PAYMENTS API INTEGRATION DOCUMENT

V1.1-9 February 2020

CashFlows™

INTRODUCTION

CashFlows delivers a range of services designed to help businesses manage their payments. These services are delivered through a single, omnichannel platform accessed via a set of powerful APIs. The Payments API is part of that suite, providing cardholder not present (e-commerce, mobile and mail order telephone order) credit and debit card acquisition, with support for both card authentication via 3D-Secure, and card authorisation through the major worldwide schemes.

If you wish to accept cardholder present transactions through physical terminals, you will need to make use of the Acquiring API instead, but please contact our customer services team to discuss your options.

How does the CashFlows Payments API work?

1. A consumer selects a product or a service to purchase from your mobile or online store.
2. The consumer's payment card details are entered via an online payment page, Virtual Terminal, or card wallet.
3. These details, along with information about the transaction (such as Amount and Currency) are forwarded by your website or gateway to our Payments API service.
4. We send the payment card details via the card scheme networks to the consumer's card issuing bank for authorisation.
5. The card issuer checks the card details, that the cardholder's account has enough funds and that the card hasn't been reported lost or stolen. If everything is OK, the issuing bank authorise the amount requested and debit those funds from the consumer's payment card account.
6. The authorisation results are returned to you via the Payment API response, and subsequently your customer, the cardholder, confirming the result of the transaction
7. We receive the funds from the card scheme networks and then remit them into your remittance account. From there, you can transfer those funds to your business bank accounts, normally using automated processes set up when your merchant account is created.

CashFlows Payments API

The Payments API provides straightforward programmatic access to functions that request card authentications, authorisations, fund captures, refunds and other operations that can be performed on a payment card.

It provides these functions through a synchronous, secure, POST-Response mechanism using a single Merchant Number to indicate which of your merchant account you wish to use for transaction processing.

Requests can be supplied in XML or JSON format, and responses received in either, dependent upon the Accept Header you request. All calls are stateless, HTTPS POSTs to specific command URIs.

The mechanism used to sign Payments API requests (SHA2-512), is consistent with other CashFlows APIs, allowing re-use of integration code across multiple CashFlows endpoints.

TABLE OF CONTENTS

Introduction *2*
How does the CashFlows Payments API work? 2
CashFlows Payments API 2
Table of Contents *3*
Sensitive Authentication Data and PCI-DSS *4*
Service URL *4*
Request/Response Formats *5*
Online Documentation *7*
Calculating Request Signatures *7*
Example of Signature Calculation in C# 9
Testing 9
Payments API Commands *10*
Authorisation 10
Payment Types 10
Auth 10
Payment 10
Recurring Payment 10
Verify 10
Recurring Auth 10
3D Secure 11
Tokens 11
Capture 11
Credit 11
Refund 12
VerifyThreeDSecure 12
Void 12
Response and Error Codes *12*
Address Verification and CVV Check Response Codes *17*
Revision History *18*
Copyright Notice *20*

SENSITIVE AUTHENTICATION DATA AND PCI-DSS

Using the Payments API to send payment card data means that you will be capturing, transmitting, and possibly storing credit/debit card data. The Card Schemes (Visa, Mastercard, American Express and others), do not permit the storage of Sensitive Authentication Data (track data and/or CVV2) post-authorisation and it is prohibited under Requirement 3 of the Payment Card Industry Data Security Standard (PCI-DSS).

If you use the Payments API you will need to demonstrate that your systems handle this data securely and that you are taking full responsibility for your PCI-DSS compliance. This includes, but is not limited to, providing your current Attestation of Compliance certificate and evidence of a recent clean vulnerability scan.

A list of approved Security Assessors can be found at:
https://www.pcisecuritystandards.org/assessors_and_solutions/qualified_security_assessors

For further information on PCI security standards, please visit the following web page:
https://www.pcisecuritystandards.org

SERVICE URL

A single API endpoint is provided for all customers for all API functions. Customers must identify themselves in each request using a unique ApiKey, and sign the message using a security token and SHA2-512 hash.

All calls use HTTPS on the standard port 443. TLS v1.2 must be used, in line with PCI-DSS requirements.
Earlier versions of TLS, and SSL of any version, are not supported.

The URLs for the Payments API are:

https://integration.cashflows.com/payments/[commandname] for the Integration/Sandbox environment.

https://live.cashflows.com/payments/[commandname] for the Production/live environment.

The [commandname] is the name of the service being requested, such as authorisation or capture. For URLs are provided in each command detail section below.

REQUEST/RESPONSE FORMATS

All Payments API commands share a common format, with the specific fields for those commands in a Request node (see the [Payments API Reference Documentation](#)).

Requests can be formatted in XML or JSON, and the **Accept header** should be set to "application/json" or "application/xml" to specify the format of the Response you wish to receive. If no Accept header is specified, JSON is assumed.

IMPORTANT NOTE: Field names and values in JSON are not case sensitive, but in XML **BOTH** field names and values are case sensitive. As an example, the "Is3Ds" field must be named using the cases shown and set to "True" or "False". If you name your field "IS3DS" in XML, the field will be ignored. The tables in the commands section below, show the correct cases to use in XML messages.

All Requests are made by POSTing the following Request structure to the command URI. Version, ApiKey, Request and Signature are supplied for all commands:

```
In JSON format

{
    "Version": "1.1",
    "ApiKey": "YourApiKey",
    "Request": {
      "Field1": Value1,
      "Field2": Value2
    },
    "Signature": "SHA512 hashed Request node contents and passphrase"
}

Or in XML

<Version>1.1</Version>
<ApiKey>YourApiKey</ApiKey>
<Request>
      <Field1>Value1</Field1>
      <Field2>Value2</Field2>
</Request>
<Signature>SHA512 hashed Request node contents and passphrase</Signature>
```

If an invalid Version number is specified, the latest is assumed. The ApiKey field contains the unique key issued to you to access this API, and the Signature is a hash of your security token and the contents of the Request node (see the section below for examples). The Fieldx and Valuex fields are specific to each command and are provided in detail in the API Reference Documentation.

Successful responses from each command also share a common format:

```
In JSON format

{
  "Version": "1.1",
  "DateTime": "2018-07-11T13:52:08.5842645Z",
  "Response": {
    "Field1": Value1,
    "Field2": Value2
  }
}

Or in XML

<Version>1.1</Version>
<Datetime>2018-07-11T13:52:08.5842645Z</Datetime>
<Response>
       <Field1>Value1</Field1>
       <Field2>Value2</Field2>
</Response>
```

All successful responses contain:

- the Version number of the Response being delivered (which will match that in the Request if that Version is valid)
- the current UTC DateTime field (in ISO-8601 format)
- a Response object with command-specific nodes within it

Unsuccessful responses are similar in format to successful ones, but with an Error object instead of a Response object:

```
In JSON format

{
  "Version": "1.1",
  "DateTime": "2018-07-11T13:52:08.5842645Z",
  "Error": {
    "Code":“String value”,
    "Message": “Human readable message”
    "Details”:{Empty, or a collection of multiple Code and Message objects}
  }
}

Or in XML

<Version>1.1</Version>
<Datetime>2018-07-11T13:52:08.5842645Z</Datetime>
<Error>
       <Code> String Value</Code>
       <Message>Human Readable Message</Message>
       <Details>Empty, or a collection of multiple Code and Message objects</Details>
</Error>
```

All error responses contain:

- the Version number of the Response being delivered (which will match that in the Request if that Version is valid)
- the current UTC DateTime field (in ISO-8601 format)
- an Error object containing an overall error Code and Message with, optionally, a Detail object containing finer grained error information.

ONLINE DOCUMENTATION

Full online command documentation can be obtained on the URL below via a Swagger interface.

https://integration.cashflows.com/payments/documentation/index.html

CALCULATING REQUEST SIGNATURES

To ensure that the API requests have been issued by valid users, messages must be signed. The Signature field is calculated by concatenating the Security token with the contents of the Request node (represented as a string) and calculating a SHA2-512 hash of that concatenated string.

The signature must be correct to receive a non-error response. Repeated use of an incorrect signature will lock out the ApiKey, and an administrator will need to unlock it.

Using the following example ApiKey and Security Token:

```
ApiKey: 12345678-1234-1234-1234-1234567890ab
Security Token:
3031E5834AAD94B05C563292E6590ED13336501627EF1248036838C9BEBC08226A030134B3D791B488C086A97EA521FB192BD
578CD41583DCB6DC21A896A497E
```

…we can create a message to send to the Capture command to capture funds previously authorised on a transaction with ID 2345678. To do this, we create the following “Request” node:

```
"Request": {"TransactionId": 2345678}
```

We then pre-pend the security token to the contents of the “Request” node:

```
3031E5834AAD94B05C563292E6590ED13336501627EF1248036838C9BEBC08226A030134B3D791B488C086A97EA521FB192BD
578CD41583DCB6DC21A896A497E"TransactionId": 2345678
```

And calculate the SHA2-512 hash of that string:

```
13D8C822AE18AD0A023806A3225682DC22C652D2514498E5DEDC050BD35B1F11BB53BD73F78EA3A631C446253D7DFF87F0DAD
6DA543E84711A9A3C68352D741D
```

Then build the Capture message with the Version, ApiKey, Request node and the above result as the Signature:

```
{
    "Version": "1.1",
    "ApiKey": "12345678-1234-1234-1234-1234567890ab",
    "Request": {"TransactionId": 2345678},
    "Signature":
"13D8C822AE18AD0A023806A3225682DC22C652D2514498E5DEDC050BD35B1F11BB53BD73F78EA3A631C446253D7DFF87F0DA
D6DA543E84711A9A3C68352D741D"
  }
```

On the CashFlows Sandbox environment, this would be POSTed to:

https://integration.cashflows.com/payments/capture/

Below is the same example but in XML.

XML Request node:

```
<Request>
  <TransactionId>2345678</TransactionId>
</Request>
```

Now with the password hash pre-pended the string looks like this (**IMPORTANT NOTE** in this example there is whitespace and linefeeds, and these **are** included in the Hash calculation!):

```
3031E5834AAD94B05C563292E6590ED13336501627EF1248036838C9BEBC08226A030134B3D791B488C086A97EA521FB192BD
578CD41583DCB6DC21A896A497E
  <TransactionId>2345678</TransactionId>

```

Which produces a SHA2-512 Hash of:

```
EAC92EE0431CC72192D1D4272E1B4A0CC29F209FA9C65F906D88629F69F60B3D827BAF09A35627AED47091A3B7EC5D8311445
499D15D6315C108530177BE92AE
```

So, with the other required fields added, the body POSTed to the Capture command URI would be:

```
<Version>1.1</Version>
<ApiKey>12345678-1234-1234-1234-1234567890ab</ApiKey>
<Request>
  <TransactionId>2345678</TransactionId>
</Request>
<Signature>EAC92EE0431CC72192D1D4272E1B4A0CC29F209FA9C65F906D88629F69F60B3D827BAF09A35627AED47091A3B7
EC5D8311445499D15D6315C108530177BE92AE</Signature>
```

For JSON requests (like the examples in this document) the request string to be hashed begins after the { following the word "Request" and ends at the } at the end of the node. The open and close curly brackets {} should not be included in the hash calculation.

For XML requests, the request string starts after the > of <Request> and ends at the < of </Request>. Again the close and open angled brackets >< should not be included in the hash calculation.

IMPORTANT: All whitespace and new lines within the request nodes **will be** included in the calculation of the Hash Value. CashFlows uses CR-LF for line breaks; Unix systems often use just LF, and this can affect calculations. If you are unable to match signatures with the Payments API and have the correct Security token and ApiKey, consider removing unnecessary whitespace from your Request nodes.

Example of Signature Calculation in C#

```
System.Security.Cryptography.SHA512 sha512 = System.Security.Cryptography.SHA512.Create();
System.Text.StringBuilder builder = new System.Text.StringBuilder();

byte[] bytes = System.Text.Encoding.UTF8.GetBytes(
       passwordHash + requestBody);

byte[] hashedBytes = sha512.ComputeHash( bytes );
foreach( byte b in hashedBytes )
{
       builder.Append( b.ToString( "x2" ) );
}
string signature = builder.ToString();
```

Testing

When integrating with the Payments API, or testing new code, you should use the Integration environment. This is the CashFlows Sandbox, which is identical to the Production platform but with simulated card-scheme responses. It is not possible when using the Integration environment to obtain real authorisations.

The URL for the Integration end-point is:

https://integration.cashflows.com/payments/[commandname]/

You may need to have generated some test data using the API to fully build and test. For example, you must have a valid authorised Transaction ID to attempt a Void operation. Likewise, the Refund command requires an original captured Transaction ID. Please contact our Implementations team (contact details at the end of this document) for more information about test accounts and data, if you are unsure how to do this.

Test transactions can be sent using our test cards:

Card Number	Token	Expiry Date	CVV
4000000000000002 (VISA)	1000000000030419	Any valid expiry date	123
4462030000000000 (VISA prepaid)	1000000000030554	Any valid expiry date	444
5555555555554444 (Mastercard)	1000000000030567	Any valid expiry date	321
5597507644910558 (Mastercard prepaid)	1000000000030568	Any valid expiry date	888
340001916255521 (AMEX)	1000000000030565	Any valid expiry date	1234

PAYMENTS API COMMANDS

Authorisation

Five different operations can be performed using the Authorisation command. The uses of each are outlined below.

Payment Types

Auth

Setting the PaymentType parameter to Auth will process a simple authorisation request to reserve funds on the cardholder's card. It should be noted that a secondary request must then be made to Capture the transaction, either individually or as part of a batch. If this step is not taken within 7 consecutive days the transaction will not be submitted for clearing, and the authorisation will be automatically removed by the issuing bank.

Payment

Authorisation and Capture in a single API operation is supported when the PaymentType is set to Payment. There is no need to send a secondary request to capture the authorisation; it is automatically batched and submitted for processing at the end of the day.

RecurringPayment

Sending the PaymentType parameter as RecurringPayment indicates that the transaction is one in a series of regular payments. Where set, and a reference to an original authorisation, payment or verify is given in the ParentTransactionId parameter, it is not necessary to include card data in the request. The parent transaction is used to retrieve the stored card details. The card details may alternatively be sent in every request (except the CVV, which must not be stored).

If using RecurringPayment, the authorisation will be processed according to the default recurrence settings on the Merchant Number requested. You can override the default by passing a RecurrenceType.

Please note that Recurring Payments must be specifically configured on your Merchant Numbers. Please contact the Customer Services team if you are unsure about your account set-up.

Verify

Passing PaymentType as Verify offers the opportunity to confirm that the card exists, is valid for use, that address data (where provided) passes AVS checks and that the consumer can authenticate via 3DS, without decrementing the cardholder's balance. This is also known as a zero-value authorisation.

RecurringAuth

Setting the PaymentType parameter to RecurringAuth indicates that the transaction is one in a series of regular payments and will process an authorisation request to reserve funds on the cardholder's card. Where set, and a reference to an original authorisation, payment or verify is given in the ParentTransactionId parameter, it is not necessary to include card data in the request. The parent transaction is used to retrieve the stored card details. The card details may alternatively be sent in every request (except the CVV, which must not be stored).

It should be noted that a secondary request must then be made to Capture the transaction, either individually or as part of a batch. If this step is not taken within 7 consecutive days the transaction will not be submitted for clearing, and the authorisation will be automatically removed by the issuing bank.

3D Secure

You can call Authorisation with Is3DS set to True, in which case you must also provide the following data which will have been provided from 3D-Secure Authentication output from your own Merchant Plug In:

- Xid
- Cavv
- Eci

Tokens

Token is a unique reference to a card number. An authorisation request may include a card number or the corresponding token. Setting the ReturnToken flag in authorisation request will return the token in the response.

Capture

Sending a Capture request following an Authorisation is necessary to received money from the cardholder, unless the PaymentType is Payment or Recurring Payment (these are automatically batched for processing).

The Capture action ensures that a previous Authorisation is included in the next processing batch and is submitted to the scheme for settlement. Unless captured, an authorisation will automatically expire after a number of days as determined by the issuing bank. All capture requests are for the full transaction amount; partial capture is not supported at present.

Credit

Used to request a Credit to the payment card associated with a previous authorisation or payment. Alternatively, for Visa branded cards, you can send the full Primary Account Number (PAN), or the tokenized value for the PAN (CardToken).
If you wish to refund the money to a cardholder with reference to a previous authorisation/capture or payment, you should use the Refund command.

NOTE: Your Merchant Account must be set up to accept credit transactions (Visa OCT and Mastercard CFT). Please contact customer services if you are uncertain.
Credit transactions are only supported for the following MCC's:

- 7995 - Gambling
- 7994 – Game of skill
- 6010 - Financial Institutions – Manual Cash Disbursements
- 6011 – Financial Institutions – Automated Cash Disbursements
- 6012 - Financial Institutions – Merchandise, Services, and Debt Repayment
- 6300 - Insurance Sales, Underwriting, and Premiums
- 6399 - Insurance, Not Elsewhere Classified
- 8999 – Professional Services (Not Elsewhere Classified)
- 5262 – Marketplaces

Refund

A refund request can be sent for a captured authorisation or payment originally secured via the Payments API. Partial refunds are supported, but the amount is capped to the value of the parent transaction. In circumstances where a greater amount must be returned to the cardholder, the Credit command should be used.

VerifyThreeDSecure

VerifyThreeDSecure command is used to check whether a customer's card is enrolled in 3DSecure. This can eliminate the need to send the card account information for 3DS Auth if the card is not enrolled in the service.

Void

Used to request the voiding of an authorisation previously obtained using the Authorisation command. The request will be accepted only if the authorisation has NOT been captured. After an Authorisation has been captured, the Refund command should be used.

Iin

Used to return details of that card such as scheme (Visa, Mastercard or American Express), type (credit, debit, corporate etc.) and Issuer (where known). The request should include at least the first 6 digits but can be a full Primary Account Number (PAN)/card number, or any combination between 6 and full PAN.

RESPONSE AND ERROR CODES

The following http status codes may be returned in the header of the response, if there is an error:

Code	Description	Troubleshooting
400	Bad request	Check parameters for correct formation and that all mandatory items are present
403	Forbidden	Check the signature calculation
429	Too many requests	Either rate limits have been exceeded or replay protection has been triggered.
500	Internal server error	Please get in touch with your implementations contact if this issue arises on Integration, or your Relationship Manager or Support should it occur in Production

The following codes may be returned in the standard message response parameter when the transaction request is unsuccessful.

Status 'A' is authorised, anything else is not. The auth code and auth message for authorised transactions cannot be predicted (as they can change from one bank/issuer to the next).
'V' is a validation error (e.g. invalid card number)
'D' is a decline
'R' is a referral (must be treated as a decline)

‘B’ is a blocked transaction
'C' is a cancelled transaction (e.g. user pressed cancel on payment page)
'S' is a system error

These will be followed by a 3-digit code, the first digit is an internal code which can be ignored. The second two digits are the actual error code for the given status.

The list is given as, for example, Vx01 which means it is the result for V101, V201, V301 etc

Code	Description
Vx01	Invalid merchant details
Vx02	Invalid expiry date
Vx03	Invalid start date
Vx04	Invalid issue number
Vx05	Invalid CVV
Vx06	Invalid card number
Vx07	Card holder name not set
Vx08	Insufficient address details
Vx09	Invalid country code
Vx10	Invalid cart ID
Vx11	Invalid email address
Vx12	Invalid phone number
Vx13	Invalid amount
Vx14	Invalid currency code
Vx15	Invalid customer IP
Vx16	Original trans not found
Vx17	Invalid merchant IP
Vx18	Unknown transaction type
Vx19	Card number changed
Vx20	Currency changed
Vx21	Original trans ref required
Vx22	Amount exceeds original
Vx23	Cannot refund this type of transaction
Vx24	Amount changed
Vx25	User account details required

Vx26	Invalid request
Vx27	Original trans not pre-auth
Vx28	Transaction mode changed
Vx29	Card/Currency combination not supported
Vx30	Unknown card type
Vx31	Issue number required
Vx32	Issue number not required
Vx33	Duplicate transaction
Vx34	Unable to void transaction
Vx35	Original trans was not authorised
Vx36	Invalid PIN
Vx37	Unknown transaction class
Vx38	Original transaction type does not match
Vx39	Card expired
Vx40	CVV Required
Vx41	Original transaction already settled
Vx42	Original transaction already cancelled
Vx43	This card does not support the required transaction type
Vx44	Transaction details do not match original
Vx52	3DS Not Enabled
Vx53	3DS Data Invalid
Vx54	Concurrent authorisations
Vx55	Invalid Funds Recipient Date (MCC 6012, 6051 or 7299 Merchants)
Vx56	Terminal mismatch
Vx57	Transaction not allowed on this card
Vx58	Original transaction requires 3DS attempt/auth
Vx59	ECOM transactions require 3DS attempt/auth
Vx60	Verify for Amex card not supported
Vx61	Recurrence Flag usage invalid
Vx62	Initial Sale/Verify ARN missing for subsequent sale
Vx63	Initial Sale/Verify for subsequent sale not approved
Vx64	Initial transaction on card expired

Dx01	Non-specific decline
Dx02	Declined due to funds (insufficient/limit exceeded)
Dx03	Retain card response
Dx05	On our blacklist
Dx07	Live/test mismatch
Dx08	Refund: Insufficient merchant funds in account
Dx10	Card authorisation attempt limit reached
Dx11	Monthly Scheme Decline Rate limit reached
Dx40	Continuous Authority cancelled for the transaction
Dx41	Continuous Authorities cancelled for the merchant
Dx43	Continuous Authorities cancelled for the card
Dx44	Function not supported
Dx45	Incorrect CVV
Dx46	Incorrect Start Date
Dx48	Invalid Currency Code
Dx47	Card Number Changed
Dx90	Pre-authorisation anti-fraud block
Dx91	Post-authorisation anti-fraud block
Rx01	Referral
Ex01	Transaction error
Cx01	Transaction cancelled
Cx02	Transaction expired
Sx00	Invalid transaction request
Sx01	Connection failure
Sx02	Invalid response
Sx03	Response timeout
Sx04	Server error
Sx05	Server error
Sx06	No response from issuer
Sx07	Service not available
Sx99	Unknown Error

ADDRESS VERIFICATION AND CVV CHECK RESPONSE CODES

The CVV/AVS result is a 3-digit value, indicating the result of checking the three or four digit card verification values, and the address details supplied during authorisation. Each response digit representing a different check.

The first value is the CVV check, the second is the Address and the third is the Postcode.

Response Code	Description
0	Not checked
1	Check was not available
2	Full match
3	Partial match
4	Not matched
5	Error

Example response	CVV match results	Address verification results	Postcode match results
232	Full match	Partial match	Full match
400	Not matched	Not checked	Not checked

Please note:

- A partial match is only possible for the address or postcode data, not for CVV check
- Not all banks support all these checks, in which case the results will be either 0 or 1

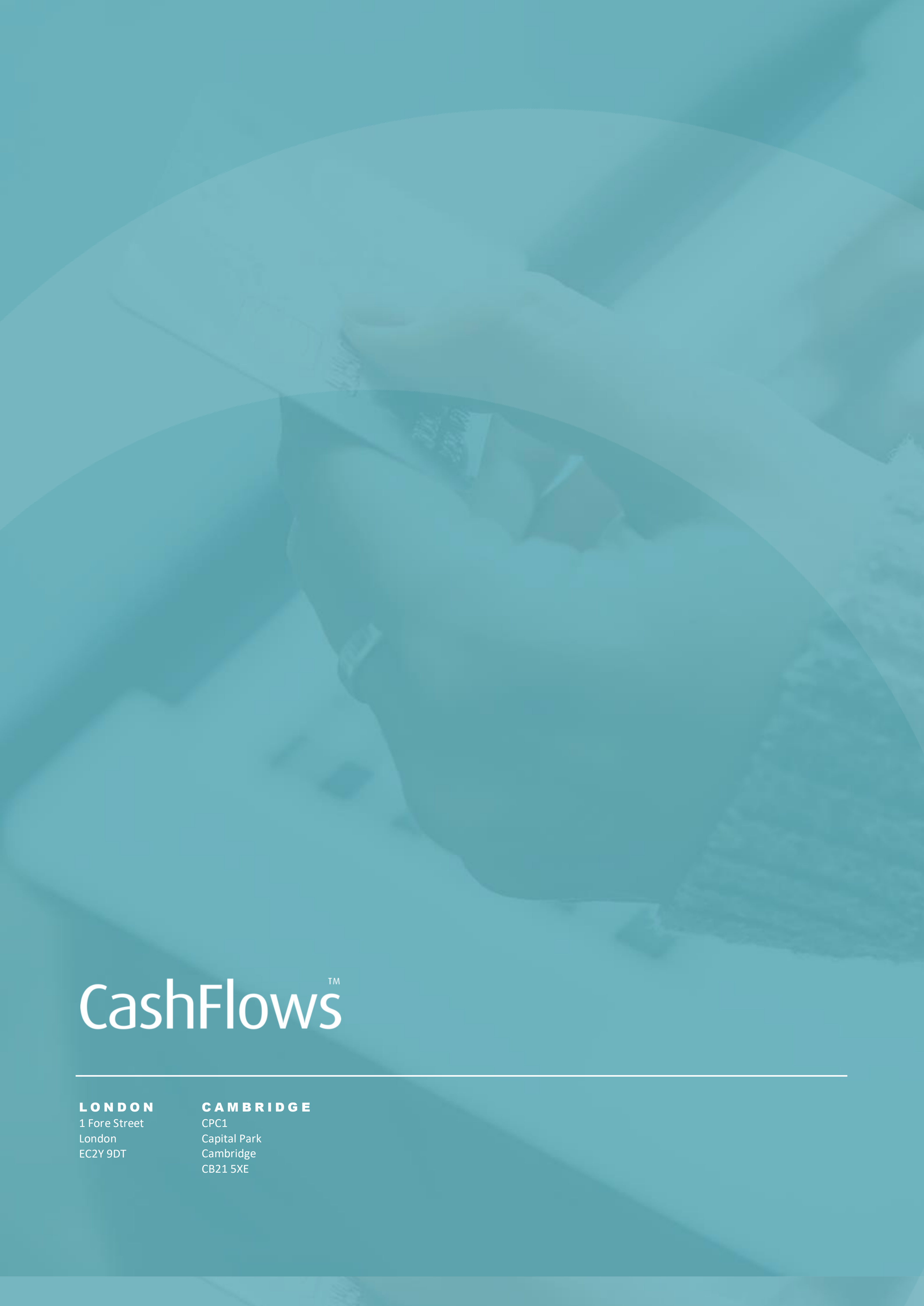
REVISION HISTORY

Date	Summary of Changes	Version No.
18/11/2016	Initial release with Auth and Capture commands.	0.1
12/01/2017	Updated with Payment, Refund, and Void commands.	0.2
06/07/2017	Updated with Verify, Credit and Three D Secure commands.	0.3
17/10/2017	Updated with revisions to Authorisation and Payment commands. Added Batch Capture.	0.4
06/12/17	Updated signature calculation section. Updated Authorisation command with new fields. Updated MerchantID parameter to SourceID throughout. Removed Payment, Recurring Payment and Verify commands. Added Error codes section.	1.0
20/12/17	Added Sensitive Authentication Data and PCI-DSS section. Expanded overall document introduction. Expanded introductions to each command section. Corrected descriptions of the Is3DS and Requires3DS parameters on Authorisation. Revised Authorisation parameter CardholderCountry to mandatory.	1.1
31/08/2018	Removed references to trailing slashes in Base URL section. Updated Request Formats section with current examples. Updated Online Documentation section with current examples. Updated Calculating Request Signatures section to • remove references to Portal (a decommissioned interface) and legacy passwords. • provide clearer advice on line break and whitespace handling. Updated base URI and all command URLs. Removed signature calculation information from the Payments API Commands section to the Calculating Request Signatures section. Updated each command's subsection with the command URLs for Integration and Live. Revised TransactionID length from 40 characters to 11 in all references. Expanded Authorisation section to improve authorisation type descriptions. Added container arrays to Authorisation Request Parameters section. Removed legacy request parameters from Authorise, Capture, Credit, Refund and Void. Updated Authorise request with new optional request parameters. Updated Capture with new mandatory request parameters. Updated Credit, Refund and Void with new response parameters. Removed legacy commands Batch Capture and ThreeDSecure.	1.1-2
08/10/2018	Reworded and reformatted to increase readability. Updated with APIKey and MerchantId.	1.1-3
30/10/2018	Updated with correct PascalCase in code examples.	1.1-4
13/11/2018	Used Strings rather than ENUMs to provide clarity for XML connectivity.	1.1-5
22/11/2018	Better detail in signature calculation example.	1.1-6
28/01/2019	Added RecurringAuth PaymentType.	1.1-7
12/06/2019	Updated with card tokenisation changes. Reformatted to incorporate automatically generated API reference document. Addition of a new validation error code Vx64.	1.1-8

26/11/2019	Updated with BIN lookup (Lin) command.	1.1-9

COPYRIGHT NOTICE

While every effort has been made to ensure the accuracy of the information contained in this publication, the information is supplied without representation or warranty of any kind, is subject to change without notice and does not represent a commitment on the part of CashFlows Europe Group. CashFlows Europe Group, therefore, assumes no responsibility and shall have no liability, consequential or otherwise, of any kind arising from this material or any part thereof, or any supplementary materials subsequently issued by CashFlows Europe Group. CashFlows Europe Group has made every effort to ensure the accuracy of this material.



CashFlows™

LONDON
1 Fore Street
London
EC2Y 9DT

CAMBRIDGE
CPC1
Capital Park
Cambridge
CB21 5XE