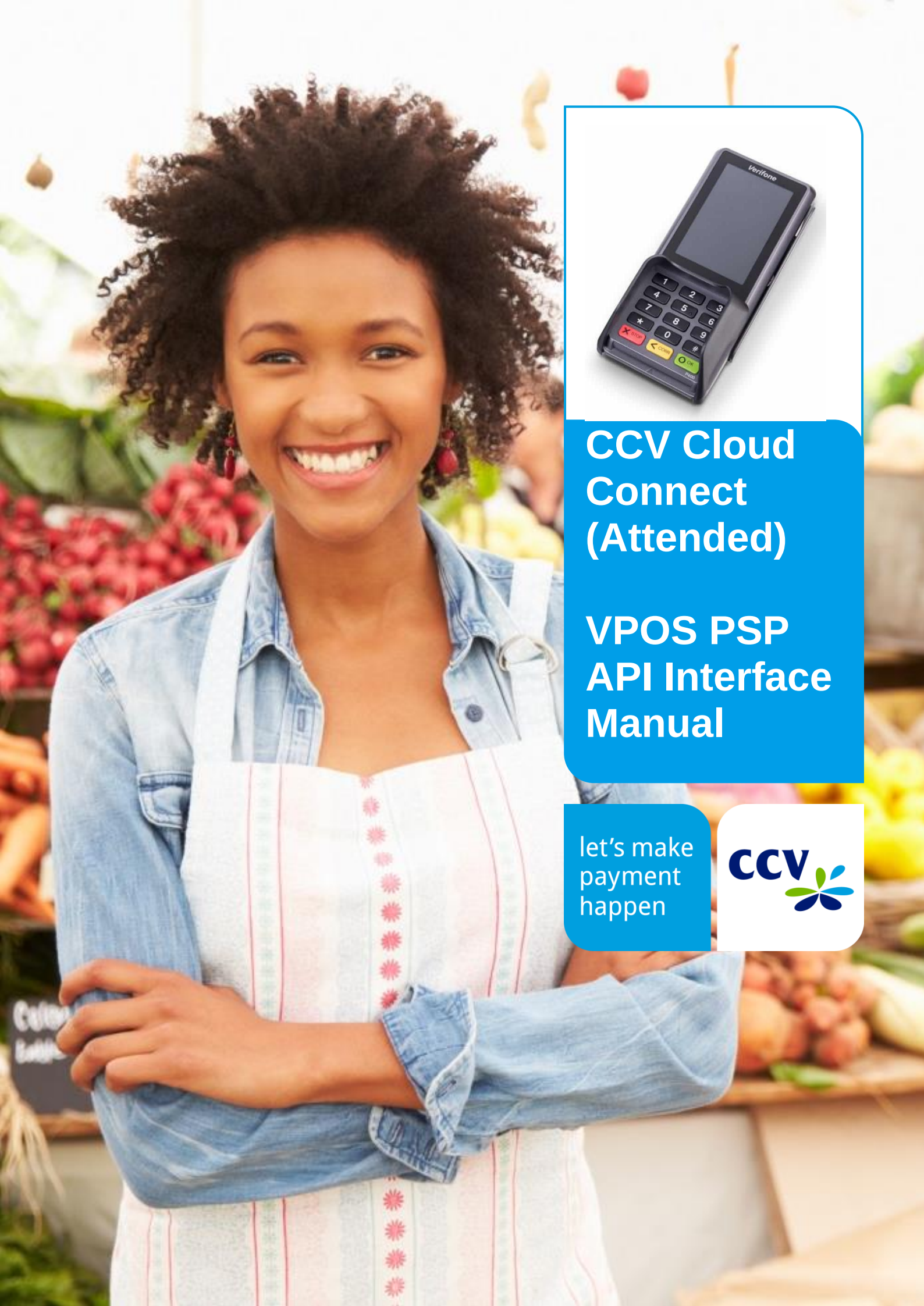


CCV Cloud Connect (Attended)

VPOS PSP API Interface Manual

let's make payment happen

ccv

Table of Contents

1 Document Properties

1.1 Document History

Date	Version	Name	Remarks
09-05-2019	1.0	Ben van de Put	First release for CAT attended based on CAT semi unattended version 1.0
3-12-2021	2.0	Ben van de Put	CAT renamed to CCC (CCV Cloud Connection) Cloudkick changes are added, being: • New managementSystemId • and removal of all functionality related to the local kick Other changes: • new url for production environment • IdemPotency reference is added, to avoid double transactions • Parameters for usage of simulated terminal adjusted to actual values
23-5-2022	2.1	Ben van de Put	The fact the every merchant needs an API key is stated more clearly The transaction type Refund is added.
8-6-2022	2.2	Ben van de Put	Review comments of CCV Lab on v2.1 are adapted • Mistakes in Refund request are corrected • The list of Errors and Error Codes in section 5.4 Error Handling are extended • Small textual corrections

1.2 Review

Date	Version	Reviewed by	Remarks
27-5-2022	2.1	CCV Lab (JL)	

1.3 Distribution List

Name	Role
CCV Internal	
Integrators	SW engineers

1.4 Copyright

2 Introduction

Payment terminals are used in various environments. These environments are:

- Attended environment: Attended is defined as a situation where a cashier starts a transaction. In this environment various transaction types (like sale, refund, pre-authorisation, sale after reservation, etc.) and various cardholder verification methods (CVM) (like: authentication by PIN, signature or no authentication)are used;
- Unattended environment: Unattended is defined as a situation where a cardholder or automated process starts a transaction. In these environments full weather resistant and vandal resistant (and therefor expensive) payment terminals are used. In an unattended environment only sale transactions are allowed and authentication with signature is not allowed;
- Semi Unattended environment: similar to Unattended environment but done with the same terminals as attended environment. This means that it is *mandatory* to use this solution indoor (the terminal is not weather resistant) and have staff available to guard the equipment (the terminal is not vandal resistant). When a shop is closed and staff is not available the equipment must not be accessible for public.

The equipment generating the amount to be paid (attended: an electronic cash register, unattended and semi unattended: a kiosk) is called POS (Point Of Sale).

Classic Payment Terminals are operated by an application on a POS or server running in the same store. The connection between POS and terminal is by RS232 or Ethernet LAN.

When a POS application (attended, unattended or semi unattended) is browser based and the POS application runs in the cloud, usage of public internet causes problems, because:

1. The classic protocols between the POS and Payment Terminal don't support authentication and encryption, which make these protocols unsuitable for transport using public internet without extra measures;
2. A transaction is normally initiated by the POS (in this case a POS located in the cloud) and nearly all merchants don't allow inbound traffic into there local network environment. Firewalls simply block every connection made from the outside to the LAN.

CCV integrated terminals for attended and semi unattended use (ITS Vx820 and ITS P400) support both the classic mode and a new mode called CCC (CCV Cloud Connection).

The CCC mode is offered as a PSP service. Instead of a website starting credit card, iDEAL or PayPal transactions using the CCV PSP, the website (or POS running in the cloud) starts a CCC transaction using the same PSP. Now, using the same PSP protocol with small extensions, the transaction is forwarded and processed by the Payment Terminal located next to the platform running a browser connected to the cloud based POS application.

To be able to do that, the website must know which PINpad has to do the transaction, because if a cardholder uses a kiosk A to order products, the payment terminal in kiosk A has to do the transaction. Or when POS 1 in shop location N is used by a cashier, the terminal located at POS 1 has to be used to do the transaction. To be able to use the CCC solution, the web application has to know the PINpad ID of the Payment Terminal.

2.1 Purpose of this document

This document describes the VPOS PSP API. This is the interface between the cloud based POS application and the PSP. The PSP controls the payment terminal.

This document only describes the protocol for attended use.

The current version of attended is restricted. It only supports:

- Sale and refund transactions (no reservation etc.);
- a one to one relation between POS and payment terminal (multiple POS devices sharing one payment terminal is not yet supported).

Another document intended for the web based POS supplier is a document describing a number of test scenario's. This document helps the supplier to test the implementation in a decent way. See ref [1] for more details.

2.2 Status

This is version 2.0 of document Cloud Accessible Terminal. The status of the document is Draft.

2.3 Terminology and abbreviations

Term/Abbreviation	Description
API	Application Programming Interface
API key	An API key is the key needed to have access to the PSP environment. The (intial) key is generated by CCV (Online Payments). A API key is merchant specific and one API key can be used to access multiple PINpads (of the same merchant) An API key may, for security reasons, never be stored on a local device in the shop
CAT	Cloud Accessible Terminal, former name of CCC
CCC	CCV Cloud Connection. PSP solution giving access to attended or semi unattended Payment Terminal in a secure way
C-TAP	Common Terminal Acquirer Protocol is a specification of the protocol, man machine interface (MMI) and security measures to process debit and credit card transactions by Payment Terminals
EFT	Electronic Funds Transfer
EFT terminals	Payment Terminal accepting debit cards and credit cards and processing the EFT transactions
ITS	Integrated Transaction Solution: range of products in the category of integrated terminals (both attended and unattended)
ITS Vx820	Integrated terminal for attended and semi unattended use based on VeriFone Vx820 terminal
ITS P400	Integrated terminal for attended and semi unattended use based on VeriFone P400 terminal
JSON	JavaScript Object Notation is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value)

Term/Abbreviation	Description
LAN	Local Area Network
PUI	Primary User Interface. Touchscreen of a kiosk operated by the cardholder
OPI	Open Payment Initiative: XML protocol to exchange data between POS (or TAS) and ITS terminal
POS	Point Of Sale or Electronic Cash Register (ECR attended or kiosk Unattended)
PSP	Payment Service Provider. Service offered by CCV for online payments initiated by a web shop. Now extended with the TAS enabling the use of the same interface for payments done on CCV ITS terminals
RFU	Reserved for Future Use
TAS	Terminal Access Server. PSP component transporting payments to multiple PINpads in a secure way.
TMS	Terminal Management System. System CCV managing the parameters, configuration and software of each individual payment terminal
VPOS	Virtual Point of Sale
VPOS PSP API	Protocol between web shop and PSP. The protocol for access to semi unattended terminals is described in this document

2.4 Typograhical conventions

To help the reader to get the most from the text in this document a number of typographical conventions are used:

Classification	Example
Important words are written in Italics	*Important*
Names or Values of JSON object names, parameter names and values are shown in Italics and between double quotes	"*Name of object*", "*Name of Parameter*" and "*Value*"
Sample traces are shown in Courier New 7	`Example trace`

Data formats

The following basic types are defined:

- Character;
- Boolean;
- Decimal digit;
- Hexadecimal digit.

Character

The basic *character* type is indicated with the letter '*a*'. The characters shall belong to range 0x20 to 0x7F of the non-extended ASCII character set.

Boolean

The *Boolean* type is indicated in this dictionary with the letter '*b*'. The Boolean type is coded "*true*" for *yes* and "*false*" for *no*.

Decimal digit

The *decimal digit* type is indicated in this dictionary with the letter '*i*'. The decimal digit values are in the range "*0*" through "*9*".

Hexadecimal digit

The *hexadecimal digit* type is indicated in this dictionary with the letter *'x'*. The values of hexadecimal digits are in the range "*0*" through "*9*" and "*A*" through "*F*" (capitals only).

Na-field

A field composed of N characters is defined as *'Na'* where the N is a positive integer number that gives the number of characters in the field.

Nb-field

A field composed of N Booleans is defined in the data dictionary as *'Nb'* where the N is a positive integer number that gives the number of Booleans in the field.

Ni-field

A field composed of N decimal digits is defined in the data dictionary as *'Ni'* where the N is a positive integer number that gives the number of decimal digits in the field.

Nx-field

A field composed of N hexadecimal digits is defined in the data dictionary as *'Nx'* where the N is a positive integer number that gives the number of hexadecimal digits in the field.

N..Mz-field

For some fields, the number of basic elements shall be between a minimal number and a maximal number. These cases are indicated by the notation "*N..Mz*", where "*N*" is the minimal number of basic elements, "*M*" is the maximal number, and "*z*"is one of the four basic types (i.e. a, b, i, x).

2.5 Audience

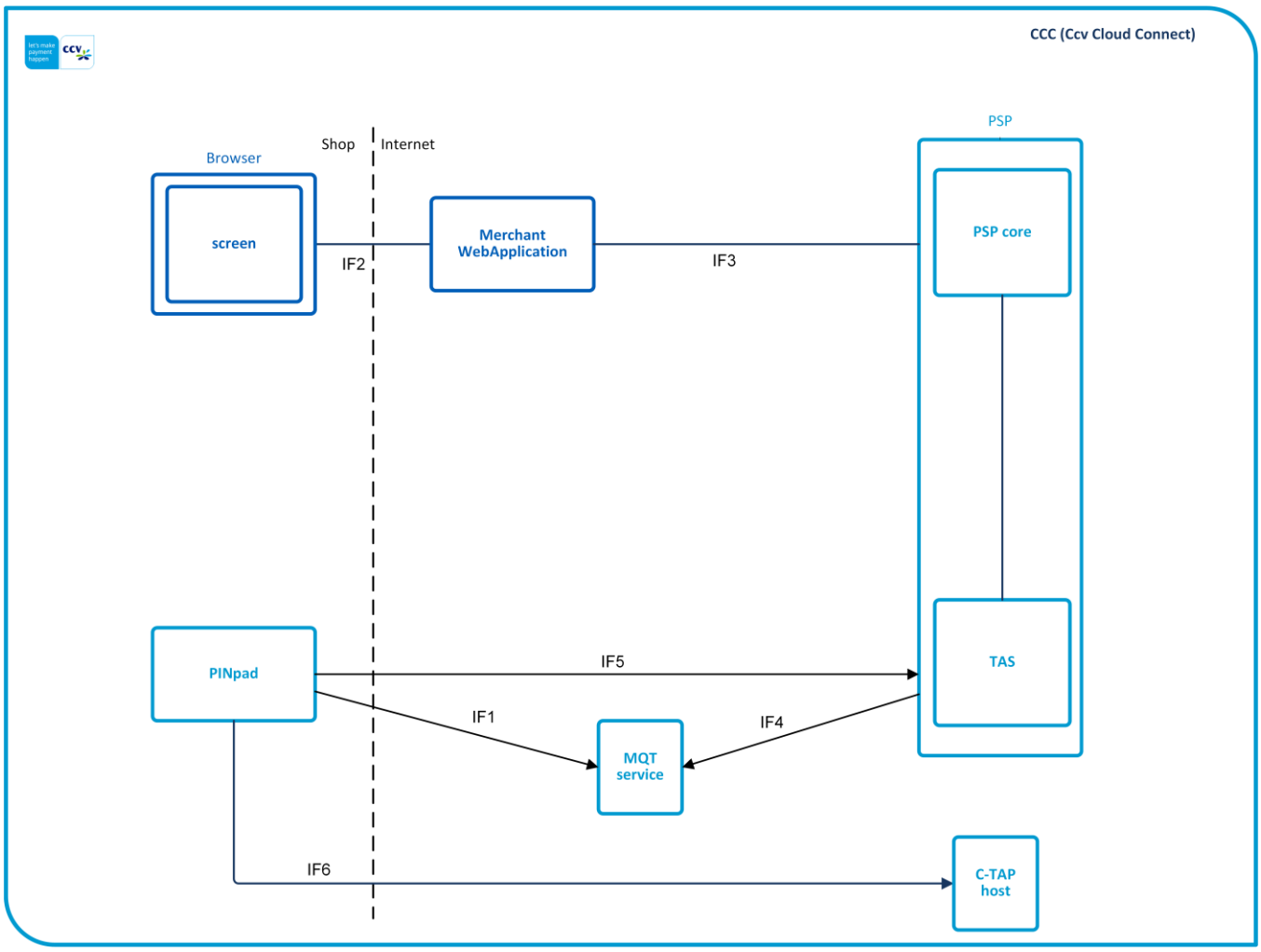
This document is intended for software engineers of web based POS suppliers.

2.6 References

[1] Testbook VPOS PSP Attended
Version: 2.1
CCV Group Competence Center Payment Acceptance
Ben van de Put
Release date: t.b.d.

3 System Overview

Below you will find a System overview of the infrastructure used in CCC:



The table below describes the various components shown in the picture above:

Component	Description
Screen	The screen of the kiosk/ECR PC running a browser
Browser	The browser shows the content generated by the Merchant Web Application and is used by the operator to buy/sell his products and/or services and to initiate the payment
Merchant Web Application	The Merchant Web Application is either a: • WebShop (semi unattended); • or a POS (attended). In both situations the Merchant Web Application must know on which device the browser runs, to be able to access the proper payment terminal

Component	Description
PSP	CCV PSP is providing a number of online payment facilities like iDEAL, PayPal etc but also the CCC facility
PSP core	PSP core is the facility accepting and processing online payments for webshops
TAS	Terminal Acceptance Server is an extension on the PSP core facilitating the CCC service with usage of the PINpad
PINpad	C-TAP Payment Terminal accepting transactions from the TAS and processing debit and credit card payments using the (convential) C-TAP protocol towards the acquirers
MQT service	Each terminal sets up a permanent connection to the MQT service (outbound traffic) to enable the PSP to call the PINpad, to connect to the PSP as soon as a transaction has been started
C-TAP Host	Host accepting and forwarding transaction towards the Acquirer for authorization

The table below describes the various interfaces shown in the picture above:

Interface	Description
IF1	MQTT with TLS1.2 with mutual authentication. As soon as a CCC PINpad is running, it will connect to the MQT service to subscribe to commands from the PSP
IF2	HTTPS, running between browser and Merchant Web Application
IF3	HTTPS with API key, running VPOS PSP API protocol described in this document. Note: **every merchant has his own API key**. An API key gives access to a merchant account and a merchant account can contain multiple terminals.
IF4	MQTT with TLS1.2 with mutual authentication. When the PSP want a terminal to connect to the PSP, it sends a command using this interface. This will be transferred to the PINpad using IF1, after which the PINpad immidiately connects to the PSP using interface IF5
IF5	TCP/IP socket connection with TLS 1.2 with mutual authentication carrying the OPI-NL protocol accepted by the PINpad to process transactions
IF6	TCP/IP socket connection running the C-TAP protocol used for authorization of the transactions

3.1 Requirements to receipts

An EFT receipt (Part C in the picture below) is generated by an EFT terminal and has the status of legal evidence and this evidence is to be owned by the cardholder. That means that equipment using EFT terminals must be able to supply the EFT receipt to the cardholder.

A typical transaction receipt consists of four parts:

	12345678901234567890123456789012345678901234567890	Responsibility
Part: A Header Lines	CCV Group B.V. Westervoortsedijk 55 6827 AT Arnhem Tel. 088-2289911	POS
Part: B Sales Lines	Article 1 EUR 2.01 Article 2 EUR 5.38 . . .	POS
Part: C EFT Lines	Kopie Kaarthouder Terminal: 20000002 Merchant: 1234 Periode: 8038 Transactie: 10052457 Contactloze betaling MAESTRO (A0000000043060) Kaart: xxxxxxxxxxxx0276xxx Kaartnr: 01 Betaling 07/02/2018 14:50 Auth. code: 123456 Totaal: 10,00 EUR Akkoord	Terminal
Part: D Footer Lines	Zondag geopend van 12.00 tot 18.00 uur	POS

During a transaction, the POS will be responsible for printing the following sections of the receipt:

- Header Lines: The contents of these header lines, for example name, address and city of the shop are generated by the POS;
- Sales lines. Containing information about the goods that the cardholder wants to purchase, for example product name, quantity, unit price, VAT. This information is only available in the POS;
- Footer lines. The contents of these footer lines are only available in the POS and may include a paper cut if the printer supports this.

The Payment Terminal is responsible for generating of the text of the EFT lines. The content of the receipt may not be altered by the POS. Even empty lines must be printed. A failing transaction does not generate a cardholder receipt nor EFT Lines.

3.1.1 Printing a receipt

Most POS devices simply print the receipt (at least part C as shown above) as a separate receipt on cardholder request. But the receipt can also be part of an invoice. Regular receipts can be send by e-mail to

the cardholder as well, but when authentication of the cardholder is not done by entering a PIN-code but by signature (and or identification), the merchant must collect (and check) the signature/identification. Therefor it is mandatory that the POS is either equipped with a:

- Printer, to print the '*merchantReceipt*' on which the signature/identification is collected;
- Or a touch pad which can be used by the merchant to collect the cardholder signature/identification on the '*merchantReceipt*'. The remainder of this document will describe printing the receipt to collect signature/identification, but collecting these items using a touch pad to store a '*merchantReceipt*' with signature/identification is also allowed.

When a receipt is optional to deliver to the cardholder the *ReadTransactionResponse* will contain: *"printCustomerReceipt": false*. When must be delivered to the cardholder the *ReadTransactionResponse* will contain: *"printCustomerReceipt": true*.

3.1.2 Supplying a receipt by e-mail

When a receipt is sent by e-mail to the cardholder the following rules must be met:

- Prior to do a transaction it shall be clear to the cardholder that the receipt is sent by e-mail;
- If the e-mail address is not yet known, the cardholder is asked for his e-mail address prior to the transaction;
- When the cardholder does not want to share his e-mail address and still wants a receipt, the receipt is either printed or the process of purchase products or services is cancelled;
- After the transaction the cardholder receipt is sent to the e-mail address of the cardholder.

3.2 Requirements to journal

Journal is information intended for the *merchant* and contains information of every single transaction. Journal is generated for both successful and failing transaction but only generated when the card is supported by the payment terminal. Journal is supplied as text based (*journalReceipt*) or XML based (*eJournal*).

This information (both *journalReceipt* and *eJournal*) is provided in the *ReadTransactionResponse*. It is *optional* to store because it is also available to merchant in the *PSP backoffice* where it is stored permanently in administration belonging to the API key. But many merchants want to have this information to be stored in their own systems, for example to be able to create totals based on card brands and/or booking period totals. In case of conflicts this information can be used towards an acquirer. Note: the information stored in the *PSP backoffice* is leading above the information stored in merchant systems.

4 Flows

This section explains a number of possible flows. These are:

- Regular payment flow ending in:
  - a successful transaction;
  - a declined transaction;
- Transactions running nearly or completely into timeout.

4.1 Regular payment flow

The VPOS PSP API only consists of six messages:

- CreateNewTransactionRequest;
- CreateNewTransactionResponse;
- Webhook;
- ReadTransactionRequest;
- ReadTransactionResponse.

Before showing flows it is important to have a general idea what the function of these messages are. This is explained in the next sections.

4.1.1 CreateNewTransactionRequest

The *CreateNewTransactionRequest* is sent from *Merchant WebApplication* to the *PSP* and is used to initiate a new transaction. The most important items in this message are:

- amount: containing the amount to be paid;
- operatingEnvironment: defines if the terminal works in Attended or Semi Unattended mode;
- merchantLanguage: defines the language shown on the intermediate page (Attended only);
- managementSystemId: as an identifier in which environment the Payment Terminal is managed (TMS NL, TMS BE, TMS CH, TMS DE for example);
- terminalId: as an identifier which terminal is used to process the transaction. The content of this item is the PINpad *TMS Terminal ID*.

After the PSP has received this message, the terminal has a time frame of 10 seconds to connect to the PSP. If the connection is not established in this time, the transaction will fail.

4.1.2 CreateNewTransactionResponse

The *CreateNewTransactionResponse* is sent from *PSP* to the *Merchant WebApplication* and is used to indicate if, according the *TAS administration*, the *Payment Terminal* is *free for use*.

On *success* the *TAS administration* will mark the *terminal a*s being *in use*. The *TAS* will issue a *SessionNumber* and the *browser* of the kiosk/ECR is ordered to redirect to a *PSP page* . In Semi Unattended environment the page shows "*Follow the instructions on the PINpad*". In Attended environment cashier display messages and abort button are shown. From this page the *IPaddress* and *port* are used to send an HTTP request to the terminal containing the *SessionNumber*. This request is used by the terminal to trigger a TCP/IP socket connection using TLS 1.2 (with mutual authentication between *PINpad* and *TAS*).

Now the *TAS* will receive the *SessionNumber* and *terminalID* from the terminal. The *TAS* notices a match of these items from both its *own administration* and *terminal* and starts a transaction using the *amount*. After this the amount is displayed on the terminal screen and the cardholder is invited to present his debit or credit card.

Now the terminal processes the payment using the *C-TAP* protocol.

When the *CreateNewTransactionResponse* returns something different then '*success*', '*pending*' or '*manualIntervention*' , the transaction shall be considered as unsuccessful and the *Merchant WebApplication* may start the transaction again.

4.1.3 Webhook

Once the *Merchant WebApplication* receives a *CreateNewTransactionResponse,* it has to wait for a *Webhook* sent by the PSP indicating that there is a *change in the status* of the running transaction.

In general a transaction will take about 5 to 30 seconds. Contactless transactions, where the cardholder only taps his card, mostly take about 5 to 10 seconds. Chip based transactions take more time because the cardholder has to enter his card into the reader, enter his PINcode and remove the card before the transaction on the PINpad is ended.

The worst case scenario where a transaction can still be successful takes up to 6 minutes. This is when the cardholder enters his card on the very last second, enters two times wrong PIN on the very last moment, enters a third and correct PIN on the last moment and when hosts communication ends up in retries, etc.

The Webhook is not guaranteed to be delivered to the web application, but will be repeated by the PSP in case no proper answer is received.

4.1.4 ReadTransactionRequest

When the status of the transaction is reported to be changed (by the Webhook) the *Merchant WebApplication* must sent a *ReadTransactionRequest* to get the result of the transaction. The PSP will respond with the *ReadTransactionResponse*.

The *Merchant WebApplication* can also use the *ReadTransactionRequest* message to poll the actual status of the running transaction. When the *ReadTransactionResponse* shows a *TransactionResult = Pending*, the *WebApplication* has to wait at least for 30 seconds for the *Webhook*. When not received the *WebApplication* can send another *ReadTransactionRequest* to poll the status again.

4.1.5 ReadTransactionResponse

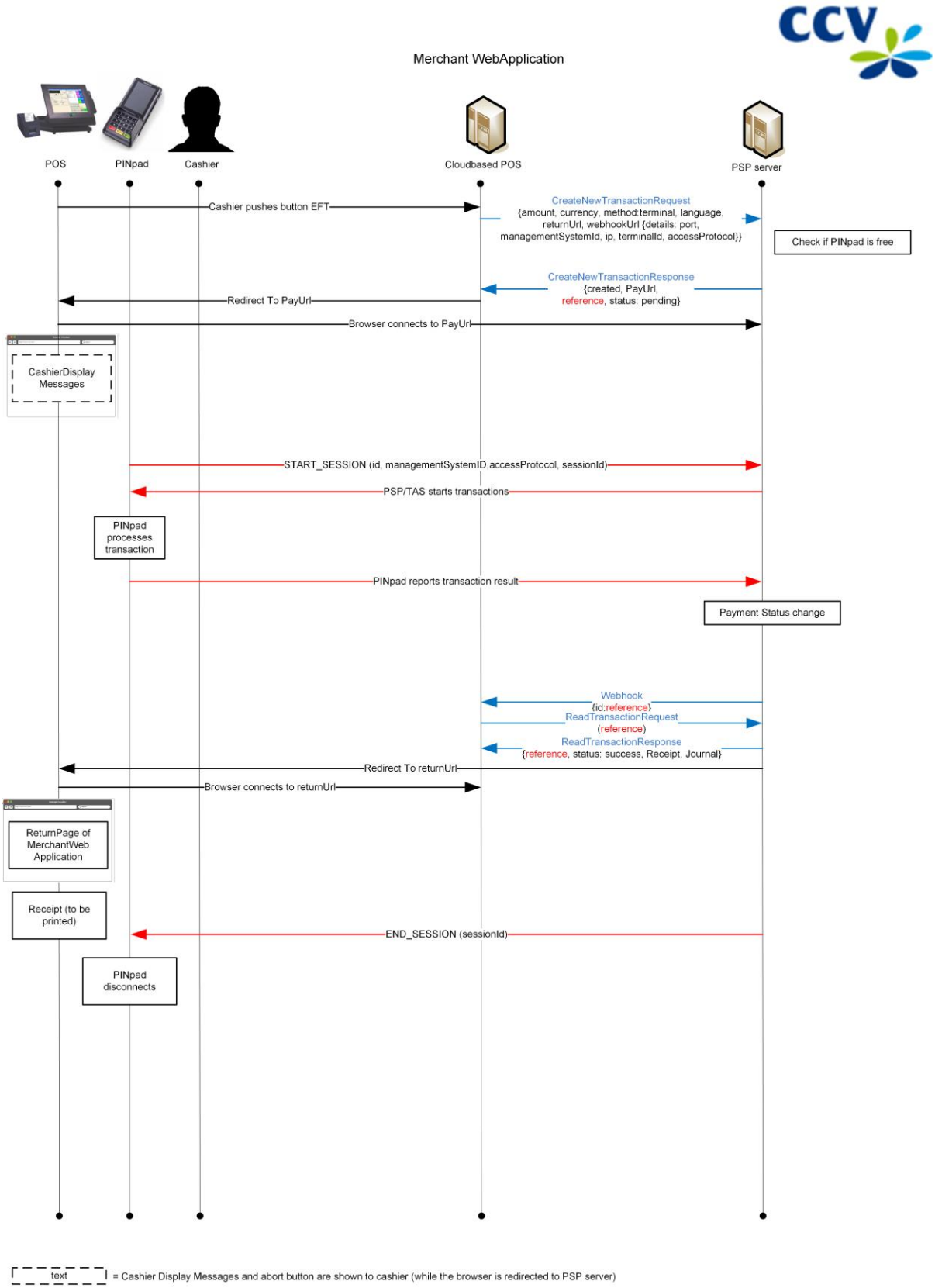
The *ReadTransactionResponse* contains a number of important fields:

- *status*: showing the result of the transaction. "*success*" is final and means that the transaction is successful, "*pending*" and "*manualintervention*" means the payment is still busy and "*failed*" is final and means that the transaction is unsuccessful;
- *customerReceipt*: Receipt to be delivered to the cardholder (only on cardholder request or always);
- *Journal both*:
  - Text formatted (*journalReceipt*);
  - XML formatted (*eJournal*);
- *printCustomerReceipt* (indicator if *customerReceipt* has to be printed according the *payment terminal*);
- *merchantReceipt* (only applicable for attended solutions);
- *askCustomerSignature* (only applicable for attended solutions);
- *askCustomerIndentification* (only applicable for attended solutions);
- *askMerchantSignature* (only applicable for attended solutions in case of refund transactions).

4.2 Flow

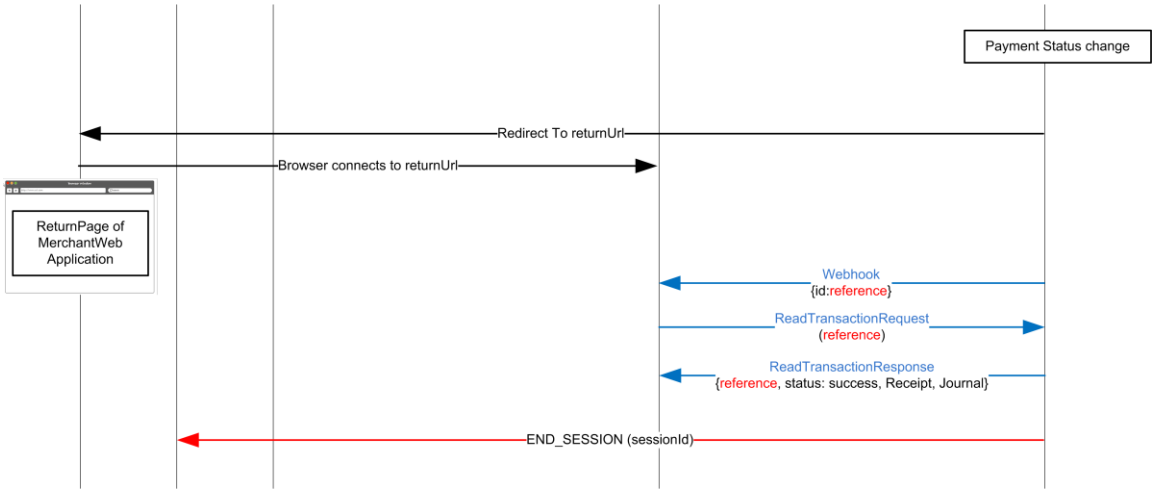
The picture below shows the happy flow as explained in the previous sections. (hier verder)

Regular flow ATTENDED



The process sending the *Webhook* and the browser redirecting to the *returnUrl* is an *asynchronous* process meaning the situation shown below can occur as well.

Note that in this situation (see picture below) the result of the transaction is for the *Merchant WebApplication* still unknown at the moment the *returnUrl* is received, because the *Webhook* and *ReadTransactionRequest* are not yet exchanged.



This means that the return page (= *returnUrl*) may not show any information about the result of the transaction, but it shall display something like "*One moment please*". As soon as the result of the transaction is known ('*status*' = "*success*" or '*status*' = "*failed*") a new page can show the final result.

4.3 Transaction running (nearly) into timeout

During a transaction a lot of failure scenarios can happen. This may vary between a clear failure or even in a situation where the result of the transaction is unknown.

When a transaction fails, it is up to the Operator to either retry the transaction by pressing the payment button again or to cancel the entire purchase.

A less convenient situation is where the transaction result is unknown. The next paragraphs show how to recover this situation.

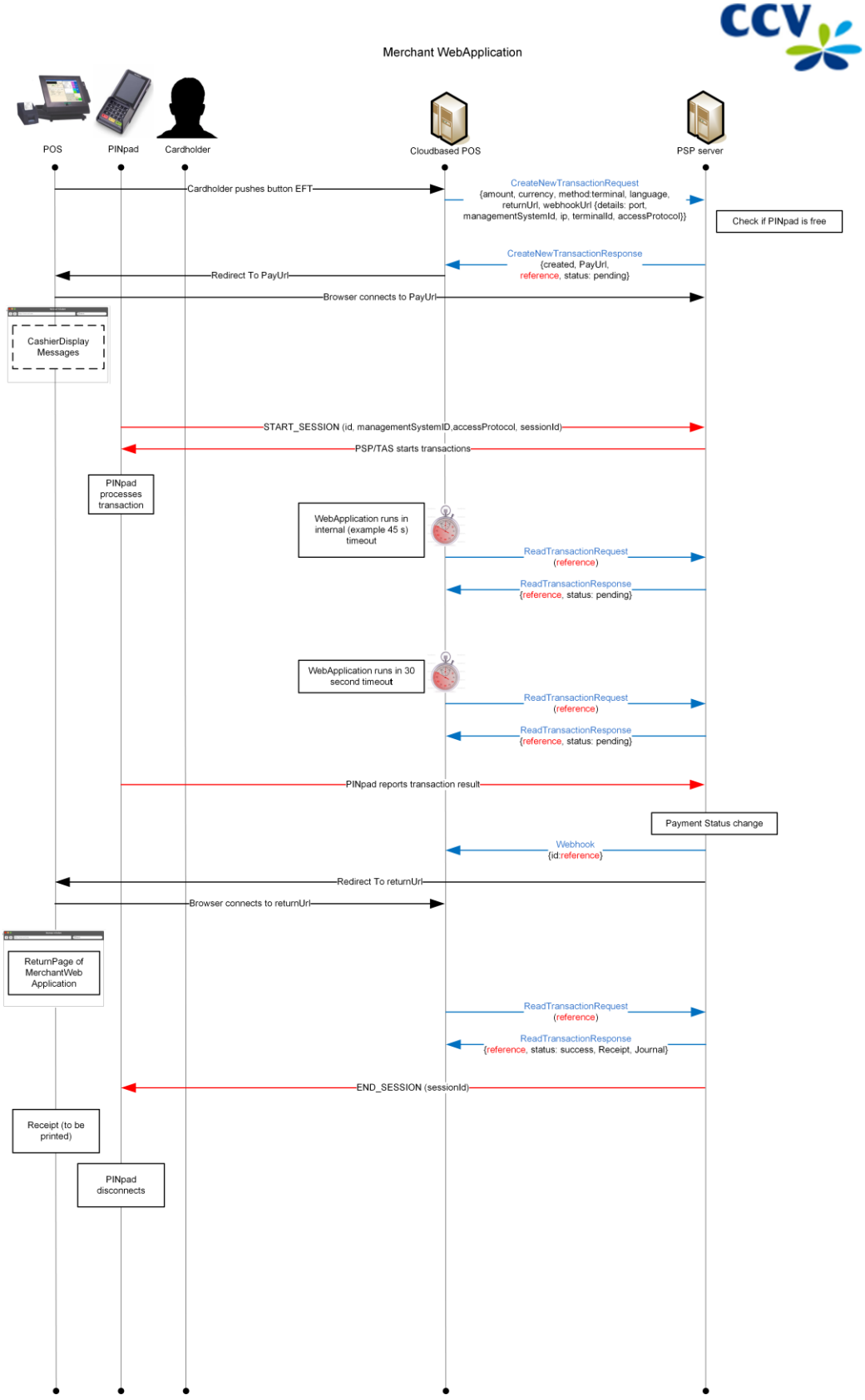
4.3.1 Late transaction response (< 6 minutes)

When a Webhook is not received within for example for example 45 seconds, the Merchant WebApplication can send a ReadTransactionRequest.

When response shows status=pending the Merchant WebApplication has to wait for another 30 seconds and can send ReadTransactionRequest to poll the status of the transaction again. Normally the PSP will send a Webhook within 6 minutes on which the Merchant *WebApplication* sends the *ReadTransactionRequest* again. The *PSP* will answer with a *ReadTransactionResponse* showing the *Status* containing either *success* or *failed*.

In a flow diagram it looks like:

Late result flow ATTENDED



text = Cashier Display Messages and abort button are shown to cashier (while the browser is redirected to PSP server)

4.3.2 Time out and recovery after six minutes

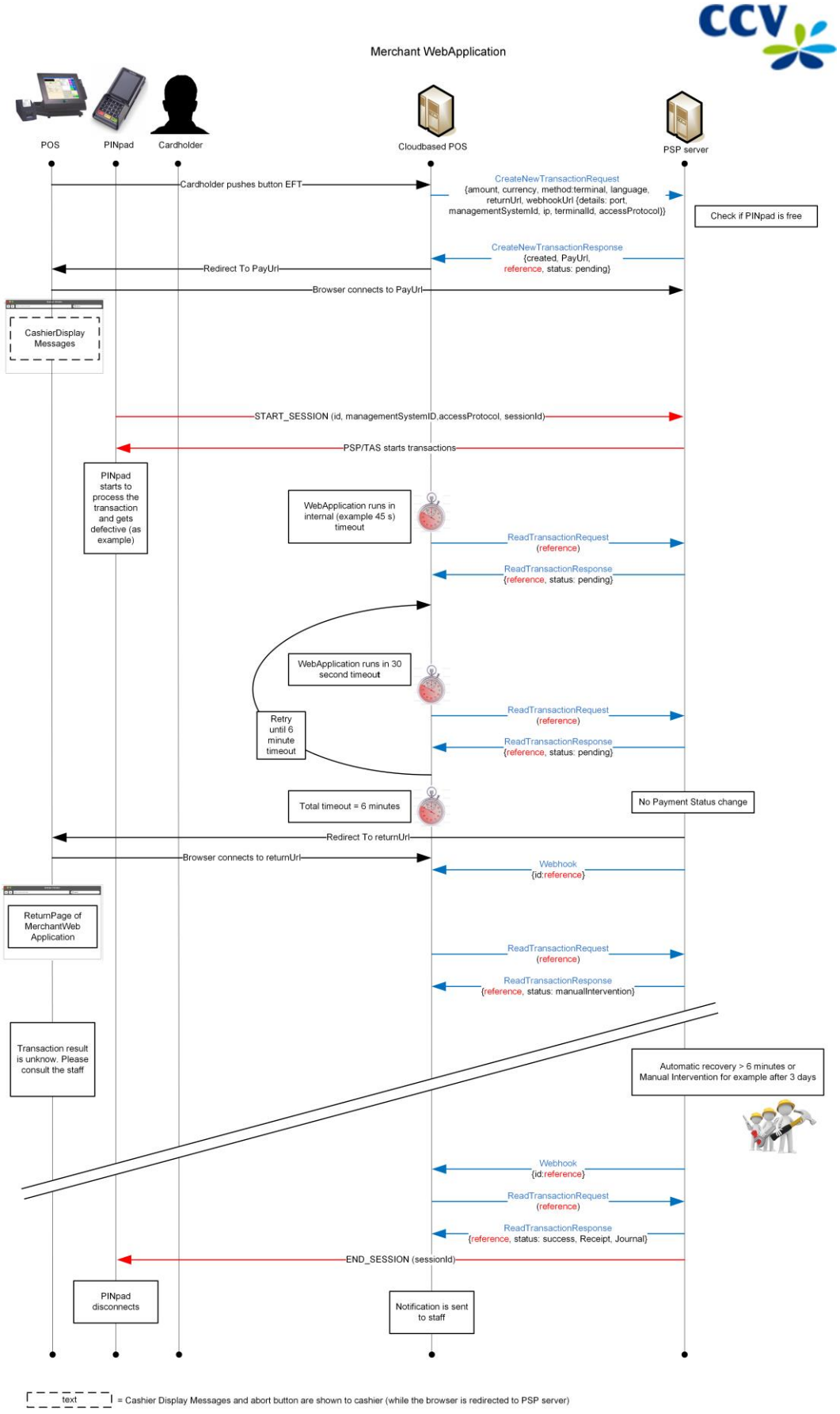
A transaction is timed out 6 minutes after receiving the *CreateNewTransactionResponse*. The browser will switch to the *returnUrl* and the *PSP* will send a *Webhook* and *ReadTransactionRequest* will still result in a response showing *status=manualintervention*. For the *Merchant WebApplication* the result of the transaction is unknown. The advice is to show a page on the kiosk/ECR telling that the *result of the transaction is UNKNOWN* and to *contact the staff of the Shop*. The page may not show any text like: "*Error*", "*Failure*" etcetera, to avoid the staff concluding that the transaction failed.

The cause of this situation is *failing* network or equipment. Once everything is up and running again, the situation will be *recovered automatically* and will result in a second *Webhook* sent by the *PSP* towards the *Merchant WebApplication* on which te a *ReadTransactionRequest* can be sent on which the WebApplication receives a response which must be used to update the *status* of the transaction.

When automatic recovery is not possible, for example due to a defective payment terminal, *staff of the PSP* will get the result of the transaction by phone, and will do a *manual intervention,* again resulting in the *Webhook*. Theoretically this can take several days.

In a flow diagram the situation described above looks like:

Result after 6 minutes flow ATTENDED



5 Commands in detail

The API is based on common HTTP API implementations. This document only shows details for the Cloud Accessible Terminal. Details for other means of on-line payments, please look at the general PSP documentation available on https://api.psp.ccv.eu/api/v1/doc.

5.1 General

For production environment n*on-secure connections* are not allowed. Using http:// instead of https:// results in a redirect:

Redirect Response:

```
Status Code: 302

Content Type: text/html; charset=iso-8859-1

Location: The secure version for the requested URL
```

Note: *non-secure connections* are only allowed in *test environment*.

Each request must define the header *User-Agent*. If not provided, the request will be blocked and result in a non-API error:

Forbidden Response:

```
Status Code: 403

Content Type: text/html; charset=iso-8859-1
```

5.2 Authentication

As a merchant you'll access the PSP as a remote service without interaction of the customer. The access is restricted and requires *authentication*. To authenticate the Merchant WebApplication an API key is provided (per merchant).

Authentication is performed using *Basic Authentication* which requires you to send an *Authorization header for each request*. The format of the value of the header is defined as *username:password*. The *username* is the *value of the API key* and the *password* remains *empty*.

An example:

```
l_e40a9b36f4c1fa31f512c623bb5a315696e98cea:
```

Note: The header string must be *Base64 encoded*.

5.2.1 Idempotency

Using idempotent requests is best used if there is no way to avoid duplicate requests. Time-outs are often the cause of retransmission and must be possible without side-effects like processing the request twice.

Idempotency can be enabled by adding the optional request header Idempotency-Reference, which should contain a non-empty unique value of maximum 50 characters generated for each transaction. GET requests are idempotent by definition. The minimum length is 6 characters but we recommend a UUID generated for each new transaction and not for example a database id.

We treat a request as identical to another for idempotency when the exact same request header is used and the following fields all match (also including empty or null values) for the selected operation.

Steps to create the user agent header for a Payment request including Idempotency-Reference:

1. Take the API key: **l_fullapikey**
2. Add ":": **l_fullapikey:**
3. Perform Base64 encoding step: **bF9mdWxsYXBpa2V5Og==**
4. Add "Basic " as prefix (inclusing space): **Basic bF9mdWxsYXBpa2V5Og==**
5. Create an "Idempotency-Reference" (example: 123e4567-e89b-12d3-a456-426655440000)
6. CURL HTTP example without body looks like:

```
curl -X POST 'https://vpos-test.jforce.be/vpos/api/v1/payment' \
    -H 'Content-Type: application/json' \
    -H 'Authorization: Basic bF9mdWxsYXBpa2V5Og==' 
    -H 'Idempotency-Reference: 123e4567-e89b-12d3-a456-426655440000'
```

If the PSP does not respond within 3500 seconds, the same message must be resend.

As a result the PSP will, in case

1. The response was lost, repeat the response
2. The request was lost, create a new transaction and return the response

As a result double transactions are not possible.

5.3 Operating Mode

When a key is generated, you can choose the operating mode of the key. The mode defines what will happen with the payment when it is processed by the system.

There are 2 operating modes: *LIVE & TEST*. To identify the operating mode of a key, use the prefix: "l_" for LIVE and "t_"for TEST.

In production environment only LIVE keys are applicable. For test environment you can use either:

- Test keys: then the payment terminal is emulated and a transaction will always succeed;
- Live keys: then a real payment terminal (with C-TAP test keys and test cards) is used. Depending on the amount, special behavior, like declining a transaction, can be invoked.

5.4 Error Handling

In case of an error the *PSP* responds with a corresponding HTTP status code. The body of the response will be an error in JSON.

Response:

```
Status Code: Dependent on the type of error: see HTTP Status for possible values

Content Type: application/json;charset=UTF-8

Content: Error
```

HTTP Status:

Code	Message	Description
400	Bad Request	Invalid input in the request
401	Unauthorized	None or invalid API key (typo, expired)
404	Not found	invalid URL
415	Unsupported Media Type	Not using application/json as content type

Code	Message	Description
500	Internal Server Error	Unexpected error
504	Gateway Timeout	Processing the request took too long. Default time out is 60000 ms

Error

All resources will return a JSON object in case of an error.

Definition

Name	Description
type	Possible values: input_error, process_error, unexpected_error
message	A generic message
field	In case of type input_error the field that caused the error. Else not present
fields	In case of type input_error an array of all failed field validations. For each field the error message is shown. See example below
reference	A reference that you can provide to the helpdesk for investigation of the failing request
failureCode	An optional field that explains more about the cause of the failure. Possible values: see Failure Codes
transactionReference	IN case of a time out, an optional transaction reference to query the status of the transaction

Example Input error

```
{

  "type" : "input_error",

  "message" : "An input parameter is incorrect",

  "reference" : "718ee617",

  "field" : "amount"

}
```

Example of multiple input errors

```
{

  "type" : "input_error",
  "message" : "An amount is required",
  "reference" : "718ee617",
  "field" : "amount",

  " fields": [

    {
```

```
      "field" : "amount",
      "message" : "An amount is required"
    }
    {
      "field" : "method",
      "message" : "A supported method is required"
    }
    {
      "field" : "method",
      "message" : "may not be null"
    }
    {
      "field" : "currency",
      "message" : "A ISO 4217 currency code is required"
    }
  ]
}
```


Failure Codes

Name	Description
processing_error	A general error if the transaction can't be processed correctly
invalid_config	The configuration of the merchant or method is not correct
cancelled	RFU (Reserved for Future Use): The transaction is cancelled by either the customer or the merchant. The field 'cancelledBy' indicates the initiator of the cancel
rejected	The transaction is rejected
unknown_reference	The reference used is not known
unsupported_currency	The 'currency ' in your request is not supported
bad_credentials	The provided authentication credentials in your request are incorrect

5.5 Resources

- The API uses HTTP as the communication protocol;
- Each request body must be encoded with UTF-8;
- Each response body is encoded with UTF-8;
- Each datetime value is in the so-called epoch timestamp; number of milliseconds since January 1st, 1970, UTC;
- Each ISO 8601 Duration is interpreted relative to the epoch time; number of milliseconds since January 1st, 1970, UTC.

5.6 Message Content

As shown before the flow consists of six messages:

- CreateNewTransactionRequest;
- CreateNewTransactionResponse;
- Webhook;
- ReadTransactionRequest;
- ReadTransactionResponse.

The next section describe the content of the various messages.

5.6.1 CreateNewTransactionRequest

With a *CreateNewTransactionRequest* you can start a:

- Sale transaction (payment);
- Refund transaction.

5.6.1.1 Sale transactions (Payment)

Below you will find an example of the *CreateNewTransactionRequest* to start a sale transaction:

```
HTTP POST https://psp.base.url/api/v1/payment

{

        "currency": "EUR",

        "amount": "0.10",

        "method": "terminal",

        "language": "nld",

        "returnUrl": "https://shop.base.url/sub-url",

        "webhookUrl": "https://shop.base.url/api/webhook",

        "details": {

                "operatingEnvironment":"ATTENDED"

                "merchantLanguage":"NLD"

                "managementSystemId": "GrundmasterNL-ThirdPartyTest",

                "terminalId": "J4S009",

                "accessProtocol": "OPI_NL"

        }

}
```

Note: comparing a refund to a payment the fields "currency", "method" and "language" are omitted because the PSP will take over this information from the original payment

The URL `https://psp.base.url/api/v1/payment` in the HTTP Post command indicates that the function starts a payment. A payment is defined as a sale transaction.

5.6.1.2 Refund transactions

For Refund transactions a number of extra requirements must be met:

- To support Refund transactions the merchant must contact his acquirer to get a contract for Refund transactions. On this contract the acquirer will insert some parameters into the terminal to enable Refund transactions. Without this contract the terminal will decline refund transactions;

- Refund transaction may only be started by authorized persons. That implies that the POS has to be equipped with a structure of rights, to avoid 'normal cashiers' to start a refund transaction;
- In the refund request the tag "reference" must contain the "reference" of the original payment transaction.
- During the refund transaction the PSP will generate:
  - a cardholder receipt with signature line. This receipt must be signed by the merchant and should be given to the cardholder;
  - a merchant receipt with signature line. It is up to the merchant policy what to do with this receipt. (For example: to be signed by cardholder or even throw it away);
  - a tag "askMerchantSignature". See askMerchantSignature (on page 38) what to do with this tag;
- The money will be transferred to the cardholder account next businessday (weekdays). To avoid misunderstandings with the cardholder, a footer, added by the POS, with the text "Het geld staat de volgende werkdag op uw rekening" (translated: the money will be in your account the next business day) is advised.

Below you will find an example of the *CreateNewTransactionRequest* to start a refund transaction:

```
HTTP POST https://psp.base.url/api/v1/refund

{

        "amount": "0.10",

        "reference" : "TL220523095203524CB87E3A2.0",

        "returnUrl": "https://shop.base.url/sub-url",

        "webhookUrl": "https://shop.base.url/api/webhook",

        "details": {

                "operatingEnvironment":"ATTENDED"

                "merchantLanguage":"NLD"

                "managementSystemId": "GrundmasterNL-ThirdPartyTest",

                "terminalId": "J4S009",

                "accessProtocol": "OPI_NL"

        }

}
```

Comparing this request to a payment, the fields "currency", "message" and "language" are omitted. The PSP takes these fields over of the original transaction.

The URL `https://psp.base.url/api/v1/refund` in the HTTP Post command indicates that the function starts a refund.

5.6.1.3 currency

Properties:
enumeration:
"*eur*" for Euro
"*gpb*" for British Pound
....

Occurence:
Mandatory

Description:
The code used for "*currency*" is according ISO 4217 and contains the domestic currency of the country where the POS is located.

5.6.1.4 amount

Properties:
1..9n
Note: a '.' is used as decimal separator and a thousands separator is not used. Example: 123456.12.

Occurence:
Mandatory

Description:
The parameter "*amount*" contains the amount to be paid.

5.6.1.5 method

Properties:
1..25a with enumeration
"*terminal*"

Occurence:
Mandatory

Description:
Although the *PSP* supports more *methods*, it is mandatory to set the "*method*" to the value "*terminal*" to enable the payments using the payment terminal.

5.6.1.6 reference

Properties:
5..50a
Example: TL180202161938854CB87E102.0

Occurence:
Optional, only with refund transactions.

Description:
Contains the value of "*reference*" of the *original transaction* to be refunded.

5.6.1.7 language

Properties:
3a with enumeration:
"*nld*" for Dutch (Nederlands)
"*eng*" for English
"*deu*" for German (Deutsch)
"*fra*" for French (Francais)

Occurence:
Mandatory

Description:
When a *Merchant WebApplication* is multi language, the tag "*language*" can be used to transfer the selected (cardholder) language on the website to the terminal.

5.6.1.8 returnUrl

Properties:
url

Occurence:
Mandatory

Description:
The URL to redirect the browser to after completion of the payment (both success, failure and timeout scenarios)

5.6.1.9 webhookUrl

Properties:
url

Occurence:
Mandatory

Description:
This information is used by the *PSP* and contains the url of the *WebApplication* where to send the *Webhook*.

5.6.1.10 details

The element "*details*" contains extra information which is only applicable to for the "*method*"="*terminal*".

This information consists of the following items:

- operatingEnvironment;
- merchantLanguage (attended only)
- managementSystemId;
- terminalId;
- accessProtocol.

The next section explain more about these items.

5.6.1.10.1 operatingEnvironment

Properties:
1..20a with enumeration:
"*ATTENDED*" for Attended use (i.e. where a cashier starts a transaction)
"*SEMI_UNATTENDED*" for Semi Unattended use (i.e. where a cardholder or automatic process starts a transaction)

Occurence:
Mandatory

Description:
When '*operatingEnvironment*' is set to "*ATTENDED*" the redirected page will show Cashier Display messages and an Abort button intended to be used by a cashier.
When '*operatingEnvironment*' is set to "*SEMI_UNATTENDED*" the redirected page will show "Follow the instruction on the PINpad" intended to be used by a cardholder.

5.6.1.10.2 merchantLanguage

Properties:
3a with enumeration:
"*ENG*" for English
"*NLD*" for Dutch (Nederlands)
"*FRA*" for French
"*DEU*" for German

Occurence:
Mandatory when '*operatingEnvironment*' is set to "*ATTENDED*"
Omitted when '*operatingEnvironment*' is set to "*SEMI_UNATTENDED*"

Description:
Defines the language shown to the cashier.

5.6.1.10.3 managementSystemId

Properties:
35a with enumeration:
"*GrundmasterNL*" for payment terminals managed on TMS in the Netherlands;
"*GrundmasterBE*" for payment terminals managed on TMS in Belgium;
"*GrundmasterNL-ThirdPartyTest*" for payment TEST terminal manage on TEST TMS in the Netherlands.

Occurence:
Mandatory

Description:
A "*terminalId*" is unique per TMS but since there are more TMS Hosts active, a specific "*terminalId*" could be active on more TMS hosts. This means that a specific "*terminalId*" does not identify a terminal uniquely. To achieve that, the *Merchant WebApplication* must send the identifier of the applicable TMS in the "*managementSystemId*" to uniquely identify the terminal.

5.6.1.10.4 terminalId

Properties:
10a

Occurence:
Mandatory

Description:
To identify the terminal on the kiosk/ECR, the *Merchant WebApplication* must send the "*terminalId*" of the terminal. In the terminal this ID is known as the *TMS Terminal ID* or *TMS TID*.

5.6.1.10.5 accessProtocol

Properties:
3..12a with enumeration
"*OPI_NL*" to connect the ITS Vx820 / ITS P400 terminal

Occurence:
Mandatory

Description:
This field is used to identify the protocol between the PSP and the terminal. In the future more values will be issued as soon as other terminals with other protocols are used in this solution.

5.6.2 CreateNewTransactionResponse

Below you will find an example of the *CreateNewTransactionResponse*:

```
HTTP 200 OK

{
        "status": "pending",
        "type": "sale",
        "currency": "eur",
        "amount": 0.1,
        "reference": "TL1710E06B.0",
        "method": "terminal",
        "payUrl": "https://psp.base.url/terminal/index.html?reference=TL1710E06B.0",
        "returnUrl": "https://shop.base.url/sub-url",
        "created": 1508155139021,
        "lastUpdate": 1508155139021,
        "language": "nld",
        "details": {
                "operatingEnvironment":"ATTENDED"
                "merchantLanguage":"NLD"
                "managementSystemId": "GrundmasterNL-ThirdPartyTest",
                "terminalId": "J4S009",
                "accessProtocol": "OPI_NL"
        }
}
```

The message header contains the HTTP status code.

5.6.2.1 status

Properties:
4..20a with enumeration
"*pending*" indicating: The transaction is created but *not completed yet*;
"*failed*" indicating: Transaction *processing failed;*
"*manualintervention*" indicating: The transaction is in a *state which can't be recovered automatically*. Contact the *PSP help desk* for further investigation. The transaction is *not completed yet*;
"*success*" indicating: The transaction is *processed successfully*.

Occurence:
Mandatory

Description:
This field indicates the *status* of the *running transaction*. When a transaction is started and the PINpad is available the "*status*" will contain the value "*pending*" indicating that the transaction is running (but the result is still pending). When the status *changes* to one of the other values, the *PSP* will send a *Webhook*. After that the *Merchant WebApplication* must send a *ReadTransactionRequest* and the response will show the *actual status of the transaction*.

5.6.2.2 type

Properties:
4..20a with enumeration
"*sale*" indicating a sale transaction
"*refund*" indicating a refund transaction

Occurence:
Conditional

Description:
This field contains information about the transaction type of the transaction. For now only sale and refund transactions are allowed. In the future there will be more enumerations create to indicate transaction types available for attended environments.

5.6.2.3 currency

Echo of the "*currency*" present in the *CreateNewTransactionRequest*.

See: currency (on page 27) for more details.

5.6.2.4 amount

Echo of the "*amount*" present in the *CreateNewTransactionRequest*.

See: amount (on page 28) for more details.

5.6.2.5 reference

Properties:
5..50a
Example: TL180202161938854CB87E102.0

Occurence:
Conditional

Description:
A *unique reference* to the transaction issued by the *PSP*.

5.6.2.6 originalReference

Properties:
5..50a
Example: TL180202161938854CB87E102.0

Occurence:
Conditional, only at refund transaction

Description:
Contains the reference of the original transaction refunded.

5.6.2.7 method

Echo of the "*method*" present in the *CreateNewTransactionRequest*.

See: method (on page 28) for more details.

5.6.2.8 payUrl

Properties:
url

Occurence:
Conditional

Description:
The "*payUrl*" is issued by the *PSP* and must be used by the browser to call/redirect to the PSP intermediate page.

5.6.2.9 returnUrl

Echo of the "*returnUrl*" present in the *CreateNewTransactionRequest*.

See: returnUrl (on page 29) for more details.

5.6.2.10 created

Properties:
epoch timestamp

Occurence:
Conditional

Description:
Timestamp issued by PSP server when transaction was created. Epoch timestamp is number of milliseconds since January 1st, 1970 UTC.

5.6.2.11 lastUpdate

Properties:
Epoch timestamp

Occurence:
Conditional

Description:
Timestamp issued by PSP server when transaction was updated. Epoch timestamp is number of milliseconds since January 1st, 1970 UTC.

5.6.2.12 language

Echo of the "*language*" present in the *CreateNewTransactionRequest*.

See: language (on page 28) for more details.

5.6.2.13 details

Echo of the "*details*" including sub fields present in the *CreateNewTransactionRequest*.

See: details (on page 29) for more details.

5.6.3 Webhook

Below you will find an example of the *Webhook*:

```
HTTP POST https://shop.base.url/api/webhook

{"id": "TL1710E06B.0"}
```

The *Webhook* is called by the *PSP* as soon as the *status* of the transaction changes. The *Webhook* is sent to the *url* defined in the field "*webhookUrl*" in the *CreateNewTransactionRequest*.

5.6.3.1 id

Properties:
5..50a
Example: TL180202161938854CB87E102.0

Occurence:
Mandatory

Description:
The "*id*" contains the *reference* to the transaction also indicated in the field "*reference*" in the *CreateNewTransactionResponse*.

5.6.4 ReadTransactionRequest

Below you will find an example of a *ReadTransactionRequest*:

```
HTTP GET https://psp.base.url/api/v1/transaction?reference=TL1710E06B.0
```

Note: the *reference* above contains the value of the field '*reference*' issued in the *CreateNewTransactionResponse* or as the field '*id*' in the *Webhook*.

5.6.5 ReadTransactionResponse

Below you will find an example of a *ReadTransactionResponse*:

```
HTTP 200 OK

{

        "status": "success",

        "type": "sale",

        "currency": "eur",

        "amount": 0.1,

        "reference": "TL1710E06F.0",

        "language": "nld",

        "method": "terminal",

        "payUrl": "https://psp.base.url/terminal/index.html?reference=TL1710E06F.0",

        "returnUrl": "https://shop.base.url/sub-url",

        "created": 1508156876895,

        "lastUpdate": 1508156898055,
```

```
        "details": {
                "printCustomerReceipt": true,
                "askCustomerIdentification": false,
                "eJournal": "<E-Journal>...</E-Journal>",
                "journalReceipt": "[\"--Journal receipt--\", \"\", \"…\"]",
                "askCustomerSignature": false,
                "operatingEnvironment":"ATTENDED"
                "merchantLanguage":"NLD"
                "managementSystemId": "GrundmasterNL-ThirdPartyTest",
                "customerReceipt": "[\"--Customer receipt--\", \"\", \"…\"]",
                "terminalId": "J4S009",
                "askMerchantSignature": false,
                "accessProtocol": "OPI_NL"
        }
}
```


The next sections explain the individual fields.

5.6.5.1 status

"*Status*" shows the transaction result. See status (on page 31) for more details

5.6.5.2 type

Echo of the "*type*" present in the *CreateNewTransactionResponse*.

See: type (on page 32) for more details.

5.6.5.3 currency

Echo of the "*currency*" present in the *CreateNewTransactionRequest*.

See: currency (on page 27) for more details.

5.6.5.4 amount

Echo of the "*amount*" present in the *CreateNewTransactionRequest*.

See: amount (on page 28) for more details.

5.6.5.5 reference

Echo of the "*reference*" present in the *CreateNewTransactionResponse*.

See: reference (on page 32) for more details.

5.6.5.6 language

Echo of the "*language*" present in the *CreateNewTransactionResponse*.

See: language (on page 33) for more details.

5.6.5.7 method

Echo of the "*method*" present in the *CreateNewTransactionRequest*.

See: method (on page 28) for more details.

5.6.5.8 payUrl

Echo of the "*payUrl*" present in the *CreateNewTransactionResponse*.

See: payUrl (on page 33) for more details.

5.6.5.9 returnUrl

Echo of the "*returnUrl*" present in the *CreateNewTransactionRequest*.

See: returnUrl (on page 29) for more details.

5.6.5.10 created

Echo of the "*created*" present in the *CreateNewTransactionResponse*.

See: created (on page 33) for more details.

5.6.5.11 lastUpdate

The field "*lastUpdate*" shows the timestamp of the last update of the transaction.

See: lastUpdate (on page 33) for more details.

5.6.5.12 details

The object "*details*" contains a number of strings related to terminal payments each having his specific function. The next sections explain every individual string.

5.6.5.12.1 customerReceipt

Properties:
0..65535a

Occurence:
Conditional. A *customerReceipt* is only present in case when the payment terminal returns one.

Description:
The string "*customerReceipt*" contains the EFT receipt to be delivered to the cardholder. The receipt is an escaped JSON string. The content can be parsed to a JSON array. The content of a receipt is pre-formatted by the payment terminal and may not be changed. Even empty lines must be printed. Using a monospace font (like Courier New) the receipt is properly aligned.

5.6.5.12.2 merchantReceipt

Properties:
0..65535a

Occurence:
Conditional. A *'merchantReceipt'* is only present when a transaction is done with a signature (and/or identification) based card.

Description:
The string "*merchantReceipt*" contains the EFT receipt to be printed and is used to collect and check the cardholder signature (and/or identification). The receipt is an escaped JSON string. The content can be parsed to a JSON array. The content of a receipt is pre-formatted by the payment terminal and may not be changed. Even empty lines must be printed. Using a monospace font (like Courier New) the receipt is properly aligned.

The *merchantReceipt* must be stored by the merchant for legal reasons. When the cardholder does a charge back on the transaction, the credit card company can ask for the the signed *merchantReceipt*. In general it means: no signed *merchantReceipt* means that the charge back will be successful.

5.6.5.12.3 printCustomerReceipt

Properties:
b

Occurence:
Conditional.

Description:
This boolean with the value *true* indicates the content of *'customerReceipt'* must be printed.

5.6.5.12.4 askCustomerSignature

Properties:
b

Occurence:
Conditional and present when a card with authentication by signature is used during the transaction.

Description:
The optional tag *'askCustomerSignature'* indicates if a signature of the cardholder is needed. When this Boolean attribute is "*true*" the POS itself shall create a box containing the message "VRAAG HANDTEKENING". Although it is mandatory to show this text at least six seconds to the cashier, it is advised to keep this box available on the screen until the cashier presses/clicks it away.

When this text occurs on the screen the cashier shall ask the cardholder to sign the *merchantReceipt* in the signature box on the ticket and check the signature against the signature available on the card.

A combination of the flags *'askCustomerSignature'* and *'askCustomerIdentification'* is possible.

5.6.5.12.5 askCustomerIdentification

Properties:
b

Occurence:
Conditional and present when a card is used during the transaction which needs the Identification of the cardholder to be noted down.

Description:
The optional tag *'askCustomerIdentification'* indicates if an identification of the cardholder is needed. When this Boolean attribute is "*true*" the POS itself shall create a box containing the message "VRAAG

IDENTIFICATIE". Although it is mandatory to show this text at least six seconds to the cashier, it is advised to keep this box available on the screen until the cashier presses/clicks it away.

When this text occurs on the screen the cashier shall ask the cardholder for his number of his passport or drivers license en write this information on the *merchantReceipt*.

A combination of the flags *'askCustomerSignature'* and *'askCustomerIdentification'* is possible.

5.6.5.12.6 askMerchantSignature

Properties:
b

Occurence:
Conditional and present at refund transactions.

Description:
The optional tag *'askMerchantSignature'* indicates if a signature of the merchant is needed on the cardholder receipt. When this Boolean attribute is "*true*" the POS itself shall create a box containing the message "ZET HANDTEKENING". Although it is mandatory to show this text at least six seconds to the cashier, it is advised to keep this box available on the screen until the cashier presses/clicks it away.

When this text occurs on the screen the cashier shall the sign the *cardholderReceipt* in the signature box on the ticket and give the signed ticket to the carholder. It is up to merchant policy what to do with the *merchantReceipt*.

5.6.5.12.7 jounalReceipt

Properties:
0..65535a

Occurence:
Conditional. The "*JournalReceipt*" is only present in case a card is recognized as a card, supported by the payment terminal.

Description:
The "*JournalReceipt*" is a text based report on transaction level and contains information intended for the merchant. This information can optionally be stored in a merchant system. It is optional to store because the same information is available in the *PSP Backoffice* where it is stored for permanently
The "*JournalReceipt*" is an escaped JSON string. The content can be parsed to a JSON array. The content of a receipt is pre-formatted by the payment terminal and may not be changed. Even empty lines must be printed. Using a monospace font (like Courier New) showing the *JournalReceipt*, it is properly aligned.

5.6.5.12.8 eJournal

Properties:
0..65535a
The "*eJournal*" is an XML string

Occurence:
Conditional. The "*eJournal*" is only present in case a card is recognized as a card, supported by the payment terminal.

Description:
"*eJournal*" is an XML report on transaction level and contains information intended for the merchant. This information can optionally be stored in a database of a merchant system. It is optional to store because the same information is available in the *PSP Backoffice* where it is stored permanently. Because it is XML based, it

is easy to be stored in a database where later on queries can be taken. This information is advised to be used to see which card brand is used during the payment and for example to create totals based on card brand and booking period.

5.6.5.12.9 operationEnvironment

Echo of the "*operatingEnvironment*" present in the *CreateNewTransactionRequest*.

See operatingEnvironment (on page 29) for more details.

5.6.5.12.10 merchantLanguage

Echo of the "*merchantLanguage*" present in the *CreateNewTransactionRequest*.

See merchantLanguage (on page 30) for more details.

5.6.5.12.11 managementSystemId

Echo of the "*managementSystemId*" present in the *CreateNewTransactionRequest*.

See: managementSystemId (on page 30) for more details.

5.6.5.12.12 terminalId

Echo of the "*terminalId*" present in the *CreateNewTransactionRequest*.

See: terminalId (on page 30) for more details.

5.6.5.12.13 accessProtocol

Echo of the "*accessProtocol*" present in the *CreateNewTransactionRequest*.

See: accessProtocol (on page 30) for more details.

6 Environments

Next to the production environment and production terminals CCV supplies a test environment and test terminals.

6.1 Production Environment

The Merchant WebApplication has to use the following parameters to access the production environment:

Host: https://api.psp.ccv.eu/api/v1

"terminalId": "{TMS_TID}" // TMS Terminal of Payment Terminal

"managementSystemId": "GrundmasterNL" //for terminals administrated by TMS NL

"managementSystemId": "GrundmasterBE" //for terminals administrated by TMS BE

"accessProtocol": "OPI_NL"

API keys test => for CCC not defined/allowed in production environment.

API keys live => merchant specific

6.2 Test environment

The *Merchant WebApplication* has to use the following parameters to access the test environment:

Host: https://vpos-test.jforce.be/vpos/api/v1

"terminalId": "{TMS_TID}" // TMS Terminal of Payment Terminal

"managementSystemId": "GrundmasterNL-ThirdPartyTest" //for test both NL and BE

"accessProtocol": "OPI_NL"

API keys live (starting with l_) are used in combination with an external environment consisting of a real test terminal with test cards and Acquiring test host

API keys test (starting with t_) are used in combination with an simulated terminal running at 188.165.118.154: 6245

Note: if used the *ReadTransactionRequest* must contain:

"ip": "188.165.118.154",

"port": "6245",

See https://api.psp.ccv.eu/api/v1/doc/testing for more details and note that the port used is 6245.

6.3 Test terminal

A test terminal is used in the test environment. This terminal does not connect to real acquiring banks but with simulated acquirers. A test terminal can be identified by looking at the top line of the terminal screen. For a test terminal, it will say there alternately "TEST TEST TEST TEST" and "!!! DO NOT PAY !!!" as shown below. At a production terminal this is not present.

A terminal must contain SW version 14.0.11 or higher.

The TMS TID (in the VPOS PSP API protocol know as "*terminalId*") can be found by pushing the PINpad buttons [STOP], [OK], [CORR], [CORR], [CORR] within 30 seconds after the screen BUITEN GEBRUIK or WELKOM is shown after power on. Now enter the Service Password and select DISPLAY INFO and TERMINAL. The terminal ID is shown behind the text "TMS TID:"

The IP-address (in the VPOS PSP API protocol known as "*ip*") can be found by pushing the PINpad buttons [STOP], [OK], [CORR], [CORR], [CORR] within 30 seconds after the screen BUITEN GEBRUIK or WELKOM is shown after power on. Now enter the Service Password and select DISPLAY INFO. The IP address is shown below the text "ETHERNET".