blink

Gateway Integration Guide

V1.21

blink powered by fidelitypayment

Version	Date	Update information
1.01	13/11/2020	Added version and version control to guide.
1.02	10/12/2020	Added Stored Credentials section.
1.03	12/12/2020	Removed reference to E-Receipts.
1.04	27/01/2021	Added Section 23 – Digital Wallets (only Google Pay).
1.05	28/01/2021	Added Apple Pay to Digital Wallets section.
1.06	17/03/2021	Integrated separate 3DSv2 Guide.
1.07	24/03/2021	Amended Test Amounts and Test Card Details.
1.08	25/03/2021	Added rtSequenceCount field to Recurring Transactions section.
1.09	25/03/2021	Updated 3DS sample code.
1.10	26/03/2021	Updated Test Amounts.
1.11	01/04/2021	Edited the description for error codes 65794 and 65796. Edited RT Agreements section to confirm that receipts are not sent for CA transactions.
1.12	06/04/2021	Amended Hosted Form section.
1.13	07/04/2021	Amended section 1.5.8 to remove XML format.
1.14	09/04/2021	Updated description of error code 66086.
1.15	18/04/2021	Updated Google Pay section and removed reference to separate 3DS guide on page 36.
1.16	21/04/2021	Updated test card expiry date guidance under section A-12.2.2.
1.17	22/04/2021	Added further clarification for values for threeDSEnrolled and threeDSAuthenticated fields.
1.18	30/04/2021	Added further clarification about the fonts that can be used with the hosted form.
1.19	06/05/2021	Amended incorrect numbering and title in 3-D Legacy API section.
1.20	07/05/2021	Added `formAmountEditable` to table 4.1.
1.21	19/05/2021	Added Device Fields section to section 17 and added further details of 3DSv2 options under section 6.

blink

CONTENTS

blink

1 Gateway Integration

1.1 About This Guide

This guide provides the information required to integrate with our Payment Gateway and gives a very basic example of code for doing so It is expected that you have some experience in server-side scripting with languages such as PHP or ASP; or that an off-the-shelf software package is being used that has inbuilt or plug-in support for our Gateway.

1.2 Terminology

The following terms are used throughout this guide:

Gateway
The Payment Gateway.

Merchant
The Merchant using the Gateway's services.

Our
The Payment Gateway Provider.

You/your
The Merchant or its representative performing the integration.

Acquirer
The bank or financial institution used by the Merchant.

Customer
A Customer of the Merchant making a payment.

Card
A payment credit, debit, prepayment or gift card issued by the Card Schemes.

Card Scheme
The operator of a payment Card network, such as Visa, Mastercard, et al.

Cardholder
The person who owns the payment Card, usually the Customer.

Issuer
The bank or financial institution that issued the payment Card to the Cardholder.

Merchant Account
An account on the Gateway mapped to an Acquirer-provided account.

Checkout
Third-party checkout solution such as PayPal, Amazon Pay other alternative payment methods.

Wallet
Third-party wallet solution such as Masterpass.

Hosted Payment Page (HPP)
A page hosted on our secure server used to collect Customer details.

Hosted Payment Field (HPF)
An individual form field hosted on our secure server used to collect sensitive Cardholder data.

1.3 Integration Methods

There are three methods of integration provided to process your transactions through the Gateway, allowing for different levels of control and communication from your website.

1.3.1 Hosted Integration

The Hosted Integration method makes it easy to add secure payment processing to your ecommerce business, using our Hosted Payment Pages (HPP). You can use this method if you do not want to collect and store Cardholder data.

The Hosted Integration method works by redirecting the Customer to our Gateway's Hosted Payment Page, which will collect the Customer's payment details and process the payment before redirecting the Customer back to a page on your website, letting you know the payment outcome. This allows you the quickest path to integrating with the Gateway.

The standard Hosted Payment Page is designed to be shown in a lightbox over your website and styled with logos and colours to match. Alternatively, you can arrange for fully customised Hosted Payment Pages to be produced that can match your website's style and layout. These fully customised pages are usually provided using a browser redirect, displaying full-page in the browser, or can be displayed embedded in an iframe on your website.

For greater control over the customisation of the payment page, our Gateway offers the use of Hosted Payment Fields, where only the individual input fields collecting the sensitive Cardholder data are hosted by the Gateway while the remainder of the payment form is provided by your website. These Hosted Payment Fields fit seamlessly into your payment page and can be styled to match your payment fields. When your payment form is submitted to your server, the Gateway will submit a payment token representing the sensitive card data it collected and your webserver can then use the Direct Integration to process the payment without ever being in contact with the collected Cardholder data.

For more information, refer to our Integration Library in appendix 23.8.5A-20.

1.3.2 Direct Integration

The Direct Integration works by allowing you to keep the Customer on your system throughout the checkout process, collecting the Customer's payment details on your own secure server before sending the collected data to our Gateway for processing. This allows you to provide a smoother, more complete checkout process to the Customer.

In addition to basic sales processing, the Direct Integration can be used to perform other actions such as refunds and cancellations, which can provide a more advanced integration with our Gateway.

1.3.3 Batch Integration

The Batch Integration is an enhancement to the Direct Integration, allowing you to send multiple transactions in a single request and monitor their status. This is useful if you wish to capture multiple transactions or collect multiple payments – for example, collecting subscription charges or loan repayments.

In addition to basic sales processing, the Batch Integration can be used to perform other actions, such as refunds and cancellations, which can provide a more advanced integration with our Gateway.

Unlike the Hosted and Direct Integrations, the Batch Integration does not process transactions sent to it immediately. Instead, the Gateway queues these transactions to be processed and returns a batch reference number which can be used to download a file that contains the current status of the transactions.

Batch Processing does not support transactions that require Customer interaction such as 3D Secure transactions, or alternative payment methods with interactive Wallet or Checkout pages.

1.4 Integration Libraries

We can provide a range of libraries to help you to integrate with the Gateway.

These libraries include simple server-side classes in many popular programming languages, through to client-side scripts to help with the integration of the Hosted Payment Page or Hosted Payment Fields.

For more information about these libraries, please refer to appendix 23.8.5A-20.

1.5 Security and Compliance

Each method requires a different level of server security and compliance with the Payment Card Industry Data Security Standard (PCI DSS).

If you use Hosted Payment Pages with the Hosted Integration or Hosted Payment Fields with the Direct or Batch Integrations, then your webserver does not need an SSL certificate and you require the lowest level of PCI DSS compliance.

If your website collects and/or stores sensitive Cardholder data, such as the card number (PAN) or card security code (CVV/CV2), then your webserver must have an SSL certificate and serve all payment forms using HTTPS. You will also need a higher level of PCI DSS compliance and to complete a PCI validation form annually.

For more information, please see https://www.pcisecuritystandards.org/

1.5.1 HTTP Requests

A request can be sent to the Gateway by submitting a HTTP POST request to the integration URL provided.

The request should have a `Content-Type: application/x-www-form-urlencoded` HTTP header and the request should be name, value pairs URL encoded as per RFC 1738.

Example URL encoding:

```
merchantID=100001&action=SALE&type=1&amount=1001&currencyCode=826&countr
yCode=826&transactionUnique=55f6db1c81d95&orderRef=Test+purchase&custome
rPostCode=NN17+8YG&responseCode=0&responseMessage=AUTHCODE%3A350333&stat
e=captured&xref=15091702MG47WN32MM88LPK&cardNumber=4929+4212+3460+0821&c
ardExpiryDate=1215
```

Please note that the field names are cAsE sEnSiTiVe.

The response will use the same URL encoding and return the request fields in addition to any dedicated response field. If the request contains a field that is also intended as a response field, then any incoming request value will be overwritten by the correct response value.

1.5.2 Hosted HTTP Requests

When using the Hosted Integration, the request must be sent from the Customer's web browser as the response will be a HTML Hosted Payment Page (HPP), used to collect the Customer's details. The format of the request is designed so that it can be sent using a standard HTML form with the data in hidden form fields. The browser will then automatically encode the request correctly according to `application/x-www-form-urlencoded` format.

When the Hosted Payment Page has been completed and the payment processed, the Customer's browser will be automatically redirected to the URL provided via the **`redirectURL`** field. The response will be returned to this page in `application/x-www-form-urlencoded` format, using a HTTP POST request.

If the request contains a field that is also intended as a response field, then any incoming request value will be overwritten by the correct response value.

An example of a Hosted Integration request is provided in appendix A-21.1 and sample code is provided in appendix A-22.1.

1.5.3 Direct HTTP Requests

When using the Direct Integration, the response will be received in the same URL encoded format, unless a **`redirectURL`** field is provided.

If a **`redirectURL`** field is provided, then the response will be a HTML page designed to redirect a browser to the URL provided, using a HTTP POST request containing the response. This allows you to collect the Cardholder's payment details on your own server, using a HTML form which POSTs to the Direct Integration, which then effectively POSTs the results back to this URL your webserver, where you can display the transaction outcome.

If the request contains a field that is also intended as a response field, then any incoming request value will be overwritten by the correct response value.

An example of a Direct Integration request is provided in appendix A-21.1 and sample code is provided in appendix A-22.1.

1.5.4 Batch HTTP Requests

When using the Batch Integration, a single HTTP POST request can contain multiple individual requests using the `multipart/mixed` content type with a boundary string specified. Within that main HTTP request, each of the parts contains a nested Direct Integration HTTP request, separated by the boundary string.

Each part should begin with a `Content-Type: application/x-www-form-urlencoded` HTTP header and contain a single Direct Integration HTTP request, as documented in section 1.5.3.

You can optionally specify a `Content-Id` HTTP header to identify each part message uniquely; if not provided, the Gateway will assign a unique id to each part. The `Content-Id` HTTP header is returned in the response. The Gateway will not validate the uniqueness of any id provided. After the mandatory `Content-type` and the optional *`Content-Id`* header, two carriage return/line feed pairs must follow (i.e. $\backslash r\backslash n\backslash r\backslash n$). Any deviation from this structure might lead to the part being rejected or incorrectly interpreted. The part request payload, formatted as a regular HTTP URL encoded request, must follow the two-line breaks directly.

To reduce the size of large batch requests, the Gateway supports compression using a `Content-Encoding` HTTP header with either a 'gzip' or 'x-gzip' value. This header can be provided in the main request or in the part request or both.

An `Authorization` HTTP header can be used in the request to provide the username and password of a Gateway Merchant Management System user account. If correct, the batch details will be recorded as having been submitted by that user; if invalid, then the request will fail and respond with a 401 (Unauthorised) HTTP status code.

The Gateway will respond in the same manner as the request with a `multipart/mixed` content type; each part is the response to one of the requests in the batched request. In addition, the response will contain a standard `Location` HTTP header, providing a URL from which further batch update responses can be downloaded; and a standard `Content-Disposition` header, allowing a browser to download the response to a file. If the request contained an `Authorization` HTTP header, then the response will contain an `X-P3-Token` HTTP header containing an authentication token that can be sent in future requests instead of the username and password. The authentication token has a limited life span, but each future request will return a new token and thus effectively rejuvenate the token's life.

Like the parts in the request, each response part contains a HTTP response, including headers and body. Each response part is preceded by a `Content-Type` HTTP header and `Content-ID` HTTP header. In addition, an `X-Transaction-ID` HTTP header is added containing the requests transaction id together with an `X-Transaction-Response` HTTP header containing a textual description of the transaction processing status.

The Gateway will not process the transactions immediately but will queue them up to process over time. The transactions may not be processed in the order provided, so should not have interdependencies. Transactions will only appear in the Merchant Management System when they have been processed. The status of queued transaction is only available by querying the status of the batch.

The current status of a batch can be queried at any time by issuing a HTTP GET request to the URL provided in the initial responses `Location` HTTP header.

An `Authorization` HTTP header must be provided in the status request, containing either the username and password of a Gateway Merchant Management System user account or an authentication token returned in the batch submission response's `X-P3-Token` HTTP header. If a valid username and password or a valid token is provided, then the response will be an updated version of the initial submission response providing the current status of each transaction. The response will only contain transactions that the authenticated user has permission to view.

An example of a Batch Integration request is provided in appendix A-21.3 and sample code is provided in appendix 1.

1.5.5 Handling Errors

When the Gateway is uncontactable due to a communications error, or problem with the internet connection, you may receive a HTTP status code in the 500 to 599 range. In this situation, you may want to retry the transaction. If you do choose to retry a transaction, then we recommend that you perform a limited number of attempts with an increasing delay between each attempt.

If the Gateway is unavailable during a scheduled maintenance period, you will receive a HTTP status code of 503 ‘Service Temporarily Unavailable’. In this situation, you should retry the transaction after the scheduled maintenance period has expired. You will be notified of the times and durations of any such scheduled maintenance periods in advance, by email, and given a time when transactions can be reattempted.

If you are experiencing these errors, then we recommend you consider the following steps as appropriate for the integration method being used:

Ensure the request is being sent to HTTPS and not HTTP. HTTP is not supported and is not redirected.
Send transactions sequentially rather than concurrently.
Configure your integration code with try/catch loops around individual transactions to determine whether they were successful or not and retry if required, based on the return code or HTTP status returned.
Configure the integration so that if one transaction fails, the entire batch does not stop at that point – i.e. log the failure to be checked and then skip to the next transaction rather than stopping entirely.

1.5.6 Redirect URL

The **`redirectURL`** request field is used to provide the URL of a webpage on your server.

When provided, the Gateway will respond with a HTML page designed to redirect the Customer's browser to the URL provided, using a HTTP POST request containing the URL encoded response.

For the Hosted Integration, this will redirect the Customer from the Hosted Payment Page back to this URL on your website.

For the Direct Integration, this allows you to collect the Cardholder's payment details on your own server using a HTML form that POSTs to the Direct Integration. which then effectively POSTs the results back to this URL on your webserver, where you can display the transaction outcome. This usage is not recommended as it makes it harder to sign the message.

The URL is mandatory for the Hosted Integration and optional for the Direct Integration. It is not supported by the Batch Integration.

The **`redirectURL`** must be a fully qualified URL, containing at least the scheme and host components.

1.5.7 Callback URL

The **`callbackURL`** request field allows you optionally to request that the Gateway sends a copy of the response to an alternative URL. In this case, each response will then be POSTed to this URL in addition to the normal response. This allows you to specify a URL on a secure shopping cart or backend order processing system, which will then fulfil any order associated with the transaction.

The URL is optional for both the Hosted Integration and the Direct Integration. It is not supported by the Batch Integration.

The **`callbackURL`** must be a fully qualified URL, containing at least the scheme and host components.

1.5.8 Field Formats

Most integration field values are either numerical or textual; and either free format or from a range of predetermined values. Some field values are records or arrays of records.

Unless otherwise stated, numerical values are whole integer values with no decimal points. Textual values should use the UTF-8 character set and will be automatically truncated if too long, unless stated otherwise in the field's description. Textual values may be transliterated[1] when sending to third parties such as Acquirers but the original value is stored by Gateway and displayed in the Merchant Management System.

Field values should use the following formats unless otherwise stated in the field's description:

Field Type	Value Format
Monetary Amounts	Either major currency units by providing a value that includes a single decimal point such as '10.99'; or in minor currency units by providing a value that contains no decimal points such as '1099'.
Timestamps	Date in the format 'YYYY-MM-DD HH:MM:SS'
Dates	Date in the format 'YYYY-MM-DD'
Country Codes	Either the ISO-3166-1 2-letter, 3-letter or 3-digit code.
Currency Codes	Either the ISO-4217 3-letter or 3-digit code.
Records	Records can be provided using the *[XX]* notation, where *XX* is the record's field name (sub-field). Records can be multi-dimensional or be sequentially indexed. For example: to send a value for the sub-field *Y* in the integration field *X,* use the field name *X[Y]*; however, to send a value for the sub-field *Y* in the fourth record for integration field *X,* then use the field name *X[4][Y]* etc.
Serialised Records	Records can be sent as a JSON or URL serialised string. The first character of the serialised string determines its format: with '{' indicating RFC 7159 JSON format, and anything else is assumed to be RFC 1738 URL encoded format.

Note: Nested records are useful when posting sub-fields direct from a HTML FORM. However, unlike the main integration fields, a nested record's sub-fields are not sorted when constructing the signature and are processed in the order received. Serialised records can overcome any problems caused by a nested record's fields being received in a different order to that used when generating the signature.

[1] Transliteration involves the changing of character case, stripping of accents from characters and removal of unsupported characters so that the values meet the requirements of the third-party.

1.6 Authentication

All requests must specify which Merchant Account they are for, using the **merchantID** request field. In addition to this, the following security measures can be used:

1.6.1 Password Authentication

You can configure a password for each Merchant Account, using the Merchant Management System (MMS). This password must then be sent in the **merchantPwd** field in each request. If an incorrect password is received by the Gateway, then the transaction will be aborted and an error response is returned.

Warning: Use of a password is discouraged in any integration where the transaction is posted from a form in the client browser as the password may appear in plain text in code.

1.6.2 Message signing

You must configure a signing secret phrase for each Merchant Account using the Merchant Management System (MMS). Each request will need to be 'signed' by providing a **signature** field containing a hash generated from the combination of the serialised request and this signing secret phrase. On receipt, the Gateway will then re-generate the hash and compare it with the one sent. If the two hashes are different then the request received must not be the same as that sent and so the contents must have been tampered with and the transaction will be aborted and an error response is returned.

The Gateway will also return hash of the response message in the returned **signature** field, allowing you to create your own hash of the response (minus the **signature** field) and verify that the hashes match.

If message signing is enabled, then the data POSTed to any callback URL will also be signed.

See appendix A-13 for information on how to create the hash.

1.6.3 Allowed IP addresses

You can configure a list of IP addresses using the Merchant Management System (MMS). Two different address lists can be configured, one for standard requests, such as sales; and one for advanced requests, such as refunds and cancellations. If a request is received from an address other than those configured, then it will be aborted and an error response is returned.

1.7 Supported Actions

All requests must specify what action they require the Gateway to perform, using the **`action`** request field. The Direct and Batch Integrations support all actions; however, the Hosted Integration only supports the basic payment actions.

1.7.1 SALE

This will create a new transaction and attempt to seek authorisation for a sale from the Acquirer. A successful authorisation will reserve the funds on the Cardholder's account until the transaction is settled.

The **`captureDelay`** field can be used to state whether the transaction should be authorised only and settled at a later date. For more details on delayed capture, refer to appendix A-10.

1.7.2 VERIFY

This will create a new transaction and attempt to verify that the card account exists with the Acquirer. The transaction will result in no transfer of funds and no hold on any funds on the Cardholder's account. It cannot be captured and will not be settled. The transaction **`amount`** must always be zero.

This transaction type is the preferred method for validating that the card account exists and is in good standing; however, it cannot be used to validate that it has sufficient funds.

1.7.3 PREAUTH

This will create a new transaction and attempt to seek authorisation for a sale from the Acquirer. If authorisation is approved, then it is immediately voided (where possible) so that no funds are reserved on the Cardholder's account. The transaction will result in no transfer of funds. It cannot be captured and will not be settled.

This transaction type can be used to check whether funds are available and that the account is valid. However, due to the problem highlighted below, it is recommended that Merchants use the VERIFY action when supported by their Acquirer.

Warning: If the transaction is to be completed then a new authorisation must be sought using the SALE action. If the PREAUTH authorisation could not be successfully voided, then this will result in the funds' being authorised twice effectively putting two holds on the amount on the Cardholder's account and thus requiring twice the amount to be available in the Cardholder's account. It is therefore recommended only to PREAUTH small amounts, such as £1.00 to check mainly account validity.

1.7.4 REFUND_SALE

This will create a new transaction and attempt to seek authorisation for a refund of a previous SALE from the Acquirer. The transaction will then be captured and settled if and when appropriate. It can only be performed on transactions that have been successfully settled. Up until that point, a CANCEL or partial CAPTURE can be refunded or partially refunded from the original SALE transaction. The previous SALE transaction should be specified using the **`xref`** field.

Partial refunds are allowed by specifying the **`amount`** to refund. Any amount must not be greater than the original received amount minus any already refunded amount. Multiple partial refunds may be made while there is still a portion of the originally received amount un-refunded.

The **`captureDelay`** field can be used to state whether the transaction should be authorised only and settled at a later date. For more details on delayed capture refer to appendix A-10.

This action is not supported by the Hosted Integration.

1.7.5 REFUND

This will create a new transaction and attempt to seek authorisation for a refund from the Acquirer. The transaction will then be captured and settled if and when appropriate. This is an independent refund and need not be related to any previous SALE. The amount is therefore not limited by any original received amount.

The **`captureDelay`** field can be used to state whether the transaction should be authorised only and settled at a later date. For more details on delayed capture refer to appendix A-10.

This action is not supported by the Hosted Integration.

1.7.6 CAPTURE

This will capture an existing transaction, identified using the **`xref`** request field, making it available for settlement at the next available opportunity. It can only be performed on transactions that have been authorised but not yet captured. An **`amount`** to capture may be specified but must not exceed the original amount authorised.

The original transaction must have been submitted with a **`captureDelay`** value that prevented immediate capture and settlement leaving the transaction in an authorised but un-captured state. For more details on delayed capture refer to appendix A-10.

This action is not supported by the Hosted Integration.

1.7.7 CANCEL

This will cancel an existing transaction, identified using the **`xref`** request field, preventing it from being settled. It can only be performed on transactions, which have been authorised but not yet settled, and it is not reversible. Depending on the Acquirer it may not reverse the authorisation and release any reserved funds on the Cardholder's account. In such cases, authorisation will be left to expire as normal releasing the reserved funds. This may take up to 30 days from the date of authorisation.

This action is not supported by the Hosted Integration.

1.7.8 QUERY

This will query an existing transaction, identified using the **`xref`** request field, returning the original response. This is a simple transaction lookup action.

This action is not supported by the Hosted Integration.

blink

2 New Transactions

You can perform a new transaction, such as a sale, by sending a request with the required action and transaction type together with details about the order and payment method.

2.1 Request Fields

Field Name	Mandatory?	Description
`merchantID`	Yes	Your Gateway Merchant ID.
`merchantPwd`	No[1]	Any password used to secure this account. Refer to section 1.6.1 for details.
`signature`	Yes[2]	Any hash used to sign this request. Refer to section 1.6.2 for details.
`action`	Yes	The action requested. Refer to section 1.7 for supported actions. Possible values are: **PREAUTH, VERIFY, SALE, REFUND, REFUND_SALE.**
`amount`	Yes[3]	The amount of the transaction.
`Type`	Yes[3]	The type of transaction. Refer to appendix A-15 for details. Possible values are: **1** – E-commerce (ECOM) **2** – Mail Order/Telephone Order (MOTO). **9** – Continuous Authority (CA).
`countryCode`	Yes[3]	Merchant's location.
`currencyCode`	Yes[3]	Transaction currency.
`paymentMethod`	No	The payment method required. For card payments either omit this field or use the value **card**.
`cardNumber`	Yes[3,4]	The primary account number (PAN) as printed on the front of the payment card. Digits and spaces only.
`cardExpiryMonth`	Yes[3,4]	Payment card's expiry month from 1 to 12.
`cardExpiryYear`	Yes[3,4]	Payment card's expiry year from 00 to 99.
`cardExpiryDate`	No[3,4]	Payment card's expiry date in MMYY format as an alternative to sending a separate `cardExpiryMonth` & `cardExpiryYear`.
`cardCVV`	Yes[3,4]	Payment card's security number. The 3-digit number printed on the signature strip.

Field Name	Mandatory?	Description
transactionUnique	No[3]	You can supply a unique identifier for this transaction. This is an added security feature to combat transaction spoofing.
orderRef	No[3]	Free format text field to store order details, reference numbers, etc. for the Merchant's records.
orderDate	No	Optional date to record with the transaction.
captureDelay	No	Number of days to wait between authorisation of a payment and subsequent settlement. Refer to appendix A-10 for details.
xref	No[5]	Reference to a previous transaction. Refer to appendix A-16 for details.
redirectURL	No[6]	A public URL which the hosted form will redirect the Customer's browser after the transaction has been completed. The URL must be fully qualified and include at least the scheme and host components. Refer to section 1.5.6 for details.
callbackURL	No[6]	A non-public URL which will receive a copy of the transaction result by POST. The URL must be fully qualified and include at least the scheme and host components. Refer to section 1.5.7 for details.
remoteAddress	No[7]	IP address of client making the transaction. This should be provided where possible to aid fraud prevention.

[1] A password is not recommended if using the Hosted Integration, use a signature instead.
[2] A signature is recommended if using the Hosted Integration.
[3] Optional if an **xref** is provided as the value will be taken from the cross-referenced transaction.
[4] Optional if using the Hosted Integration, any value provided will be used to initialise any HPP input field.
[5] Mandatory for a REFUND_SALE request to specify the original SALE transaction.
[6] Mandatory for Hosted Integration. Not supported by the Batch Integration.
[7] Not supported by the Hosted Integration, which will automatically use the Customer's IP address.

If the REFUND_SALE action is used, then the request may not attempt to change the payment details, or the request will fail with a **responseCode** of **65542 (REQUEST MISMATCH)** because the refund must be made to the original card.

blink

2.2 Response Fields

The response will contain all the fields sent in the request (minus any **cardNumber** and **cardCVV**) plus the following:

Field Name	Returned?	Description
responseCode	Always	A numeric code providing the specific outcome. Common values are: **0** - Successful / authorised transaction. **1** - Card referred – Refer to card issuer. **2** - Card referred – Special condition. **4** - Card declined – Keep card. **5** - Card declined. Check **responseMessage** for more details of any error that occurred.
responseStatus	Always	A numeric code providing the outcome category. Possible values are: **0** – Authorisation Approved / No reason to decline **1** – Authorisation Declined. **2** – Authorisation Error / Transaction malformed.
responseMessage	Always	Message received from the Acquiring bank, or any error message.
transactionID	Always	A unique ID assigned by the Gateway.
xref	Always	You may store the cross reference for repeat transactions. Refer to appendix A-16 for details.
state	Always	Transaction state. Refer to appendix A-14.2 for details.
timestamp	Always	Time the transaction was created or last modified.
transactionUnique	If supplied	Any value supplied in the initial request.
authorisationCode	On success	Authorisation code received from Acquirer.
referralPhone	If provided	Telephone number supplied by Acquirer to phone for voice authorisation when provided.
amountReceived	On success	Amount the Acquirer authorised. This should always be the full **amount** requested.
amountRefunded	If refund	Total amount of original SALE that has so far been refunded. Returned when **action** is REFUND_SALE.
orderRef	If supplied	Any value supplied in the initial request.
cardNumberMask	Always	Card number masked for Merchant storage.
cardTypeCode	Always	Code identifying the type of card used.

Field Name	Returned?	Description
		Refer to appendix A-11 for details.
`cardType`	Always	Description of the type of card used. Refer to appendix A-11 for details.
`cardSchemeCode`	Always	Code identifying the Card Scheme used. Refer to appendix A-11 for details.
`cardScheme`	Always	Description of the card scheme used. Refer to appendix A-11 for details.
`cardIssuer`	Always	Card Issues name (when known).
`cardIssuerCountry`	Always	Card issuing country's name (when known).
`cardIssuerCountryCode`	Always	Card issuing country's ISO-3166 2-letter code (when known).
`acquirerResponseCode`	Conditional	Response code supplied by the Acquirer, maybe prefixed with 'G:' if the Acquirer is itself a payment Gateway.
`acquirerResponseMessage`	Conditional	Response message supplied the Acquirer.
`acquirerResponseDetails`	Conditional	Details about the Acquirer response containing any error messages and codes. This can be used together with the normal `responseCode`/`responseMessage` response fields to further determine the reason for any failure.
`acquirerTransactionID`	Conditional	Transaction identifier/reference used to identify the transaction in the Acquirer's system.

Other response fields may be returned as documented elsewhere in this guide. Undocumented fields may be returned at the Gateways discretion but should not be relied upon.

The **acquirerResponse*XXXX*** fields are dependent on the Acquirer in use and are supplied for additional information only.

The response is also POSTed to any URL provided by optional `callbackURL`.

3 Management Requests

You can perform a management action on an existing transaction, such as a capture or cancellation, by sending a request with the required action together with the cross reference for the transaction to act on.

Management request are supported by the Direct and Batch Integrations, they are not supported by the Hosted Integration.

3.1 Request Fields

Field Name	Mandatory?	Description
`merchantID`	Yes	Your Gateway Merchant ID.
`merchantPwd`	No[1]	Any password used to secure this account. Refer to section 1.6.1 for details.
`signature`	Yes[2]	Any hash used to sign this request. Refer to section 1.6.2 for details.
`action`	Yes	The action requested. Refer to section 1.7 for supported actions. Possible values are: **AUTHORISE**, **CAPTURE**, **CANCEL**, **QUERY**.
`xref`	Yes	Reference to a previous transaction. Refer to appendix A-16 for details.
`amount`	No[3]	The amount to capture or refund.
`callbackURL`	No	A non-public URL which will receive a copy of the transaction result by POST. The URL must be fully qualified and include at least the scheme and host components. Refer to section 1.5.7 for details.

[1] A password is not recommended if using the Hosted Integration, use a signature instead.
[2] A signature is mandatory if using the Hosted Integration.
[3] An amount is only required for partial refunds or partial captures.

3.2 Response Fields

Apart from the fields below, the response will be the same as for a new transaction but will contain the details of the existing transaction.

Field Name	Returned?	Description
responseCode	Always	A numeric code providing the outcome of the management request. Check **responseMessage** for more details of any error that occurred.
responseStatus	Always	A numeric code providing the outcome category. Possible values are: **0** – Authorisation Approved / No reason to decline **1** – Authorisation Declined. **2** – Authorisation Error / Transaction malformed.
responseMessage	Always	Description of above response code.
action	Always	The requested action and original action separated by a colon. For example. **CANCEL:SALE**

Other response fields may be returned as documented elsewhere in this guide. Undocumented fields may be returned at the Gateways discretion but should not be relied upon.

The response is also POSTed to any URL provided by optional **callbackURL**.

.

4 Hosted Payment Page Options

You can customise the appearance of the Hosted Payment Page by sending additional fields in the request. Not all fields may be supported if you have a customised Hosted Payment Page.

Note: Use /hosted for Hosted Form v2 and /payment form for Hosted Form v2.

4.1 Request Fields

Field Name	Mandatory?	Description
`cardNumber`	No[1]	Default value for the Card number field.
`cardCVV`	No[2]	Default value for the Card security number field.
`cardExpiryMonth`	No	Default value for the Card expiry month field.
`cardExpiryYear`	No	Default value for the Card expiry year field.
`cardExpiryDate`	No	Alternative to `cardExpiryMonth`/`cardExpiryYear`
`customerName`	No	Default value for the Cardholder's name field.
`customerAddress`	No	Default value for the Cardholder's address field.
`customerPostcode`	No	Default value for the Cardholder's postcode field.
`customerEmail`	No	Default value for the Cardholder's email field.
`customerPhone`	No	Default value for the Cardholder's phone number field.
`cardCVVMandatory`	No	Force a Card security number to be entered.
`customerAddressMandatory`	No	Force a Cardholder's address to be entered.
`customerPostcodeMandatory`	No	Force a Cardholder's postcode to be entered.
`customerEmailMandatory`	No	Force a Cardholder's email address to be entered.
`customerPhoneMandatory`	No	Force a Cardholder's phone number to be entered.
`formAmountEditable`	No	Enables an editable amount to be entered.
`formResponsive`	No	Request the Hosted Payment Page adjust its layout according to the browser display size etc. Possible values are: **N** – Set to standard mode. **Y** – Set to responsive mode.
`formAllowCancel`	No	Request the Hosted Payment Page show a cancel button to allow the payment to be cancelled resulting in a transaction `responseCode` of 65576 **(REQUEST CANCELLED)**.

paymentMethod	No	Request the Hosted Payment Page invoke an alternative payment method on display without the need for the Customer to select it.
allowedPaymentMethods	No	Comma separated list of **paymentMethod**s supported by the Merchant to show on Hosted Payment Page where supported.

[1] This should only be used by Merchants who are storing Card numbers as per PCI DSS requirements.
[2] This should only be used for test purposed as Merchants are not allowed to store Card CVV numbers.

5 AVS/CV2 Checking

5.1 Background

AVS and CV2 fraud checking is available on all card transactions processed by the Gateway where supported by the Acquirer.

These fraud prevention checks are performed by the Acquirer while authorising the transaction. You can choose how to act on the outcome of the check (or even to ignore them altogether).

5.1.1 AVS Checking

The Address Verification System (AVS) uses the address details that are provided by the Cardholder to verify that the address is registered to the card being used. The address and postcode are checked separately.

5.1.2 CV2 Checking

CV2, CVV, or Card Verification Value is a 3-digit or 4-digit security code. The check verifies that the code is the correct one for the card used.

For most cards, the CVV is a 3 digit number to the right of the signature strip. For American Express cards, this is a 4 digit number printed, not embossed, on the front right of the card.

The AVS/CV2 checking preferences can be configured per Merchant Account within the Merchant Management System (MMS). These preferences can be overridden per transaction by sending one of the preference fields documented in section 5.3 that hold a comma separated list of the check responses that should be allowed in order to continue to completion. Responses not in the list will result in the transaction being declined with a `responseCode` of **5 (AVS/CV2 DECLINED)**.

AVS/CV2 fraud checking is not available with every Acquirer and must be enabled on your Merchant Account before it can be used. Please contact support to find out whether your Acquirer supports it and if it can be enabled on your Merchant Account.

5.2 Benefits and Limitations

5.2.1 Benefits

Can be enabled with just a few extra integration fields.

The results are available immediately and returned as part of the transaction.

The checks can be managed independently, allowing you the utmost control over how the results are used.

The checks can be configured to decline a transaction automatically, where required.

There are no extra costs for using AVS/CV2 checking with your transactions.

Fully configurable within the Merchant Management System (MMS).

5.2.2 Limitations

The AVS checks are mainly supported by Visa, MasterCard and American Express in the USA, Canada and United Kingdom. Cardholders with a bank that does not support the checks might receive declines due to the lack of data.

Because AVS only verifies the numeric portion of the address and postcode, certain anomalies such as apartment numbers and house names can cause false declines.

The checks are meant for consumer cards. Company cards are not fully supported due to the Acquirers' not having access to this information.

5.3 Request Fields

These fields should be sent in addition to basic request fields in section 2.1.

Field Name	Mandatory?	Description
`customerAddress`	Yes[1]	For AVS checking, this must be a registered billing address for the card.
`customerPostCode`	Yes[2]	For AVS checking, this must be a registered postcode for the card.
`cardCVV`	Yes[3]	For CVV checking, this must be the Card Verification Value printed on the card.
`avscv2CheckRequired`	No[4]	Is AVS/CV2 checking required for this transaction? Possible values are: **N** – Checking is not required. **Y** – Abort if checking is not enabled.
`cv2CheckPref`	No[4]	List of `cv2Check` response values that are to be accepted; any other value will cause the transaction to be declined. Value is a comma separated list containing one or more of the following**: not known, not checked, matched, not matched, partially matched**.
`addressCheckPref`	No[4]	List of `addressCheck` values that are to be accepted; any other value will cause the transaction to be declined. Value is a comma separated list containing one or more of the following: **not known, not checked, matched, not matched, partially matched**.
`postcodeCheckPref`	No[4]	List of `postcodeCheck` response values that are to be accepted; any other value will cause the transaction to be declined. Value is a comma separated list containing one or more of the following: **not known, not checked, matched, not matched, partially matched**.

[1] Mandatory if AVS address checking is required.
[2] Mandatory if AVS postcode checking is required.
[3] Mandatory if CV2 checking is required.
[4] Overrides any Merchant Account setting configured via the Merchant Management System (MMS).

5.4 Response Fields

These fields will be returned in addition to the AVS/CV2 request fields in section 5.3 and the basic response fields in section 2.2.

Field Name	Returned?	Description
`avscv2CheckEnabled`	Always	Is AVS/CV2 checking enabled for this Merchant Account? Possible values are: **N** – Merchant account is not enabled. **Y** – Merchant account is enabled.
`avscv2ResponseCode`	If checks performed	The result of the AVS/CV2 check. Refer to appendix A-2 for details.
`avscv2ResponseMessage`	If checks performed	The message received from the Acquiring bank, or any error message with regards to the AVS/CV2 check. Refer to appendix A-2 for details.
`avscv2AuthEntity`	If checks performed	Textual description of the AVS/CV2 authorising entity as described in appendix A-2. Possible values are: **not known, merchant host, acquirer host, card scheme, issuer.**
`cv2Check`	If checks performed	Description of the AVS/CV2 CV2 check as described in appendix A-2. Possible values are**: not known, not checked, matched, not matched, partially matched**.
`addressCheck`	If checks performed	Description of the AVS/CV2 address check as described in appendix A-2. Possible values are**: not known, not checked, matched, not matched, partially matched**.
`postcodeCheck`	If checks performed	Description of the AVS/CV2 postcode check as described in appendix A-2. Possible values are**: not known, not checked, matched, not matched, partially matched**.

6 3-D Secure Authentication

6.1 Background

3-D Secure authentication is an additional fraud prevention scheme that is on all ecommerce card transactions processed by the Gateway, where supported by the Acquirer.

It allows the Cardholder to assign a password to their card that is then verified whenever a transaction is processed through a site that supports the use of the scheme. The addition of password protection allows extra security on transactions that are processed online.

3-D Secure stands for Three Domain Server. There are 3 parties that are involved in the 3-D Secure process:

The company from which the purchase is being made.
The Acquiring Bank (the bank of the company)
VISA and Mastercard (the card issuers themselves)

The gateway supports 3-D Secure as implemented by Visa, Mastercard and American Express and marketed under the brand names of Verified by VISA (VBV), Mastercard Secure Code (MSC) and American Express (SafeKey). Implementations by JCB (J/Secure) and DCI (ProtectBuy) are not currently supported.

3-D Secure is also the only fraud prevention scheme available that offers you liability cover for transactions that are verified by the checks. This provides additional protection for transactions using the scheme as distinct from those that do not.

The 3-D Secure preferences can be configured per Merchant Account within the Merchant Management System (MMS). These preferences can be overridden per transaction by sending one of the preference fields documented in section 6.5.1, which hold a comma separated list of the check responses that should be allowed to continue to completion. Responses not in the list will result in the transaction being declined with a **`responseCode`** of **65803 (3DS_NOT_AUTHENTICATED)**.

It is important that the correct Merchant Category Code (MCC) is used to avoid transactions falling back to 3-D Secure v1.

The Gateway supports both 3-D Secure version 1.0.2 and version 2.1.0 and will use the highest version available. Version 2.2.0 will be supported in the future.

3-D Secure is not available with all Acquirers and must be enabled on your Merchant Account before it can be used. Please contact support to find out whether your Acquirer supports it and if it can be enabled on your Merchant Account.

3-D Secure is supported by the Hosted and Direct Integrations. It is not supported by the Batch Integration.

6.2 Benefits and Limitations

6.2.1 Benefits

The results are available immediately and returned as part of the transaction.

The checks can be managed independently allowing you the utmost control over how the results are used.

The checks can be configured to decline the transaction automatically, where required.

If authentication is completed, liability for any subsequent fraud-related chargeback on that transaction shifts from you to the card issuer (but note the limitations below and check your Acquirer's Terms and Conditions fully on this point).

There are no extra Gateway costs for using 3-D Secure. Your Acquirer may charge to add this onto your business account; however you may also find that your transaction charges are lower as a result of using 3-D Secure.

Fully configurable within the Merchant Management System (MMS).

6.2.2 Limitations

Authenticated 3-D Secure transactions do not guarantee a liability shift and chargebacks can still occur. This is decided at the discretion of your Acquirer, with whom you should check its policy.

The gateway does not support 3-D Secure for JCB or Diner's club cards.

3-D Secure transactions require a browser in order to display the Customer authentication dialog.

6.3 Hosted Implementation

If your Merchant Account is set up for 3-D Secure, the Hosted Payment Page will automatically attempt to display the 3-D Secure authentication page for the Customer's bank.

The 3-D Secure authentication form is designed and controlled by the Customer's Issuing bank, but you can change the Merchant name and website address that is displayed on the form by sending the **`merchantName`** and/or **`merchantWebsite`** request fields.

Any **`merchantWebsite`** must be a fully qualified URL containing at least the scheme and host components.

6.4 Direct Implementation

If your Merchant account is set up for 3-D Secure, the Gateway will require further authentication details provided by the 3-D Secure system.

6.4.1 Initial Request (Verify Enrolment)

If no 3-D Secure authentication details are provided in the initial request, the Gateway will determine if the transaction is eligible for 3-D Secure by checking whether the card is enrolled in the 3-D Secure scheme.

If the Gateway determines that the transaction is not eligible for 3-D Secure, then it will continue to process it as a normal transaction without 3-D Secure, unless the **`threeDSRequired`** request field indicates that the transaction should be aborted instead.

To support 3-D Secure, you must pass the **threeDSRedirectURL** field in the initial request. This field must contain the complete URL to a web page on your server that the Access Control Server (ACS) will HTTP POST the authentication results back to, when the authentication has been completed.

For 3-D Secure v2 you must also provide details about the Cardholder's device, as documented in section 6.5.4. You may also pass additional information about the transaction and Cardholder, using the **threeDSOptions** field as also documented in section 6.5.4. This extra information can be sent to help facilitate fraud checks by the ACS.

If the Gateway determines that the transaction is eligible, it will respond with a **`responseCode`** of **65802 (3DS AUTHENTICATION REQUIRED)** and included in the response will be a **`threeDSReqest`** containing data that should be sent to the ACS at the **`threeDSURL`** using a HTTP POST request. The response will also contain a **`threeDSRef`** that can be used to continue the transaction when the authentication has been completed.

6.4.2 Continuation Request (Check AuthenticationandAuthorise)

On completion of the 3-D Secure authentication the ACS will send the challenge results to the **threeDSRedirectURL** provided in the initial request using a HTTP POST request. The contents of this POST request should be return to the Gateway unmodified in the **threeDSResponse** field together with the **threeDSRef** received in the initial response. This new request will check the authentication results and either response with the details for a further challenge, send the transaction to the Acquirer for approval or abort the transaction depending on the authentication result and your preferences, either sent in the **`threeDSPref`** field or set in the Merchant Management System (MMS).

If you would like an example of a 3-D Secure integration, please refer to our sample code appendix **A-22.2.**

6.4.3 Multiple Challenges and Frictionless Flow

The API supports the issuing of multiple challenges where the continuation request may indicate the requirement to perform another challenge by responding with a **`responseCode`** of **65802 (3DS AUTHENTICATION REQUIRED)** and including a further **`threeDSReqest`**, **`threeDSURL`** and **`threeDSRef`**. When this happens, these further challenge details should be treated the same as the first and POSTed to the ACS.

For 3-D Secure version 1, a single challenge is performed. However, for version 2, there may be zero, one or two challenges.

With 3-D Secure version 2, an initial device fingerprinting method might have to be invoked on the ACS, the results of which are used to determine whether the Cardholder must complete a challenge or whether a frictionless flow can be achieved where the transaction can continue unchallenged.

6.4.4 Cardholder Challenge

The Cardholder challenge takes place with the Cardholder's browser, usually within an IFRAME embedded on the payment form. To start the challenge, the IFRAME should contain a HTML FORM with hidden INPUT fields storing the **threeDSRequest** name/value data. JavaScript should then be used to submit the form automatically, causing the form data to be sent via a HTTP POST to the **threeDSURL**.

The IFRAME should be of sufficient size to display the ACS challenge form. For 3-D Secure version 1, this is a minimum size of 390x400 pixels. However, version 2 allows different sizes to be specified giving the Merchant more flexibility in the design of its payment form. The required size can be set using the 'challengeWindowSize' option, passed in the **threeDSOptions** field in the initial request.

6.4.5 Device Fingerprinting Challenge

The device fingerprinting method invocation is handled in the same way as a normal Cardholder challenge, except that it can be done silently in a hidden IFRAME, invisible to the normal payment flow. This silent device fingerprinting method request can be determined by the presence of a **threeDSMethodData** element in the **threeDSRequest** data. This method should take no longer than 10 seconds and therefore if the ACS has not POSTed the results back within 10 seconds then the browser can stop waiting and the transaction can be continued as normal but the **threeDSResponse** field should be returned indicating the timeout by including a **threeDSMethodData** element with the value of 'timeout'.

6.4.6 External Authentication Request

You can choose to obtain the 3-D Secure authentication details from a third-party, in which case they should provide them as part of a standard request. If the Gateway receives valid third-party authentication details, then it will use those and not attempt to contact the 3-D Secure system itself.

6.5 Request Fields

6.5.1 Initial Request (Hosted and Direct Integration)

These fields should be sent in addition to basic request fields in section 2.1.

Field Name	Mandatory?	Description
merchantName	No[1]	Merchant name to use on 3DS form.
merchantWebsite	No[1]	Merchant website to use on 3DS form. The website must be a fully qualified URL and include at least the scheme and host components.
threeDSRequired	No[1]	Is 3DS required for this transaction? Possible values are: **N** – 3DS is not required. **Y** – Abort if 3DS is not enabled.
threeDSCheckPref	No[1]	List of **threeDSCheck** response values that are to be accepted, any other value will cause the transaction to be declined. Value is a comma separated list containing one or more of the following values: 'not known', 'not **checked', ' not authenticated', 'attempted** authentication', 'authenticated' .
threeDSRedirectURL	Yes	A URL on the Merchant's server to which the ACS can POST the challenge results, thus redirecting the challenge IFRAME to this page.
threeDSOptions	No	Further 3-D Secure options that can be used by the ACS for advance fraud checking.

[1] Overrides any Merchant Account setting configured via the Merchant Management System (MMS).

6.5.2 Continuation Request (Direct Integration)

These fields may be sent alone[1].

Field Name	Mandatory?	Description
`threeDSRef`	Yes	The value of the `threeDSRef` field in the initial Gateway response.
`threeDSResponse`	Yes	The data POSTed back from the ACS when the challenge has completed.

[1] Note: It is only necessary to send the **threeDSRef** and the **threeDSResponse** in the continuation request, because the **threeDSRef** will identify the Merchant Account and initial request. The message does not need to be signed. However, you can send any of the normal request fields to modify or supplement the initial request. Any card details and transaction amount sent in the second request must match those used in the first request, else the second request will fail with a **responseCode** of **64442 (REQUEST MISMATCH)**.

blink

6.5.3 External Authentication Request (Direct Integration)

These fields should be sent in addition to basic request fields from section 2.1.

Field Name	Mandatory?	Description
threeDSEnrolled	If 3DS enabled	The 3DS enrolment status for the credit card. Refer to appendix A-3 for details. Possible values are: **Y** – Enrolled. **N** - Not Enrolled. **U** - Unable to Verify. (v1 only) **E** – Error check enrolment. (v1 only)
threeDSAuthenticated	If 3DS enrolled	The 3DS authentication status for the credit card. Refer to appendix A-3 for details. Possible values are: **Y** - Authentication Successful. **N** - Not Authenticated. **U** - Unable to Authenticate. **A** - Attempted Authentication. **R** – Authentication rejected by Issuer. (v2 only) **C** – Challenge required. (v2 only) **D** – Decoupled challenge required. (v2 only) **I** – Challenge preference acknowledged. (v2 only) **E** – Error checking authentication.
threeDSXID	If 3DS authenticated	The unique identifier for the transaction in the 3DS system.
threeDSECI	If 3DS authenticated	The Electronic Commerce Indicator (ECI).
threeDSCAVV	If 3DS authenticated	The Cardholder Authentication Verification Value (CAVV).

Note: If 3-D Secure is not enabled for the Merchant Account, then any 3-D Secure authentication fields sent in the request are ignored and the transaction is processed as normal without 3-D Secure.

6.5.4 3-D Secure 2 Options (Hosted and Direct Integration)

The following options may be sent in the **threeDSOptions** field to help customise the 3-D Secure 2 experience or can be used by the ACS for advance fraud checking. There are currently no options supported for 3-D Secure 1.

Some options are automatically initialised by the Gateway from other standard integration fields as shown in square brackets in the options description. The standard integration field should be used rather than the option, apart from the very rare circumstances where the two must have different values.

The field may be sent as a URL encoded string, JSON encoded string or an array of key/value pairs.

Field Name	Description
`accountAgeIndicator`	Cardholder Account Age Indicator. The length of time that the cardholder has had the account with the 3DS Requestor. Possible values are: 01 – No account (guest check-out) 02 – Created during this transaction 03 – Less than 30 days 04 – 30-60 days 05 – More than 60 days
`accountChangeDate`	Cardholder Account Change Date. The date that the cardholder's account with the 3DS Requestor was last changed. Accepted date format is YYYYMMDD.
`accountChangeIndicator`	Cardholder Account Change Indicator. Length of time since the cardholder's account information with the 3DS Requestor was last changed. Possible values are: 01 – Changed during this transaction. 02 – Less than 30 days 03 – 30-60 days 04 – More than 60 days
`accountDate`	Date Cardholder account opened with the 3DS Requestor. Accepted date format is YYYYMMDD.
`accountDayTransactions`	Number of account transactions in the last day. Number of transactions (successful and abandoned) for this cardholder account with the 3DS Requestor across all payment accounts in the previous 24 hours.
`accountId`	Cardholder Account Identifier. Additional information about the account optionally provided by the 3DS Requestor in AReq messages.
`accountPasswordChangeDate`	Cardholder Account Password Change Date. Date that cardholder's account with the 3DS Requestor had a

	password change or account reset. Accepted date format is YYYYMMDD.
accountPasswordChangeIndicator	Cardholder Account Password Change Indicator. Indicates the length of time since the cardholder's account with the 3DS Requestor had a password change or account reset. Possible values are: 01 – No change 02 – Changed during this transaction 03 – Less than 30 days 04 – 30-60 days 05 – More than 60 days
accountPurchaseCount	Cardholder Account Purchase Count. Number of purchases with this cardholder account during the previous six months.
accountProvisioningAttempts	Number of account provisioning attempts in the last day. Number of Add Card attempts for the account in the last 24 hours.
accountType	Indicates the type of account
accountYearTransactions	Number of account transactions in the last year. Number of transactions (successful and abandoned) for this cardholder account with the 3DS Requestor across all payment accounts in the previous year.
acquirerCountryCode	Acquirer country code when the Acquirer country differs from the Merchant country and the Acquirer is in the EEA (this could mean that the transaction is covered by PSD2). (Default on file)
acquirerBIN	Acquiring institution identification code (Default on file)
acquirerMerchantID	Acquirer-assigned merchant identifier (Default on file)
acsChallengeMandatedIndicator	ACS Challenge Mandated Indicator. Indication of whether a challenge is required for the transaction to be authorized due to local/regional mandates or other variable. Required in ARes messages if TransactionStatus = C.
addressMatch	Shipping and Billing addresses are the same. This field is used to indicate to the ACS whether the cardholder shipping address and billing address are the same. Possible values are: Y – Shipping address matched billing address. N – Shipping address does not match billing address.
authenticationECI	Value to be passed in the authorisation message
authenticationIndicator	Indicates the type of authentication request. Possible values are:

	01 – Payment – default 02 – Recurring 03 – Installment 04 – Add Card 05 – Maintain Card 06 – Verify Cardholder 07 – Billing Agreement
billingAddressCity	The city of the address. Maximum length 50 characters. [**customerTown**]
billingAddressCountryCode	The country of the address. The format is a 3 digit country code. [**customerCountryCode**]
billingAddressLine1	The first line of the street address or equivalent local portion of the address. Maximum length 50 characters. [**customerAddress**]
billingAddressLine2	The second line of the street address or equivalent local portion of the address. Maximum length 50 characters.
billingAddressLine3	The third line of the street address or equivalent local portion of the address. Maximum length 50 characters.
billingAddressPostcode	The ZIP or other postcode of the address. Maximum length is 16 characters. [**customerPostcode**]
billingAddressState	The state or province of the address. Maximum length 3 characters.
browserAcceptHeader	HTTP accept header sent from the Cardholder's browser [**browserAcceptContent**]
browserIPAddress	IP address of the Cardholder's browser [**remoteAddress**]
browserJavaEnabledVal	Ability of the Cardholder's browser to execute Java. Possible values are: jeNotPresent (0) – Not Present jeTrue (1) – True jeFalse (2) – False [**browserCapabilities**]
browserJavaScriptEnabled	Ability of the Cardholder's browser to execute JavaScript. Possible values are: bjeNotPresent (0) – Not Present bjeTrue (1) – True bjeFalse (2) – False [**browserCapabilities**]

browserLanguage	The Cardholder's browser language [**browserAcceptLanguage**]
browserScreenColorDepth	The screen colour depth of the Cardholder's browser [**browserScreenResolution**]
browserScreenHeight	The screen height of the Cardholder's browser [**browserScreenResolution**]
browserScreenWidth	The screen width of the Cardholder's browser [**browserScreenResolution**]
browserTimeZone	The timezone offset of the Cardholder's browser [**browserTimeZone**]
browserUserAgent	The User-Agent provided by the Cardholder's browser [**browserUserAgent**]
cardholderEmail	The Cardholder's email address [**customerEmail**]
cardholderHomePhone	The Cardholder's home phone number. Phone numbers must be specified in the following format: CountryCode-Subscriber (e.g. 1-1234567899). The "-" is used to separate the "Country Code" and "Subscriber" sections. **[customerPhone]**
cardholderMobilePhone	The Cardholder's mobile phone number. Phone numbers must be specified in the following format: CountryCode-Subscriber (e.g. 1-1234567899). The "-" is used to separate the "Country Code" and "Subscriber" sections. [**customerMobile**]
cardholderName	Name of the Cardholder [**customerName**]
cardholderWorkPhone	The Cardholder's work phone number. Phone numbers must be specified in the following format: CountryCode-Subscriber (e.g. 1-1234567899). The "-" is used to separate the "Country Code" and "Subscriber" sections.
challengeWindowSize	Challenge window size. Preconfigured sizes are width x height in pixels of the window displayed in the cardholder browser. Possible values are: 1 250 x 400 2 390 x 400 3 500 x 600 4 4 600 x 400 5 Full screen

deliveryEmailAddress	Merchandise Delivery Email Address [**deliveryEmail**]
deliveryTimeframe	Merchandise Delivery Timeframe. Possible values are: 01 - Electronic Delivery 02 - Same day shipping 03 - Overnight shipping 04 - Two-day or more shipping
giftCardAmount	Total gift card(s) amount
giftCardCount	Total number of gift cards purchased
giftCardCurrencyCode	Gift Card Currency
installmentPaymentData	Max authorisations permitted for installment payments
merchantCategoryCode	Merchant category code [**merchantCategoryCode**]
merchantCountryCode	Country code of the merchant [**countryCode**]
merchantFraudRate	Merchant fraud rate in the EEA (all EEA card fraud divided by EEA card volumes) calculated as per PSD2 RTS. This value is sent to Mastercard only who will not calculate or validate the fraud score: Value will be a numeric value, between 1 and 99, representing the fraud rate, such as: 1 (less than or equal to 1 basis point [bp], which is 0.01%) 2 (between 1 bp +- and 6 bps) 3 (between 6bps +- and 13 bps) 4 (between 13 bps +- and 25 bps) 5 (greater than 25 bps)
merchantName	Merchant name [**merchantName**]
paymentAccountAge	Payment Account Age. Date that the payment account was enrolled in the cardholder's account with the 3DS Requestor. Accepted date format is YYYYMMDD.
paymentAccountAgeIndicator	Payment Account Age Indicator. Indicates the length of time that the payment account was enrolled in the cardholder's account with the 3DS Requestor. Possible values are: 01 - No account (guest check-out) 02 -Created during this transaction 03 - Less than 30 days 04 - 30-60 days 05 - More than 60 days
preOrderDate	Expected date pre-ordered purchase will be available. Accepted date format is YYYYMMDD.

preOrderPurchaseIndicator	Pre-Order Purchase Indicator. Indicates whether Cardholder is placing an order for merchandise with a future availability or release date. Possible values are: 01 - Merchandise available 02 - Future availability
priorAuthData	3DS Requestor Prior Transaction Authentication Data. Part of the optional 3DS Requestor Prior Transaction Authentication Information that contains information about a 3DS cardholder authentication that occurred prior to the current transaction.
priorAuthMethod	3DS Requestor Prior Transaction Authentication Method. Mechanism used by the Cardholder to previously authenticate to the 3DS Requestor. Possible values are: 01 - Frictionless authentication occurred by ACS 02 - Cardholder challenge occurred by ACS 03 - ACS verified 04 - Other issuer methods 05-79 - Reserved for future EMVCo use 80-99 - Reserved for DS use
priorAuthTimestamp	3DS Requestor Prior Transaction Authentication Timestamp. Date and time in UTC of the prior cardholder authentication. Accepted date format is YYYYMMDDHHMM.
priorReference	3DS Requestor Prior Transaction Reference. This data element provides additional information to the ACS to determine the best approach for handling a request. It contains an ACS Transaction ID for a prior authenticated transaction (for example, the first recurring transaction that was authenticated with the cardholder).
recurringExpDate	Recurring expiration date. This field contains the date after which no further authorisations shall be performed. The format of this field must be YYYYMMDD.
recurringFrequency	The number of days between recurring payments
reorderItemsIndicator	Reorder Items Indicator. Indicates whether the cardholder is reordering previously purchased merchandise. Possible values are: 01 - First time ordered 02 - Reordered
reqAuthData	3DS Requestor Authentication Data. Data that documents and supports a specific authentication process. In the current version of the specification, this data element is not defined in detail, however the intention is that for each 3DS Requestor Authentication Method, this field carry data that the ACS can use to verify the authentication process.

reqAuthMethod	3DS Requestor Authentication Method. Method used by the Cardholder to authenticate to the 3DS Requestor. Possible values are: 01 – No 3DS Requestor authentication occurred (i.e. cardholder 'logged in' as guest) 02 - Login to the cardholder account at the 3DS Requestor system using 3DS Requestor's own credentials 03 – Login to the cardholder account at the 3DS Requestor system using federated ID 04 – Login to the cardholder account at the 3DS Requestor system using the issuer credentials 05 – Login to the cardholder account at the 3DS Requestor system using third-party authentication 06 – Login to the cardholder account at the 3DS Requestor system using FIDO Authenticator 07-79 Reserved for EMVCo future use 80-89 Reserved for future DS use
reqAuthTimestamp	3DS Requestor Authentication Timestamp. Date and time in UTC of the cardholder authentication. Accepted date format is YYYYMMDDHHMM.
requestorChallengeIndicator	3DS Requestor Challenge Indicator. Indicates whether a challenge is requested for this transaction. Possible values are: 01 – No Preference 02 – No challenge required 03 – Challenge required: 3DS Requestor Preference 04 – Challenge requested: Mandate 05 – No challenge requested (transactional risk analysis is already performed). Valid for version 2.2.0 only. 06 – No challenge requested (data share only). Valid for version 2.2.0 only. 07 – No challenge requested (strong customer authentication is already performed). Valid for version 2.2.0 only. 08 – No challenge requested (utilise whitelist exemption if no challenge required). Valid for version 2.2.0 only. 09 – Challenge requested (whitelist prompt requested if challenge required). Valid for version 2.2.0 only. 10-79 – Reserved for EMVCo future use 80-89 Reserved for future DS use If not provided, the ACS action would be identical to 01 (no preference).
requestorID	Directory server assigned 3DS Requestor identifier (Default on file)
requestorName	Directory server assigned 3DS Requestor name *(Default on file)*
requestorURL	3DS Requestor website or customer care site [**merchantWebsite**]

secureCorporatePaymentIndicator	Indicates dedicated payment processes and procedures were used, potential secure corporate payment exemption applies.
serverOperatorID	3DS Server identifier *(Default on file)*
serverRefNumber	Assigned server reference number. *(Default on file)*
shipAddressUsageDate	Shipping address first usage date. Date when the shipping address used for this transaction was first used with the 3DS Requestor. Accepted date format is YYYYMMDD.
shipAddressUsageIndicator	Shipping address usage indicator. Indicates the length of time since the shipping address used for this transaction was first used with the 3DS Requestor. Possible values are: 01 This transaction 02 Less than 30 days 03 30-60 days 04 More than 60 days
shipIndicator	Shipping method indicator. Indicates shipping method chosen for the transaction. Merchants must choose the Shipping Indicator code that most accurately describes the cardholder's specific transaction, not their general business. If one or more items are included in the sale, the Shipping Indicator code for the physical goods is used, or if all digital goods, the Shipping Indicator code that describes the most expensive item. Possible values are: 01 – Ship to cardholders billing address 02 – Ship to another verified address on file with merchant 03 – Ship to address that is different than the cardholder's billing address 04 – "Ship to Store"/Pick-up at local store (Store address shall be populated in shipping address fields) 05 – Digital goods (includes online services, electronic gift cards and redemption codes) 06 – Travel and Event tickets, not shipped 07 – Other (for example, Gaming, digital services not shipped, emedia subscriptions, etc.)
shipNameIndicator	Shipping Name Indicator. Indicates if the Cardholder Name on the account is identical to the shipping Name used for this transaction. Possible values are: 01 – Account Name identical to shipping Name 02 – Account Name different than shipping Name
shippingAddressCity	The city of the address. The maximum length is 50 characters. [**deliveryTown**]
shippingAddressCountryCode	The country of the address. The format is a 3 digit country code. [**deliveryCountryCode**]

shippingAddressLine1	The first line of the street address or equivalent local portion of the address. The maximum length is 50 characters. [**deliveryAddress**]
shippingAddressLine2	The second line of the street address or equivalent local portion of the address. The maximum length is 50 characters.
shippingAddressLine3	The third line of the street address or equivalent local portion of the address. The maximum length is 50 characters.
shippingAddressPostcode	The ZIP or other postcode of the address. The maximum length is 16 characters. [**deliveryPostcode**]
shippingAddressState	The state or province of the address. The maximum length is 3 characters and should be the country subdivision code.
suspiciousAccountActivity	Suspicious account activity indicator. Indicates whether the 3DS Requestor has experienced suspicious activity (including previous fraud) on the cardholder account. Possible values are: 01 – No suspicious activity has been observed 02 – Suspicious activity has been observed
transactionType	Transaction Type. Identifies the type of transaction being authenticated. This field is required in AReq messages in some markets (e.g. for Merchants in Brazil). Otherwise, optional. Possible values are: 01 Goods/Services Purchase (Default) 02 Check Acceptance 10 Account Funding 11 Quasi-Cash Transaction 28 Prepaid Activation and Load
whitelistStatus	Whitelist Status. Enables the communication of trusted beneficiary/whitelist status between the ACS, the DS and the 3DS Requestor. Possible values are: 01 3DS Server 02 DS 03 ACS

6.6 Response Fields

6.6.1 Initial Response (Direct Integration)

These fields will be returned in addition to the request fields from section 6.5.1 and the basic response fields in section 2.2.

Field Name	Returned?	Description
threeDSEnabled	Always	Is 3DS enabled for this Merchant Account? Possible values are: **N** – Merchant Account is not enabled. **Y** – Merchant Account is enabled.
threeDSXID	If 3DS enabled	The unique identifier for the transaction in the 3DS system.
threeDSVETimestamp	If 3DS enabled	The time the card was checked for 3DS enrolment and any initial challenge determined.
threeDSEnrolled	If 3DS enabled	The 3DS enrolment status for the credit card. Refer to appendix A-3 for details. Possible values are: **Y** – Enrolled. **N** - Not Enrolled. **U** - Unable to Verify. (v1 only) **E** - Error Verifying Enrolment.(v1 only)
threeDSRef	If 3DS enabled	Value to return in the continuation request.
threeDSURL	If 3DS enrolled	The URL of the ACS to which the challenge data should be sent via a HTTP POST request from the Cardholder's browser.
threeDSRequest	If 3DS enrolled	The challenge data that should be sent to the ACS via HTTP POST request from the Cardholder's browser.

6.6.2 Continuation Response (Direct Integration)

These fields will be returned in addition to the request fields from section 6.5.1; the initial response fields in section 6.6.1; and the basic response fields in section 2.2.

Field Name	Returned?	Description
threeDSResponse	If 3DS enrolled	The data POSTed back from the ACS when the challenge has completed.
threeDSCATimestamp	If 3DS enrolled	The time the last challenge was checked.
threeDSAuthenticated	If 3DS enrolled	The 3DS authentication status for the credit card. Refer to appendix A-3 for details. Possible values are: **Y** - Authentication Successful. **N** - Not Authenticated. **U** - Unable to Authenticate. **A** - Attempted Authentication. **E** - Error Checking Authentication. For 3DS version 2.2 only. **R** – Authentication rejected by Issuer. **C** – Challenge required. **D** – Decoupled challenge required. **I** – Acknowledges request not to challenge cardholder.
threeDSECI	If 3DS authenticated	This contains a two-digit Electronic Commerce Indicator (ECI) value, which is to be submitted in a credit card authorisation message. This value indicates to the processor that the Customer data in the authorisation message has been authenticated. The data contained within this property is only valid if the **threeDSAuthenticated** value is **Y** or **A**.
threeDSCAVV	If 3DS authenticated	This contains a 28-byte Base-64 encoded Cardholder Authentication Verification Value (CAVV). The data contained within this property is only valid if the **threeDSAuthenticated** value is **Y** or **A**.
threeDSErrorCode	If 3DS error	Any error response code returned by the ACS if there is an error in determining the card's 3DS status.
threeDSErrorDescription	If 3DS error	Any error response description returned by the ACS if there is an error in determining the card's 3DS status.

6.6.3 External Authentication Response (Direct Integration)

These fields will be returned in addition to the request fields from section 6.5.3 and the basic response fields in section 2.2.

Field Name	Returned?	Description
`threeDSEnabled`	Always	Is 3DS enabled for this Merchant Account? Possible values are: **N** – Merchant Account is not enabled. **Y** – Merchant Account is enabled.

Note: If 3-D Secure is not enabled for the Merchant Account, then any 3-D Secure authentication fields sent in the request are ignored and the transaction is processed as normal without 3-D Secure.

6.6.4 Cardholder Information (Hosted and Direct Integration)

In the case of a frictionless flow, the card Issuer may sometimes wish to provide a message to the Cardholder. In this case, the **threeDSResponseMessage** will start with the text 'Cardholder Info: ' and be followed by the message from the card Issuer.

7 Risk Checking

7.1 Background

The Gateway is integrated with Kount, the leading solution for digital fraud prevention.

If you have an existing account with Kount, or sign up for one, you can request that the Gateway pass your transactions to them for risk checking before they are sent to the Acquirer for authorisation.

Kount's patented fraud prevention technology combines device fingerprinting; supervised and unsupervised machine learning; a robust policy and rules engine; business intelligence tools; and a web-based case-management and investigation system.

Their team of experts can help you understand and identify the rules necessary to optimise your protection, as well as provide ongoing support. To get the most out of your investment, you may want to dedicate an individual or a team to monitor your rules and ensure they continue to work as intended.

The risk checking preferences can be configured per Merchant Account within the Merchant Management System (MMS). These preferences can be overridden per transaction by sending new preferences as documented in section 7.4.1. You must use the Kount management portal to configure your risk parameters and thresholds.

Risk checking is an advanced feature and must be enabled on your Merchant Account before it can be used. Please contact support if you wish to have it enabled.

7.2 Benefits and Limitations

7.2.1 Benefits

The results are available immediately and returned as part of the transaction.

The checks can be managed independently, allowing you the utmost control over how the results are used.

The checks can be configured to decline the transaction automatically where required.

Leverage the ability to review transactions and decide what course of action to take.

The checks can reduce chargebacks by blocking transactions made without the Cardholder's consent that would have resulted in the Cardholder raising a chargeback to recover the fraudulent transaction amount.

Providing enhanced risk checking increases Customer confidence and thus increases the likelihood of their making a purchase.

Fully configurable within the Merchant Management System (MMS).

7.2.2 Limitations

Checking cannot prevent all fraudulent transactions and could even prevent some non-fraudulent transactions.

There are additional fees associated with having a Kount account.

You will have to spent time analysing your transactions and establishing fraud rules.

7.3 Implementation

When risk checking is required, each transaction will be sent to Kount for checking and the result of the check will be returned in the **`riskCheck`** response field with one of the following values:

not known	- the checks could not be performed due to our error
not checked	- the checks could not be performed by Kount
approve	- the transaction is not risky and should proceed
decline	- the transaction is risky and should be declined
review	- the transaction is risky but proceed with caution
escalate	- the transaction is risky but proceed with caution

The actions to take for each **`riskCheck`** response can be configured for the Merchant Account, using the Merchant Management System. Alternatively, the preferred actions can be passed with the transaction request in the **`riskCheckPref`** field. The possible actions are as follows:

continue	- continue processing as normal
authonly	- authorise only, don't capture
decline1	- decline without reason
decline2	- decline with reason
finished	- abort with reason

The **continue** action allows the transaction to continue as normal and be sent to the Acquirer for authorisation. *A **`riskCheck`** value of **approve** will always be treated as if the action was **continue,** regardless of whether the preferences say otherwise.*

The **authonly** action allows the transaction to be authorised but not automatically captured giving you time to review it and decide whether you want to take the risk and capture the transaction or assume it to be fraudulent and cancel it.

The **decline1** and **decline2** actions will cause the transaction to be declined. Both decline the transaction and return with a **responseCode** of **5** (**DECLINED**) and a **`responseMessage`** of 'DECLINED' or 'RISK DECLINED' respectively. The first action should be used if you don't wish to alert the Customer to the fact that you suspected that their transaction was fraudulent and declined it for that reason.

The finished action will abort the transaction, causing it to return with a **responseCode** of either **65857** (**RISK_CHECK_ERROR**) or **65862** (**RISK_CHECK_DECLINED**) depending on whether an error prevented the transaction from being checked by Kount, resulting in a **`riskCheck`** value of 'not known' or 'not checked'.

The **`riskCheckPref`** field can be provided in the request to override any settings configured in the Merchant Management System (MMS) for this Merchant Account. The value should be a comma separated list of *result=actions* pairs. If a result is not specified in the list, then an action of **decline1** is assumed. For example: "**decline=decline1,review=authonly,escalate=authonly**".

7.4 Request Fields

7.4.1 Request Fields

These fields should be sent in addition to basic request fields in section 2.1.

Field Name	Mandatory?	Description
`riskCheckRequired`	No[1]	Is risk checking required for this transaction? Possible values are: **N** – risk checking is not required. **Y** – risk checking is required.
`riskCheckPref`	No[1]	List of `riskCheck` response values and the action to be taken for those responses. Value is a comma separated list containing one or more of the following risk check results and associated actions: Results: **not known**, **not checked**, **approve, decline, review, escalate**. Actions: **continue**, **decline1**, **decline2**, **authonly**, **finished**.
`riskCheckOptions`	No	Record containing options used to customise the risk checking. Refer to section 7.4.2 for values.

[1] Overrides any Merchant Account setting configured via the Merchant Management System (MMS).

7.4.2 Risk Check Options

The following options may be sent in the **riskCheckOptions** field to customise the risk checking. Where possible, the options will be initialised from standard integration fields as shown in square brackets in the option's description.

Field Name	Description
IPAD	Customer's IPv4 address (**X.X.X.X**). [**remoteAddress**]
MACK	Merchants acknowledgement to ship/process the order (**Y** or **N**).
SESS	Unique Session ID[1]. Used to link to Kount's browser device data collector.
ANID	Automatic Number Identification (ANI) submitted with order.
CASH	Total cash amount in currency submitted.
ORDR	Merchant's Order Number. [**merchantOrderRef**]
UNIQ	Merchant assigned account number for Customer. [**merchantCustomerRef**]
EPOC	Date Customer account was created by merchant.
NAME	Customer's name (or name submitted with the order). [**customerName**]
GENDER	Customer's gender (**M** or **F**) [**customerGender**]
BPREMISE	Customer's billing address premises name (UK only). [**customerCompany**]
BSTREET	Customer's billing address street (UK only). [**customerStreet**, **customerAddress**]
B2A1	Customer's billing address county/state. [**customerAddress**]
B2A2	Customer's billing address county/state. [**customerAddress2**]
B2CI	Customer's billing address county/state. [**customerTown**]
B2ST	Customer's billing address county/state. [**customerCounty**]
B2PC	Customer's billing address postcode. [**customerPostcode**]
B2CC	Customer's billing address country code. [**customerCountryCode**]

Field Name	Description
EMAL	Cardholder's email address. [**customerEmail**]
B2PN	Cardholder's phone number. [**customerPhone**]
DOB	Cardholder's date of birth. [**customerDateOfBirth**, **recipientDateOfBirth**]
S2NM	Name of person receiving the delivery. [**deliveryName**]
SPREMISE	Delivery premises name (UK only). [**deliveryCompany**]
SSTREET	Delivery street address (UK only). [**deliveryStreet**, **deliveryAddress**]
S2A1	Delivery address line 1. [**deliveryAddress**]
S2A2	Deliver address line 2. [**deliveryAddress2**]
S2CI	Delivery town/city. [**deliveryTown**]
S2ST	Delivery county/state. [**deliveryCounty**]
S2PC	Delivery postcode. [**deliveryPostcode**]
S2CC	Delivery country code. [**deliveryCountryCode**]
S2EM	Delivery email address. [**deliveryEmail**]
S2PN	Phone number of delivery location. [**deliveryPhone**]
SHTP	Shipping type. [**shippingType**, **shippingMethod**]
PROD_TYPE[*XX*]	Type for the *XX*th item purchased. [**items.XX.description**]
PROD_ITEM[*XX*]	SKU for the *XX*th item purchased. [**items.XX.productCode**]
PROD_DESC[*XX*]	Description *XX*th item purchased. [**items.XX.description**]

Field Name	Description
PROD_QUANT[*XX*]	Quantity of *XX*th item purchased. [**items.XX.quantity**]
PROD_PRICE[*XX*]	Unit amount for *XX*th item purchased. [**items.XX.amount**]
UDF[*XXXX*]	User defined field *XXXX*.

[1] SESS, the unique session id, is automatically sent from the Kount Device Collector loaded in the Hosted Payment Page.

For further information on the options, refer to the Kount Integration documentation: https://kount.github.io/docs/ris-data-submission/.

The options should be passed as either a nested record or serialised record as described in section 1.5.8. The option names are case sensitive.

blink

7.5 Response Fields

These fields will be returned in addition to the risk check request fields in section 7.4 and the basic response fields in section 2.2.

Field Name	Returned?	Description
`riskCheckEnabled`	Always	Is risk checking enabled for this Merchant Account? Possible values are: **N** – Merchant account is not enabled. **Y** – Merchant account is enabled.
`riskCheck`	If checked	The result of the risk check. Possible values are: **approve** – ok, recommend proceed to authorisation. **decline** – probably fraudulent, recommend decline. **review** – possibly fraudulent, recommend review. **escalate** – possibly fraudulent, recommend review.
`riskCheckDetails`	If checked	The raw response received from Kount minus any sensitive data.
`riskCheckResponseCode`	If checked	Response code for the risk processing stage.
`riskCheckResponseMessage`	If checked	Response message for the risk processing stage.

8 Payment Facilitators

8.1 Background

If you are a Payment Facilitator (PayFac/PF) or Independent Sales Organisation (ISO), then you must send additional fields to identify yourself and your sub-merchants.

These fields must be sent with every new transaction; however, they can be cloned from an existing transaction if using an **xref** as described in appendix A-18.

Payment Facilitator support is not available with every Acquirer. Please contact support to find out if your Acquirer supports it and what fields are required.

8.2 Request Fields

Field Name	Mandatory?	Description
`facilitatorID`	Yes	Your facilitator identifier as assigned by the Scheme.
`facilitatorName`	No[1]	Your trading name as registered with the Scheme.
`isoID`	No[1]	Your ISO identifier as assigned by the Scheme.
`subMerchantID`	No[1]	Unique identifier assigned to this SubMerchant.
`merchantXXXX`	No[1]	SubMerchant details as documented in section 17.2.
`statementNarrativeX`	No[1]	Statement details as documented in section 10.1.2.

[1] Which additional fields are mandatory will depend on your Acquirer.

9 UK MCC 6012 Merchants

9.1 Background

Every Merchant Account has a category code, also known as the MCC code, attached to it. This category code identifies the market that the payment is related to, allowing issuing banks to identify what product or service is, or was, being provided.

The merchant category code 6012 is related to payments taken for financial institutions, primarily those merchants that deal with loan payments or other credit-related activities. According to Visa, this is the most fraudulent merchant category in the UK market due to compromised debit card details being used to pay or transfer balances to other cards. Acquirers are therefore unable to confirm whether a payment is genuine, despite matching the full CVV2 with AVS.

To address this situation, issuing banks have requested additional payment information to be provided with payment requests in order to verify that the cardholder is knowingly entering into a credit-related contractual agreement with the merchant.

If you are a Merchant who has been assigned the MCC 6012 you must collect the following data for the primary recipient for each UK domestic Visa or Mastercard transaction[1]:

Unique account identifier for the loan or outstanding balance funded. For example, the loan account number or the PAN (Primary Account Number) if it is a credit card balance.
Last name (family name)
Date of Birth (D.O.B)
Postcode

Primary recipients are the entities (people or organisations) that have a direct relationship with the financial institution. Also, these primary recipients have agreed to the terms and conditions of the financial institution.

The new fields are not currently mandatory. However, some Acquirers are now declining transactions that are missing this information and so we recommend the information is always provided, even if your Acquirer doesn't currently mandate them.

If you are not a UK MCC 6012 Merchant or the payment is not a UK domestic one, then you need not provide these additional authentication details though the Gateway will accept them if you do.

[1] The additional details are currently only required by Visa and Mastercard however we recommend sending for all card types in order to be prepared for when other card Schemes follow suite.

9.2 Request Fields

To comply with the rules, an MCC6012 Merchant must send these additional fields:

Field Name	Mandatory?	Description
`merchantCategoryCode`	Yes[1]	Merchant's VISA MCC (should be 6012).
`receiverName`	Yes	Surname only - up to 6 letters allowed.
`receiverAccountNo`	Yes	Account number. If a PAN is supplied only the first 6 and last 4 digits will be used.
`receiverDateOfBirth`	Yes	Primary recipient's date of birth.
`receiverPostcode`	Yes	Primary recipient's postcode. (Only the district is required but full postcodes are accepted, therefore 'W12 8QT' or just 'W12' are acceptable values).

[1] Only required if the Merchants Category Code is not configured on their gateway account.

10 Billing Descriptor

10.1 Background

The Billing Descriptor is how your details appear on the Cardholder's statement. It is set up with the Acquirer when the Merchant Account is opened. It is used by the Cardholder to identify who a payment was made to on a particular transaction.

Selecting a clear Billing Descriptor is important for you to avoid a chargeback when the Cardholder does not recognise the name on the transaction.

10.1.1 Static Descriptor

The Static Descriptor is the descriptor agreed between yourself and your Acquirer when the Merchant Account is opened. The descriptor used is typically your trading name, location and contact phone number.

10.1.2 Dynamic Descriptor

The Dynamic Descriptor is a descriptor sent with the transaction that includes details on the goods purchased or service provided, this is often used by large companies that provide many services and where the brand of the service is more familiar than the company name. The Dynamic Descriptor usually replaces any Static Descriptor on a per transaction basis.

Not all Acquirers accept Dynamic Descriptors and, for those that do, the required format varies. Often, your Merchant name is shortened to three (3) letters, followed by an asterisk (*), followed by a short description of the service or product that the business provides. This field typically has a limit of twenty-five (25) characters including the phone number.

For more information on whether your Acquirer allows Dynamic Descriptor and the format in which they should be sent, please contact customer support.

blink

10.2 Request Fields

The Dynamic Descriptor is built using one or more of the following narrative fields.

Field Name	Mandatory?	Description
`statementNarrative1`	No	Merchant's name.
`statementNarrative2`	No	Product, service or other descriptive info.

11 Surcharges

11.1 Background

Surcharges are an additional charge that you can apply to the transactions that are processed through your Merchant Account.

Transactions that are sent for authorisation are subject to processing charges from your Acquirer and surcharges enable you to pass the processing charges that you incur on to your Customers.

You may, for example, be charged a fixed amount for debit card transactions and a percentage amount for credit card transactions. Consequently, the Gateway gives you the option to add both a fixed amount and percentage amount when applying a surcharge.

Surcharges should only be added to cover the processing charges that are incurred by your business. There is no Gateway imposed limit to the value of the surcharges that can be added to your transaction, although there are legal requirements. As a rule, the surcharge must not exceed the processing costs that you pay.

Some businesses apply surcharges to cover the costs that they incur; while others use the surcharges to subsidise the charges.

Surcharge amounts may be limited or illegal in your jurisdiction. For example, surcharging is illegal in the European Union and many US states. It is up to you to check with your Acquirer and comply with any laws.

Surcharges is an advanced feature and must be enabled on your Merchant Account before it can be used. Please contact support if you wish to have it enabled.

11.2 Implementation

11.2.1 Surcharge Rules

The **surchargeRules** field allows you to provide multiple rules specifying what surcharges should be applied to a transaction. If a transaction matches multiple rules, then the most specific rule will be used; or the first in the list.

Each surcharge rule contains the following fields:

Field Name	Mandatory?	Description
`cardType`	Yes	One or more 2-letter card type codes for which this rule applies (see) The following two card type codes are also supported, in addition to the codes listed in appendix A-11: **CC** – matches any credit card. **DD** – matches any debit card.
`currency`	No	Zero or more 3-letter ISO-4217 currency codes.
`surcharge`	Yes	Surcharge amount in minor (N) or major (N.N) units or a percentage (N%).

The surcharge rules should be passed in a sequential array of records, either as nested records or serialised records as described in section 1.5.8. The record field names are case sensitive.

11.2.2 Surcharge Amounts

The Gateway doesn't usually validate that any **amount** and **grossAmount** fields are the same and that any **netAmount**, **taxAmount** and **taxRate** tally. However, in order to update them when a surcharge is applied, the amount and **grossAmount** must match and the correct **taxRate** must be provided or be able to be calculated from one or more of the other fields. Failure in this respect can cause the Gateway to return one of the following **responseCode** values; **66360** (**INVALID_GROSSAMOUNT**), **66361** (**INVALID_NETAMOUNT**), **66338** (**INVALID_TAXAMOUNT**), **66362** (**INVALID_TAX_RATE**).

If the request contains a **surchargeAmount** field, then the Gateway will assume that surcharging has already been performed externally and will not attempt to apply any further surcharges.

blink

11.3 Request Fields

Field Name	Mandatory?	Description
`surchargeRequired`	No[1]	Is surcharging required for this transaction? Possible values are: **N** – Surcharging is not required. **Y** – Surcharging is required.
`surchargeRules`	No[1]	Surcharge rules as documented in section 11.2.1.
`surchargeAmount`	No	Surcharge amount already added. A further surcharge will not be added.

[1] Overrides any Merchant Account setting configured via the Merchant Management System (MMS).

blink

11.4 Response Fields

These fields will be returned in addition to the Surcharge request fields in section 11.3 and the basic response fields in section 2.2.

Field Name	Returned?	Description
`surchargeEnabled`	Always	Is surcharging enabled on this Merchant Account?
`surchargeAmount`	Always	Surcharge amount added.
`amount`	Always	Original request value with additional surcharge.
`grossAmount`	Conditional	Original request value adjusted for new `amount`.
`netAmount`	Conditional	Original request value adjusted for new `amount`.
`taxAmount`	Conditional	Original request value adjusted for new `amount`.

12 Receipts and Notifications

12.1 Background

The Gateway can be configured to email transaction receipts automatically to the Customer and notifications to the Merchant.

12.1.1 Customer Email Receipts

The Customer can be emailed a transaction receipt automatically each time a transaction is processed by the Gateway. Receipts are sent at the time the transaction is authorised and only for transactions where the Acquirer has approved the authorisation. Receipts are not sent for declined or referred authorisations or aborted transactions.

This functionality is enabled globally on a per Merchant Account basis using the Merchant Management System (MMS). This global setting can also be overridden per transaction if required, using the **`customerReceiptsRequired`** field.

Customer receipts require the Customer to provide an email address; if no email address is provided using the **`customerEmail`** field, then no receipt will be sent.

12.1.2 Merchant Email Notifications

You can be automatically emailed a transaction notification each time a transaction is processed by the Gateway. Notifications are sent at the time the transaction is authorised and only for transactions where the Acquirer approved, declined or referred the authorisation. Notifications are not sent for aborted transactions.

This functionality is enabled globally on a per Merchant Account basis, using the Merchant Management System (MMS).

12.2 Request Fields

12.2.1 General Fields

Field Name	Mandatory?	Description
`customerReceiptsRequired`	No[1]	Send a Customer receipt if possible. Possible values are: **N** – Don't send a receipt. **Y** – Send if Customer's email provided.
`customerEmail`	No	Customer's email address.
`notifyEmail`	No	Merchant's notification email address.

[1] Overrides any Merchant Account setting configured via the Merchant Management System (MMS).

12.3 Response Fields

The request fields for the required receipts and notifications are returned together with the appropriate fields from the following:

Field Name	Returned?	Description
`customerReceiptsResponseCode`	If required	Result of sending email to Customer.
`customerReceiptsResponseMessage`	If required	Description of above response code.
`notifyEmailResponseCode`	If required	Result of sending email to Merchant.
`notifyEmailResponseMessage`	If required	Description of above response code.

13 Recurring Transaction Agreements

13.1 Background

A Recurring Transaction Agreement (RTA) is used to request that the Gateway should perform repeat payments on your behalf, using pre-agreed amounts and schedules.

An RTA can be configured easily and quickly, using the Merchant Management System (MMS). An RTA can also be set up while performing the initial transaction request, by including the integration request fields described in section 13.3. The RTA is only set up in the transaction results in a successful payment authorisation.

The transaction should be either SALE or VERIFY transaction and the **rtAgreementType** field should be provided to indicate the type of Continuous Authority/Repeat Billing agreed between you and your Customer. This will dictate whether the subsequent repeat transactions are taken as part of a CPA agreement or as just standard MOTO transactions.

Merchants who use this system to implement billing or subscription type payments are encouraged to use Continuous Authority (CA) transactions to comply with Card Payment Scheme practices. *Your Acquirer may refuse to accept the recurring transactions if they are not subject to an agreement between yourself and your Customer.*

Please note that email receipts are not sent for Recurring Transactions, but this can be overridden in the direct integration code.

Refer to appendix A-17 for more information on the different types of repeat or recurring transactions.

13.2 Scheduling

There are two different types of scheduling available when requesting the Gateway to take recurring transactions automatically on the Merchant's behalf. In addition, a start date can be provided to allow for a recurring subscription with an initial free trial period.

13.2.1 Fixed Scheduling

Fixed scheduling causes the subsequent transaction to be taken at fixed intervals of time and for fixed amounts. A different initial date and amount or final date and amount can be provided for use when the agreed payment term or amount doesn't exactly divide by the fixed time intervals.

Fixed scheduling is specified by providing an **rtScheduleType** field with a value of 'fixed' and providing the **rtCycleDuration**, **rtCycleDuration** and **rtCycleCount** fields to define the interval at which transactions should be taken and the number of transactions to take.

An **rtCycleCount** field value of 0 can be provided to indicate that transactions should be taken ad-infinitum until the RTA is stopped.

13.2.2 Variable Scheduling

Variable scheduling causes the subsequent transaction to be taken on prespecified dates and for prespecified amounts.

Variable scheduling is specified by providing an **rtScheduleType** field with a value of 'variable' and providing the **rtSchedule** field with a value containing an array of one or more schedule records.

Each schedule record must contain the following fields:

Field Name	Mandatory?	Description
`date`	Yes	Date on which to take a payment.
`amount`	Yes	Amount to take on the provided date.

The schedule records should be passed in a sequential array of records, either as nested records or serialised records as described in section 1.5.8. The record field names are case sensitive.

13.3 Request Fields

Field Name	Mandatory?	Description
`rtName`	No	Free format short name for the agreement.
`rtDescription`	No	Free format longer description for the agreement.
`rtPolicyRef`	No	Merchant Reference (MPRN).
`rtAgreementType`	No	Recurring transaction agreement type. Indicates the type of Continuous Payment Authority or Repeat Billing agreement made with the Cardholder. Possible values are: <not provided> - no CPA agreed. **recurring** – recurring type CPA agreed. **instalment** – instalment type CPA agreed.
`rtMerchantID`	No	Merchant ID to use for the recurring transactions (defaults to `merchantID`).
`rtStartDate`	No	Start date of agreement (defaults to date received).
`rtScheduleType`	No	Schedule type. Possible values are: **fixed** – fixed interval schedule (default). **variable** – variable interval schedule.
`rtSchedule`	Yes[1]	Nested array or serialised string containing payment schedule information as per section 13.2.2.
`rtInitialDate`	No[2]	Date of initial payment (defaults to `rtStartDate`).
`rtInitialAmount`	No[2]	Amount of initial payment (defaults to `rtCycleAmount`).
`rtFinalDate`	No[2]	Date of final payment.
`rtFinalAmount`	No[2]	Amount of final payment (defaults to `rtCycleAmount`).
`rtCycleAmount`	No[2]	Amount per cycle (defaults to `amount`).
`rtCycleDuration`	Yes[2]	Length of each cycle in `rtCycleDurationUnit` units.
`rtCycleDurationUnit`	Yes[2]	Cycle duration unit. One of: **day**, **week**, **month** or **year**.
`rtCycleCount`	Yes[2]	Number of cycles to repeat (zero to repeat forever).
`rtMerchantData`	No	Free format Merchant data field.
`rtSequenceCount`	No	Total number of recurring/instalments including initial transaction.

[1] For use with variable schedules only.

[2] For use with fixed schedules only.

blink

13.4 Response Fields

Field Name	Returned?	Description
`rtID`	Always	Recurring Transaction Agreement ID.
`rtResponseCode`	Always	Result of setting up RT Agreement.
`rtResponseMessage`	Always	Description of above response code.

blink

14 Duplicate Transaction Checking

14.1 Background

Duplicate transaction checking prevents transaction requests from accidentally processing more than once. This can happen if a Customer refreshes your checkout page or clicks a button that issues a new transaction request repeatedly in short succession. While duplicate checking can help prevent repeat transactions from going through, we recommend talking with your developers to see whether changes can be made to your form to reduce the likelihood of this occurring (e.g. disabling the Submit button after it has been clicked).

14.2 Implementation

To help prevent duplicate transactions, each transaction can specify a time window during which previous transactions will be checked to see whether they could be possible duplicates.

This time window is specified using the **`duplicateDelay`** field. The value for this field can range from 0 to 9999 seconds (approximately 2 ¾ hours).

If the transaction request does not include the **`duplicateDelay`** field or specifies a value of zero, then a default delay of 300 seconds (5 minutes) is used.

The following fields are used in transaction comparison and must be the same for a transaction to be regarded as a duplicate:

- **`merchantID`**
- **`action`**
- **`type`**
- **`amount`**
- **`transactionUnique`**
- **`currencyCode`**
- **`xref`** (if provided in lieu of card details)
- **`cardNumber`** (may be specified indirectly via cross reference)

If a transaction is regarded as being a duplicate, then a **`responseCode`** of **65554 (REQUEST DUPLICATE**) will be returned.

14.3 Request Fields

Field Name	Mandatory?	Description
`duplicateDelay`	No	Duplicate transaction time window in seconds. **Numeric value between 0 and 9999.**

15 Purchase Data

15.1 Background

The Gateway can be sent advanced purchase information with each transaction, where required.

The Gateway provides several fields that you can use to store advanced purchase information about the transaction, including details on individual items purchased. These fields are only sent to the Acquirer if needed. The stored data can be obtained by sending a QUERY request.

The details may also be used for advanced purposes, such as displaying shopping cart information on the MasterPass Wallet and PayPal Checkout.

15.1.1 American Express Purchases

Purchases using American Express cards will send a subset of this information to the card scheme as appropriate.

With American Express, you can provide tax ***or*** discount reason (but not both). If **`taxAmount`** is provided, then **`taxReason`** is used; if **`discountAmount`** is provided, then **`discountReason`** is used. If both are provided, then **`taxReason`** is used.

Only the first six line item details are sent to American Express and then only the **`itemXXDescription`**, **`itemXXQuantity`** and **`itemXXGrossAmount`** fields are sent.

15.1.2 Purchase Orders

These fields together with other advanced fields, as detailed in section 16, can be used to send full information relating to a purchase order and related invoice indicating types; quantities; and agreed prices for products or services. Details on the supplier; shipping; delivery can also be included.

At present, this information is not sent to the Acquirer, unless needed, but future enhancements to the Gateway may include sending such information as Level 2 or 3 Purchasing data as defined by the relevant card schemes.

blink

15.2 Request Fields

The following request fields may be sent to provide more information on the breakdown of the purchase amount:

Field Name	Mandatory?	Description
`grossAmount`	No	Total gross amount of sale.
`netAmount`	No	Total net amount of sale.
`taxRate`	No	Total tax rate (percentage).
`taxAmount`	No[1]	Total tax amount of sale.
`taxReason`	No[1]	Reason for above tax (e.g. VAT).
`discountAmount`	No[1]	Total discount amount of sale.
`discountReason`	No[1]	Reason for above discount.
`handlingAmount`	No	Handling costs.
`insuranceAmount`	No	Insurance costs.

[1] Amex/Diners require either tax or discount, not both.

The following request fields may be sent to provide more information on the purchased items:

item*XX*Amount[1]	No	Amount for *XX*th item purchased.
item*XX*Description[1]	No	Description of *XX*th item purchased.
item*XX*Quantity[1]	No	Quantity of *XX*th item purchased.
item*XX*GrossAmount[1]	No	Gross amount for *XX*th item purchased.
item*XX*NetAmount[1]	No	Net amount for *XX*th item purchased.
item*XX*TaxAmount[1]	No	Tax amount for *XX*th item purchased.
item*XX*TaxRate[1]	No	Total tax rate for *XX*th item purchased.
item*XX*TaxReason[1]	No	Tax reason for *XX*th item purchased.
item*XX*DiscountAmount[1]	No	Total discount for *XX*th item purchased.
item*XX*DiscountReason[1]	No	Discount reason for *XX*th item purchased.
item*XX*ProductCode[1]	No	Product code for *XX*th item purchased.
itemXXProductURL[1]	No	Shopping cart URL for XXth item purchased.
item*XX*CommodityCode[1]	No	Commodity code for *XX*th item purchased.
item*XX*UnitOfMeasure[1]	No	Unit of measure for *XX*th item purchased.
item*XX*UnitAmount[1]	No	Unit amount for *XX*th item purchased.
itemXXImageUrl[1]	No	Image of XXth item purchased.
itemXXSize[1]	No	Size of XXth item purchased in the format 'LengthxWidthxHeight Unit'.
itemXXWeight[1]	No	Weight of XXth item purchased in the format 'Weight Unit'.
Items	No	Nested line item records (see below).

[1] XX is a number between 1 and 99.

The purchased items can be passed as either individual **item*XXField*** fields; or as a single **items** field whose value is a sequential array of nested records as described in section 1.5.8.

Both formats cannot be used together. The presence of an **items** field will cause the Gateway to ignore any individual fields.

The Gateway does not currently support **items** to be given as a serialised array of records.

Note: no attempt is made to check that any gross, net and tax amounts are correct with respect to each other. It is the sender's responsibility to ensure alternative amount formats are correct.

Line item fields can either be sent 'flat' using field names containing the item row number as a sequential number from 1 to 99; or be sent using nested arrays of the form **`items[XX][field]`** where ***`XX`*** is the row number from 1 to 99 and ***`field`*** is the field name from the above table without the **`item`*`XX`*** prefix and starting with a lowercase first letter. For example, the tax rate for item 5 can be sent either as **`item5TaxRate;`** or as **`items[5][taxRate]`**. The two formats should not be mixed. If a request field of **`items`** is seen, then the 'flat' fields are ignored.

16 Custom Data

You may send arbitrary data with the request by appending extra fields, which will be returned unmodified in the response. These extra fields are merely 'echoed' back and not stored by the Gateway.

Caution should be made to ensure that any extra fields do not match any currently documented fields or possible future fields. One way to do this is to prefix the field names with a value unique to you, the Merchant.

You can also use the **`merchantData`** field to store custom data with the transaction. This stored data can then be retrieved at a later date, using a QUERY request. Associative data can be serialised using the notation **`merchantData[name]=value`**; or, alternatively, a JSON or XML encoded string could be stored.

16.6 Request Fields

Field Name	Mandatory?	Description
`merchantData`	No	Arbitrary data to be stored together with this transaction.

17 Advanced Data

The Gateway provides a number of fields that you can use to store information about the transaction. These fields are only sent to the Acquirer if needed. The stored data can be obtained by sending a QUERY request.

17.1 Customer Request Fields

These fields can be used to store details about the Customer and any relationship between the Customer and Merchant such as any purchase order raised.

If AVS checks are in use, then the Customer and Cardholder are assumed to be the same person and the address and postcode fields are taken as being the registered billing address of the card.

Field Name	Mandatory?	Description
customerName	No	Cardholder's name.
customerCompany	No	Cardholder's company (if applicable).
customerAddress	No[1]	Cardholder's address.
customerPostcode	No[1]	Cardholder's postcode.
customerTown	No	Cardholder's town/city.
customerCounty	No	Cardholder's county/province.
customerCountryCode	No	Cardholder's country.
customerPhone	No	Cardholder's phone number.
customerMobile	No	Cardholder's mobile phone number.
customerFax	No	Cardholder's fax number.
customerEmail	No	Cardholder's email address.
customerOrderRef	No	Customer's reference for this order (Purchase Order Reference).
customerMerchantRef	No	Customer's reference for the Merchant.
customerTaxRef	No	Customer's tax reference number.

[1] Mandatory if AVS checking required.

blink

17.2 Merchant Request Fields

These fields can be used to store details about the Merchant and any relationship between the Merchant and Customer such as any invoice reference.

Field Name	Mandatory?	Description/Value
merchantName	No	Merchant's contact name.
merchantCompany	No	Merchant's company name.
merchantAddress	No	Merchant's contact address.
merchantTown	No	Merchant's contact town/city.
merchantCounty	No	Merchant's contact county.
merchantPostcode	No	Merchant's contact postcode.
merchantCountryCode	No	Merchant's contact country.
merchantPhone	No	Merchant's phone.
merchantMobile	No	Merchant's mobile phone number.
merchantFax	No	Merchant's fax number.
merchantEmail	No	Merchant's email address.
merchantWebsite	No	Merchant's website. The website must be a fully qualified URL and include at least the scheme and host components.
merchantOrderRef	No	Merchant's reference for this order (Invoice/Sales Reference).
merchantCustomerRef	No	Merchant's reference for the Customer.
merchantTaxRef	No	Merchant's tax reference number.
merchantOriginalOrderRef	No	Reference to a back order.
merchantCategoryCode	No	Scheme assigned Merchant Category Code (MCC).

blink

17.3 Supplier Request Fields

These fields can be used to store details about the Supplier address. This is where any purchased goods are being supplied from if different from the Merchant's address.

Field Name	Mandatory?	Description/Value
`supplierName`	No	Supplier's contact name.
`supplierCompany`	No	Supplier's company name.
`supplierAddress`	No	Supplier's contact address.
`supplierTown`	No	Supplier's contact town/city.
`supplierCounty`	No	Supplier's contact county.
`supplierPostcode`	No	Supplier's contact postcode.
`supplierCountryCode`	No	Supplier's contact country.
`supplierPhone`	No	Supplier's phone.
`supplierMobile`	No	Supplier's mobile phone number.
`supplierFax`	No	Supplier's fax number.
`supplierEmail`	No	Supplier's email address.

blink

17.4 Delivery Request Fields

These fields can be used to store details about the delivery address. This is where any purchased goods are being delivered to if different from the Customer's address.

Field Name	Mandatory?	Description/Value
`deliveryName`	No	Name of person receiving the delivery.
`deliveryCompany`	No	Name of company receiving the delivery.
`deliveryAddress`	No	Delivery address.
`deliveryTown`	No	Delivery town/city.
`deliveryCounty`	No	Delivery county.
`deliveryPostcode`	No	Delivery postcode.
`deliveryCountryCode`	No	Delivery country.
`deliveryPhone`	No	Phone number of delivery location.
`deliveryMobile`	No	Mobile phone number of delivery location.
`deliveryFax`	No	Fax number of delivery location.
`deliveryEmail`	No	Delivery email address.

blink

17.5 Receiver Request Fields

These fields can be used to store details about the recipient of the purchased goods where different from the Customer's and Delivery details. It is most commonly used by Financial Institutions (MCC 6012 Merchants) who need to record the primary recipient of a loan.

Field Name	Mandatory?	Description/Value
`receiverName`	No	Receiver's contact name.
`receiverCompany`	No	Receiver's company name.
`receiverAddress`	No	Receiver's contact address.
`receiverTown`	No	Receiver's contact town/city.
`receiverCounty`	No	Receiver's contact county.
`receiverPostcode`	No	Receiver's contact postcode.
`receiverCountryCode`	No	Receiver's contact country.
`receiverPhone`	No	Receiver's phone.
`receiverMobile`	No	Receiver's mobile phone number.
`receiverFax`	No	Receiver's fax number.
`receiverEmail`	No	Receiver's email address.
`receiverAccountNo`	No	Receiver's account number.
`receiverDateOfBirth`	No	Receiver's date of birth.

blink

17.6 Shipping Request Fields

These fields can be used to store details about the shipping method and costs.

Field Name	Mandatory?	Description/Value
`shippingTrackingRef`	No	Shipping tracking reference.
`shippingMethod`	No	Shipping method (e.g. Courier, Post, etc.).
`shippingAmount`	No	Cost of shipping.
`shippingGrossAmount`	No	Gross cost of shipping.
`shippingNetAmount`	No	Net cost of shipping.
`shippingTaxRate`	No	Tax rate as percentage to 2 decimal places.
`shippingTaxAmount`	No	Tax cost of shipping.
`shippingTaxReason`	No	Tax reason (e.g. VAT).
`shippingDiscountAmount`	No	Discount on shipping.
`shippingDiscountReason`	No	Reason for discount.

Note: No attempt is made to check that any gross, net and tax amounts are correct with respect to each other. It is the sender's responsibility to ensure alternative amount formats are correct.

blink

17.7 Device Information Fields

These fields can be used to provide details of the device from which the transaction is being made. Although not strictly mandatory, they may be required for fraud checking, in which case it is highly recommended that they be provided.

Field Name	Mandatory?	Description/Value
`deviceType`	No	Type of Consumer's device. One of the following values: **desktop**, **laptop**, **tablet**, **phone**, **other**.
`deviceChannel`	No	Communications channel used by the Consumer's device. One of the following values: **browser**, **app**, **other**.
`deviceIdentity`	No[1]	Content of the HTTP User-Agent header received from the Consumer's device. Truncated to 2048 characters maximum.
`deviceTimeZone`	No[1]	Time zone offset in minutes between UTC and the Consumer's device. The offset is positive if the local time zone is behind UTC and negative if it is ahead.
`deviceCapabilities`	No[1]	Comma separated list of capabilities supported by the Consumer's device. One or more of the following values: **java**, **javascript**.
`deviceAcceptContent`	No[1]	Content of HTTP Accept header received from the Consumer's device. Truncated to 2048 characters maximum.
`deviceAcceptCharset`	No[1]	Content of HTTP Accept-Charset header received from the Consumer's device. Truncated to 2048 characters maximum.
`deviceAcceptEncoding`	No[1]	Content of HTTP Accept-Encoding header received from the Consumer's device. Truncated to 2048 characters maximum.
`deviceAcceptLanguage`	No[1]	Content of HTTP Accept-Language header received from the Consumer's device. Truncated to 2048 characters maximum.
`deviceScreenResolution`	No[1]	Screen resolution of the Consumer's device. Formatted as [HxWxD] where: • H – screen height in pixels • W – screen width in pixels

		• D – colour depth in bits Screen height and width must be between 1 and 999999 pixels. Colour depth must be one of the following values: **1**, **4**, **8**, **15**, **16**, **24**, **32**, **48**.
`deviceOperatingSystem`	No	Operating system used by the Consumer's device. One of the following values: **win**, **unix**, **linux**, **macos**, **ios**, **android**, **other**.

[1] This field is mandatory for 3-D Secure v2 unless an alternative is provided via the **`threeDSOptions`** field.

18 Gateway Wallet

18.1 Background

The Gateway supports an internal digital Wallet that is available to all Merchants using the Gateway.

The Gateway allows you to store your Customer's payment card, billing and delivery address details and other information securely encrypted in its internal Wallet. You can then allow your Customer to select from stored payment cards to check out faster on your website.

Management of this Wallet is done using the Gateway's REST API. However, you can use the Hosted, Direct or Batch Integrations to perform transactions, using cards and addresses stored in the Wallet; or to store new cards and address used with successful transactions.

18.2 Benefits and Limitations

18.2.1 Benefits

Details can be used from or added to the Wallet with just a few extra integration fields.

Customers can select from previously stored details, making the checkout process more streamlined, resulting in fewer abandoned carts and thus increasing sales.

Compatible with existing card base fraud solutions such as Address Verification Service (AVS), 3-D Secure and third-party fraud providers.

There are no extra costs to use the internal Gateway Wallet.

The Wallet transactions are controlled within the Merchant Management System (MMS) in the same manner as normal card transactions.

18.2.2 Limitations

The payment details are stored internally by the Gateway and not available for use with other Gateway Merchants or other payment gateways.

18.3 Hosted Implementation

Customers who have payment details already saved will have the option to select from those details rather than having to renter them. Customers will also have the option to delete stored details[1]

The details are only saved if the transaction is successful, ensuring that the Wallet is not filled up with invalid payment details.

The details requiring to be stored in the Wallet are validated when the transaction is performed, prior to any authorisation with the Acquirer. If any of the details are invalid, then the transaction will be aborted with a **responseCode** of **66304** (**INVALID_REQUEST**) and a **responseMessage** indicating which data could not be stored in the Wallet. Any failure that occurs post authorisation will not abort the transaction but will be available in the appropriate ***xxxx*StoreResponseCode** response fields.

The **`walletOwnerRef`** field can be used to assign a unique Customer reference to the Wallet, allowing you to identify which of your Customers owns the Wallet. This could be the Customer reference you use within your own Customer accounts or Shopping Cart software. You must ensure that this value is less than 50 characters, or the transaction will be aborted with a **`responseCode`** of **65xxx** (**INVALID_WALLETCUSTOMERREF**).

18.4 Direct Implementation

If a transaction is sent to the Direct Integration, then with the addition of a few extra integration fields, it can be instructed to use payment details stored in the Wallet and/or store the used payment details.

Using stored payment details is similar to performing cross-referenced transactions where the payment details are cloned from a previous transaction[1]. However, in this case the payment details are taken from the Wallet and not a previous transaction.

The details are only saved if the transaction is successful, ensuring that the Wallet is not filled up with invalid payment details.

The details requiring to be stored in the Wallet are validated when the transaction is performed prior to any authorisation with the Acquirer. If any of the details are invalid, then the transaction will be aborted with a **responseCode** of **66304** (**INVALID_REQUEST**) and a **responseMessage** indicating which data could not be stored in the Wallet. Any failure that occurs post authorisation will not abort the transaction but will be available in the appropriate ***xxxx*StoreResponseCode** response fields.

The **`walletOwnerRef`** field can be used to assign a unique Customer reference to the Wallet allowing you to identify which of your Customers owns the Wallet. This could be the Customer reference you use within your own Customer accounts or Shopping Cart software. You must ensure that this value is less than 50 characters, or the transaction will be aborted with a **`responseCode`** of **65xxx** (**INVALID_WALLETCUSTOMERREF**).

blink

18.5 Request Fields

Field Name	Mandatory?	Description
`walletID`	No	Identifier for an existing Wallet to use.
`walletName`	No	Name for any new Wallet created.
`walletDescription`	No	Description for any new Wallet created.
`walletOwnerRef`	No	Owner Reference for any new Wallet created.
`walletData`	No	Merchant Data for any new Wallet created.
`walletStore`	No	Request that all payment details be stored in the Wallet. A new Wallet will be created if needed. Possible values are: **Y**- store all payment details. **N**- store details according to their `xxxStore` value.
`cardID`	No	Identifier for an existing card stored in a Wallet.
`cardName`	No	Name for any new card stored.
`cardDescription`	No	Description for any new card stored.
`cardData`	No	Merchant Data for any new card stored.
`cardStore`	No	Request that the payment card details be stored in the Wallet. A new Wallet will be created if needed. Possible values are: **Y**- store the card details. **N**- do not store the card details.
`customerAddressID`	No	Identifier for an existing address stored in a Wallet.
`customerAddressName`	No	Name for any new address stored.
`customerAddressDescription`	No	Description for any new address stored.
`customerAddressData`	No	Merchant Data for any new address stored.
`customerAddressStore`	No	Request that the customer address details be stored in the Wallet. A new Wallet will be created if needed. Possible values are: **Y**- store the customer address details. **N**- do not store the customer address details.
`deliveryAddressID`	No	Identifier for an existing address stored in a Wallet.
`deliveryAddressName`	No	Name for any new address stored.
`deliveryAddressDescription`	No	Description for any new address stored.

deliveryAddressData	No	Merchant Data for any new address stored.
deliveryAddressStore	No	Request that the delivery address details be stored in the Wallet. A new Wallet will be created if needed. Possible values are: **Y**- store the delivery address details. **N**- do not store the delivery address details.

blink

18.6 Response Fields

These fields will be returned in addition to the request fields from section.

Field Name	Mandatory?	Description
`walletStoreResponseCode`	No	Result of creating or updating the Wallet details. Refer to appendix A-1 for details.
`walletStoreResponseMessage`	No	Description of above response code.
`cardStoreResponseCode`	No	Result of creating or updating the card details. Refer to appendix A-1 for details.
`cardStoreResponseMessage`	No	Description of above response code.
`customerAddresStoreResponseCode`	No	Result of creating or updating the address details. Refer to appendix A-1 for details.
`customerAddressStoreResponseMessage`	No	Description of above response code.
`deliveryAddressStoreResponseCode`	No	Result of creating or updating the address details. Refer to appendix A-1 for details.
`deliveryAddressStoreResponseMessage`	No	Description of above response code.

If new items are stored in the Wallet, then their identifiers will be returned in the appropriate **walletID**, **cardID**, **customerAddressID** and **deliveryAddressID** together with any values provided for or assigned by default to the other item fields.

Failure to store any of the details in the Wallet will be reported using the appropriate ***xxxx*StoreResponseCode** response field.

19 Masterpass Wallet

19.1 Background

Masterpass is a digital wallet from Mastercard that is available to all Merchants using the Gateway.

It allows customers to store their payment and shipping information in one central, secure location. With Masterpass, customers can shop, click, and check out faster on your website.

Masterpass transactions process and settle just like credit card transactions. You can identify Masterpass transactions in the Merchant Management System by their unique payment type logo, which includes the credit card brand name at the bottom.

There are no additional fees for processing Masterpass transactions – pricing for Masterpass is the same as your other credit card transactions.

Masterpass versions 6 and 7 are supported by the Gateway.

Masterpass has upgraded to Mastercard's new guest checkout option and Customers can no longer sign up. As such this integration is subject to change.

Masterpass is an advanced feature and must be enabled on your Merchant Account before it can be used. Please contact support if you wish to have it enabled.

Masterpass is supported by the Hosted and Direct Integrations. It is not supported by the Batch Integration.

19.2 Benefits and Limitations

19.2.1 Benefits

The Wallet details are stored externally to the Gateway and available with any third-party Checkout that supports Masterpass.

Customers can select from previously stored details, making the checkout process more streamlined, resulting in fewer abandoned carts and thus increasing sales.

Compatible with existing card base fraud solutions such as Address Verification Service (AVS), 3-D Secure and third-party fraud providers.

There are no extra costs to add Masterpass to your Gateway account.

The Masterpass transactions are controlled within the Merchant Management System (MMS) in the same manner as normal card transactions.

19.2.2 Limitations

Your Customer will need a Masterpass Wallet with some stored card details in order to make full use of this payment method.

Repeat transactions using the retrieved payment details are supported but may require permission from Masterpass.

19.3 Hosted Implementation

If a transaction is sent to the Hosted Integration using a **merchantID** that has Masterpass enabled, then the Hosted Payment Page will display a MasterPass payment button that, when clicked, will open the Masterpass Wallet and allow the Customer to select their payment card and address details.

To customise the Masterpass Wallet experience, you may send various options in the masterPassCheckoutOptions field in your initial request.

Additional information available from the Masterpass Wallet will be made available in the masterPassCheckoutDetails response field.

Note: Custom Hosted Payment Pages might not support the displaying of the Masterpass button. If you have a custom page that doesn't support this, then please contact support to have your Hosted Payment Page upgraded.

19.4 Direct Implementation

Masterpass transactions require you to display the Masterpass Wallet to your Customer as part of the transaction flow. The transaction must be done in two stages, with the Wallet being displayed between the stages. They can optionally also be done in three stages, allowing you to display an order confirmation after the Wallet and before authorising the transaction. You can change the amount at this stage to allow for shipping costs when you know the confirmed delivery address the Customer selected from the Wallet.

19.4.1 Initial Request (Checkout Preparation)

To request that a transaction be processed using details selected from the Customer's Masterpass Wallet, the request must contain a **paymentMethod** of 'masterpass' and a **masterPassCallbackURL** containing the URL of a page on your server to return to when the Wallet is closed. In addition, you may send **masterPassCheckoutOptions** to customise the Wallet experience. When the Gateway receives these two fields, assuming there are no other errors with the request, it will attempt to find a suitable Masterpass enabled Merchant Account in the current account mapping group (refer to appendix A-6).

If the Gateway is unable to find a suitable account, then the transaction will be aborted and it will respond with a **responseCode** of **65569 (MASTERPASS_NOT_SUPPORTED)**.

Otherwise the Gateway will respond with a **`responseCode`** of **65572 (MASTERPASS_CHECKOUT REQUIRED)** and the response will include a **masterPassCheckoutURL** field containing the URL required to load the Masterpass Wallet and a **masterPassCheckoutOptions** containing any data required to be sent to the Wallet. The response will also contain a unique **masterPassData** field that must be echoed back in the continuation requests. No transaction will have been created by the Gateway at this stage and this request will not appear in the Merchant Management System.

At this point your server must redirect the Customer's browser to the Masterpass Wallet at the provided **masterPassCheckoutURL**. Alternatively, the **masterPassCheckoutURL** can be used in conjunction with the Masterpass JavaScript code to implement a lightbox style Wallet that allows the Merchants website to remain visible in the background. Further details on how to use the Masterpass JavaScript SDK can be obtained from Masterpass.

19.4.2 Continuation Request (Checkout Details and Authorise)

On completion of the Masterpass Wallet, it will redirect the Customer's browser to the **masterPassCallbackURL** provided, including an OAuth token, OAuth verifier and status URL parameters. If the checkout was successful, the status will be 'success'. Alternatively, if the checkout was cancelled the status will be 'cancel'.

These URL parameters should be sent to the Gateway in the **masterPassToken**, **masterPassVerifier** and **masterPassStatus** fields of a new request. The new request must contain the **masterPassData** received in the initial response. This new request will retrieve the Customer's chosen payment and delivery details from Masterpass and then send the transaction to the Acquirer for authorisation, returning the result similarly to a normal card-based authorisation transaction.

If the continuation request contains any details that would normally be read from the Masterpass Wallet, then these will take precedence and overwrite the Wallet details. Note: in such cases, the transaction will no longer class as being a Masterpass transaction and will be treated by the Acquirers as if the Wallet was not used.

If the chosen details cannot be retrieved or if the **masterPassStatus** field indicated that the Wallet was cancelled, then the Gateway will return a **`responseCode`** of **65570 (MASTERPASS_CHECKOUT_FAILURE).**

19.4.3 Separate Checkout Details and Authorisation Requests

You can choose to obtain the Wallet details before sending the transaction for authorisation by sending the **masterPassOnly** field in the above continuation request. If this field is sent with a value of 'Y', then the Gateway will load the Wallet details and then return them to you without sending the request for authorisation. You can then display them and/or adjust the amount, for example, according to delivery charges appropriate to the received delivery address. You should then send a new request, containing the **masterPassData** received, to continue the transaction and authorise it.

If the continuation request contains any details that would normally be read from the Masterpass Wallet, then these will be ignored and the Wallet details returned. Note: this is different from usual processing, where incoming fields usually take precedence.

The outcome of this request will depend on the value of the **masterPassStatus** field and the ability to communicate with Masterpass. On success, the Gateway will return a **`responseCode`** of **65571 (MASTERPASS_CHECKOUT_SUCCESS)** and response will include the chosen payment card and address details. If the Wallet was cancelled or if the chosen details cannot be retrieved, then the Gateway will return a **`responseCode`** of **65570 (MASTERPASS_CHECKOUT_FAILURE).**

Note: this stage can be repeated multiple times by including the **masterPassOnly** field with a value of 'Y' each time. To complete the transaction, the final request must not contain the **masterPassOnly** field or it must not have a value of 'Y'.

19.5 Request Fields

19.5.1 Initial Request (Hosted and Direct Integrations)

These fields should be sent in addition to basic request fields in section 2.1 excluding any card details.

Field Name	Mandatory?	Description
paymentMethod	Yes[1]	Must contain the value 'masterpass' in lower case letters only.
masterPassCallbackURL	Yes[2]	URL on Merchant's server to return to when the Masterpass Wallet is closed.
masterPassCheckoutOptions	No	Record containing options used to customise the Masterpass Wallet. Refer to section 20.5.3 for values.
masterPassCheckoutID	No	Merchant's unique checkout identifier as provided by Masterpass.

[1] Optional for Hosted Integration.
[2] Not required for Hosted Integration.

19.5.2 Continuation Request (Direct Integration)

Field Name	Mandatory?	Description
masterPassData	Yes	Unique reference returned in the initial response.
masterPassStatus	Yes[1]	The Wallet status returned to the Merchant.
masterPassToken	Yes[1]	The OAuth token returned to the Merchant.
masterPassVerifier	Yes[1]	The OAuth verifier returned to the Merchant.
masterPassOnly	No	Pass **Y** to complete the processing as far as the next Wallet stage and then return with the loaded Wallet details.

[1] The **masterPassStatus**, **masterPassToken** and **MasterPassVerifier** should be initialised with values received by your website when the wallet redirects to your **masterPassCallbackURL** URL. If the checkout was cancelled, then only the **masterPassStatus** field need be sent to the Gateway.

19.5.3 Wallet Options (Hosted and Direct Integrations)

The following options may be set in the **masterPassCheckoutOptions** field to customise the Masterpass Wallet.

Field Name	Description
`version`	Masterpass version required. Possible values are: **v6** – use version 6 of the Masterpass API (default). **v7** – use version 7 of the Masterpass API (preferred).
`requireLightboxCheckout`	Use the lightbox version of the Masterpass Wallet rather than the full screen checkout when possible. Possible values are: **false** – use the full screen Wallet. **true** – use the lightbox Wallet if possible.
`suppress3Ds`	Suppress Masterpass 3-D Secure processing. This allows the Gateway to do the 3-D Secure processing using the chosen card details. Possible values are: **false** – allow the Wallet to handle 3-D Secure. **true** – allow the Gateway to handle 3-D Secure.
`requestBasicCheckout`	Deprecated name for `suppress3Ds`.
`suppressShippingAddress`	Suppress the requirement for the Customer to select a shipping address as well as payment card details. Possible values are: **false** – shipping address must be selected. **true** – shipping address need not be selected.
`requestShippingAddressEnable`	Deprecated name for `suppressShippingAddress`.
`shippingLocationProfile`	Provide a Masterpass shipping profile. Refer to the Masterpass document for details.
`rewardsProgram`	Enable the Masterpass loyalty program. Refer to the Masterpass document for details.
`loyaltyEnabled`	Deprecated name for `rewardsProgram`.
`suppressWalletSelector`	Suppress the ability to select alternative Wallet providers from within the Masterpass Wallet. Possible values are: **false** – allow alternative Wallets to be selected. **true** – don't allow alternative Wallets to be selected.
`walletSelectorBypassEnable`	Deprecated name for `suppressWalletSelector`
`merchantCheckoutId`	Merchant's unique checkout identifier as provided by Masterpass. Either as passed in the initial request or as configured on the Gateway for your account.

Field Name	Description
`allowedCardTypes`	List of Masterpass card types to allow selection from within the Wallet. Will be returned in the response from the card types configured for your Merchant Account.

The options should be passed as either a nested record or serialised record as described in section 1.5.8. The option names are case sensitive.

The deprecated options were originally used with v6 of the Masterpass API and the Gateway will accept both the newer v7 option name and the original v6 option name regardless of the value of any version option provided. The Gateway may return the correct name for the version when it returns the **masterPassCheckoutOptions** in the initial response.

The nature of the URL returned in the **masterPassCheckoutURL** response field depends on whether the Masterpass lightbox or full-page redirect checkout is required as specified using the **requireLightboxCheckout** option. If the option is passed as 'true', then the URL will reference the Masterpass JavaScript file that should be loaded to provide the code required to open the lightbox style Wallet. The **masterPassCheckoutOptions** response values should then be passed to the JavaScript call to open the lightbox. If the option is not passed or not 'true', then the URL will be an address to redirect the Customer's browser to in order to display the MasterPass Checkout pages. This URL will have its query component initialised from any **masterPassCheckoutOptions** request values, and any response values need not be used.

If the **suppress3Ds** or **requestBasicCheckout** option is not passed, then it will be defaulted to 'true', so that the Gateway's 3-D Secure processing will be used as opposed to the Wallet's 3-D Secure processing. This ensures your 3-D Secure preferences are followed.

If the suppressShippingAddress **or suppressShippingAddressEnable** option is passed as 'true' then no attempt will be made to return the delivery address fields. Any delivery address fields passed in the transaction will be echoed back unaltered.

Any **merchantCheckoutId** or **allowedCardTypes** options will be overwritten and therefore should not be passed in the request but will be available in the response.

19.5.4 Purchase details (Hosted and Direct Integrations)

The following request fields may be sent to provide information on the purchased items and to populate the cart on the Masterpass Wallet (v6 only).

Field Name	Mandatory?	Description
`itemXXDescription`	No	Description of *XX*th item purchased.
`itemXXQuantity`	No	Quantity of *XX*th item purchased.
`itemXXGrossAmount`	No	Gross amount for *XX*th item purchased.
`itemXXTaxAmount`	No	Tax amount for *XX*th item purchased.
`itemXXProductCode`	No	Product code for *XX*th item purchased.
`itemXXImageUrl`	No	Shopping cart URL for *XX*th item purchased.
`itemXXSize`	No	Size of *XX*th item purchased in the format 'LengthxWidthxHeight Unit'.
`itemXXWeight`	No	Weight of XX*X*th item purchased in the format 'Weight Unit'.
`items`	No	Nested array of line items.

Refer to section 15.2 for more information on these fields.

19.6 Response Fields

19.6.1 Initial Response (Direct Integration)

These fields will be returned in addition to the request fields from section 19.5.1 and the basic response fields in section 2.2 minus any card details.

Field Name	Mandatory?	Description
`masterPassData`	Yes	Unique reference required to continue this transaction when the Masterpass Wallet has completed.
`masterPassCheckoutURL`	Yes	URL required to load the Masterpass Checkout
`masterPassCheckoutOptions`	No	Any checkout options passed in the request.

19.6.2 Continuation Response (Direct Integration)

These fields will be returned in addition to the request fields from section 19.5.2; the initial response fields in section 19.6.1; and the basic response fields in section 2.2 minus any card details.

Field Name	Mandatory?	Description
masterPassData	No	Provided, if **masterPassOnly** was used in the continuation response to indicate that a further request will be sent to finalise the transaction.
masterPassWalletID	Yes	Masterpass Wallet ID.
masterPassCheckout	Yes	Masterpass Wallet details in original retrieved XML format minus any card numbers.
card*XXXX*	Yes	Card details chosen in the Masterpass Wallet as documented in section 2.2.
customer*XXXX*	No[1]	Customer details if provided by the Masterpass Wallet as documented in section 17.1
delivery*XXXX*	No[1]	Delivery details if provided by the Masterpass Wallet as documented in section 17.4

[1] The response will include Customer/billing address and delivery address details if provided by the Masterpass Wallet.

20 PayPal Transactions

20.1 Background

PayPal is an additional payment method that is available to all Merchants using the Gateway who have a PayPal account.

To use PayPal, you will be supplied with a separate PayPal Merchant Account that can be grouped with your main Merchant Account using the account mapping facility as documented in appendix A-8. This allows transactions to be sent using your main Merchant Account and then routed automatically to the PayPal Merchant Account in the same mapping group.

It allows you to offer payment via PayPal in addition to normal card payments.

PayPal transactions will appear in the Merchant Management System (MMS) alongside any card payments and can be captured, cancelled and refunded in the same way as card payments.

PayPal transactions can also be used for recurring billing but require you to indicate in the initial transaction that it will be the basis for recurring billing and that a billing agreement will be entered into between your Customer and PayPal when they agree to the payment.

PayPal transactions cannot be used for ad-hoc ‘Card On File’ repeat transactions unless a billing agreement has been set up.

For more information on how to accept PayPal transactions, please contact customer support.

PayPal is supported by the Hosted and Direct Integrations. It is not supported by the Batch Integration.

20.2 Benefits and Limitations

20.2.1 Benefits

Provides your customers with the flexibility of paying by using their PayPal account when this is more suitable to them than using a traditional credit or debit card.

The in-context PayPal Express Checkout helps improve conversion rates with an easier way to pay without customers leaving your website.

There are no extra costs for adding a PayPal Merchant Account. However, you will still be liable for the PayPal transaction fees.

The full PayPal transaction information is available and returned as part of the transaction.

Transactions are controlled within the Merchant Management System (MMS) in the same manner as normal card transactions.

20.2.2 Limitations

You will need a PayPal account.

Recurring transactions are not supported unless as part of a prearranged billing agreement.

Independent refunds that are not tied to a previous sale transaction are not supported without prior agreement.

Transactions require a browser in order to display the PayPal Checkout.

The PayPal Checkout cannot be opened from within a browser IFRAME and so care must be taken to ensure that any PayPal Checkout button is not placed within such an IFRAME.

20.3 Hosted Implementation

If a transaction is sent to the Hosted Integration using a **merchantID** that is part of a routing group containing a PayPal Merchant, then the Hosted Payment Page will display a PayPal payment button that, when clicked, will open the PayPal Checkout and allow the Customer to pay using their PayPal account.

To customise the PayPal Checkout experience, you may send various options in the **payPalCheckoutOptions** field in your initial request.

Additional information available from the PayPal Checkout will be made available in the **checkoutDetails** response field.

Note: Custom Hosted Payment Pages might not support the displaying of the PayPal Checkout button. If you have a custom page that doesn't support this, then you would need to contact support to have your Hosted Payment Page upgraded.

20.4 Direct Implementation

PayPal transactions require you to display the PayPal Checkout to your Customer as part of the transaction flow. The transaction must be done in two stages, with the Checkout being displayed between the stages. They can also be optionally done in three stages allowing you to display an order confirmation after the Checkout and before authorising the transaction. You can change the amount at this stage to allow for shipping costs when you know the confirmed delivery address the Customer selected as part of the PayPal Checkout.

PayPal supports the normal payment and management actions. This section explains how to make payment requests. Management requests are performed as detailed in section 3.

20.4.1 Initial Request (Checkout Preparation)

To request that a transaction be processed via PayPal the request must contain a **paymentMethod** of 'paypal' and a **checkoutRedirectURL** containing the URL of a page on your server to return to when the Checkout is closed. In addition, you may send **checkoutOptions** to customise the Checkout experience. When the Gateway receives this **paymentMethod**, assuming there are no other errors with the request, it will attempt to find a suitable PayPal Merchant Account in the current account mapping group.

If the Gateway is unable to find a suitable account, then the transaction will be aborted and it will respond with a **responseCode** of **66364 (INVALID PAYMENTMETHOD)**.

Otherwise the Gateway will respond with a **`responseCode`** of **65826 (CHECKOUT REQUIRED)** and included in the response will be a **checkoutURL** field containing the URL required to load Checkout and a **checkoutRequest** containing any data required to be sent to the Checkout. The response will also contain a unique **checkoutRef** which must be echoed back in the continuation requests.

At this point your server must redirect the Customer's browser to the provided **checkoutURL**. Alternatively, the **checkoutURL** can be used in conjunction with the PayPal In-Context JavaScript code to implement an In-context Checkout which allows the Merchants website to remain visible in the background. Further details on how to use the In-Context Checkout are provided in the PayPal guide at https://developer.paypal.com/docs/classic/express-checkout/in-context/enable_in_context_checkout/.

20.4.2 Continuation Request (Checkout Details and Authorise)

On completion of the PayPal Checkout it will redirect the Customer's browser to the **checkoutRedirectURL** provided including a **token** and **status** URL parameters. If the checkout was successful, the status will be 'success' alternatively if the Checkout was cancelled the status will be 'cancel'. The received redirect request parameters inclusive of these **token** and **status** parameters should then be sent to the Gateway in the **checkoutResponse** fields of a new request. The **checkoutResponse** field can be sent either as the original URL query string received or as an array of the decoded query parameters. This new request will load the Checkout details including any delivery address if required and send the transaction to PayPal for

authorisation, returning the result as per a normal authorisation transaction. The new request must contain the **checkoutRef** received in the initial response.

20.4.3 Separate Checkout Details and Authorisation Requests

You can choose to obtain the Checkout details before actually sending the transaction for authorisation by sending the **checkoutOnly** field in the above continuation request. If this field is sent with a value of 'Y' then the Gateway will load the Checkout details and then return them to you without sending the request for authorisation. You can then display them and/or adjust the amount, for example, according to delivery charges appropriate to the received delivery address. You should then send a new request containing the **checkoutRef** received to continue the transaction and authorise it.

Note: this stage can be repeated multiple times by including the **checkoutOnly** field with a value of 'Y' each time. To complete the transaction, the final request must not contain the **checkoutOnly** field or it must not have a value of 'Y'.

20.5 Request Fields

20.5.1 Initial Request (Hosted and Direct Integrations)

These fields should be sent in addition to basic request fields in section 2.1 excluding any card details.

Field Name	Mandatory?	Description
paymentMethod	Yes[1]	Must contain the value 'paypal' in lower case letters only.
checkoutRedirectURL	Yes[2]	URL on Merchant's server to return to when the PayPal Checkout is closed.
checkoutOptions	No[3]	Record containing options used to customise the PayPal Checkout. Refer to section 20.5.3 for values.
payPalCheckoutOptions	No[4]	Record containing options used to customise the PayPal Checkout. Refer to section 20.5.3 for values.

[1] Optional for Hosted Integration.
[2] Not required for Hosted Integration.
[3] Direct Integration Only
[4] Hosted Integration Only

20.5.2 Continuation Request (Direct Integration)

These fields may be sent alone[1].

Field Name	Mandatory?	Description
checkoutRef	Yes	Unique reference return in the initial response.
checkoutResponse	Yes	The GET and or POST data received by the **checkoutRedirectURL** page.
checkoutOnly	No	Pass **Y** to complete the processing as far as the next Checkout stage and then return with the loaded Checkout details.

[1] It is only necessary to send the **checkoutRef** and the **checkoutResponse** in the continuation request because the **checkoutRef** will identify the Merchant Account and initial request. The message does not need to be signed. You can send any of the normal request fields to modify or supplement the initial request – however, in this case the request should be signed. The **checkoutRedirectURL** and **checkoutOptions** fields sent in the initial request cannot be modified and any sent in the second request must match those used in the first request, or the second request will fail with a **responseCode** of **64442** (**REQUEST MISMATCH**).

20.5.3 Checkout Options (Hosted and Direct Integrations)

The following options may be set in the **payPalCheckoutOptions** Hosted Integration field or the **checkoutOptions** Direct Integration field to customise the PayPal Checkout.

Field Name	Description
`inContext`	Use the in-context PayPal Checkout rather than the full screen Checkout when possible. Possible values are: **0** – use the full screen Checkout. **1** – use the in-context Checkout if possible.
`userAction`	Determines whether buyers complete their purchases on PayPal or on your website. Possible values are: **commit** – sets the submit button text to 'Pay Now' on the PayPal Checkout. This text lets buyers know that they complete their purchases if they click the button. **continue**– sets the submit button text to 'Continue' on the PayPal Checkout. This text lets buyers know that they will return to the Merchant's cart to complete their purchases if they click the button.
`maxAmount`[1]	The expected maximum total amount of the order, including shipping and taxes.
`reqBillingAddress`	Determines whether the buyer's billing address on file with PayPal is returned. **This feature must be enabled by PalPal**.
`reqConfirmShipping`	Determines whether the buyer's shipping address on file with PayPal must be a confirmed address. Possible values are: **0** – does not need to be confirmed **1** – must be confirmed
`noShipping`	Determines whether PayPal displays shipping address. Possible values are: **0** – display the shipping address **1** – do not display shipping address and remove shipping information **2** – If no `delivery`***`XXXX`*** fields passed, PayPal obtains them from the buyer's account profile.
`addrOverride`	Determines whether the PayPal Checkout displays the shipping address sent using the `delivery`***`XXXX`*** fields and not the shipping address on file with PayPal for this buyer. Displaying the PayPal street address on file does not allow the buyer to edit that address. Possible values are: **0** – PayPal should not display the address. **1** – PayPal should display the address.

[1] PayPal refer to this field as MAXAMT.

Field Name	Description
localeCode	Locale of the pages displayed by PayPal during Express Checkout. It is either a two-letter country code or five-character locale code supported by PayPal.
allowNote	Enables the buyer to enter a note to the merchant on the PayPal page during Checkout. The note is returned in the `checkoutDetails` response field.
pageStyle	Name of the Custom Payment Page Style used for the PayPal Checkout. It is the same name as the Page Style Name used when adding styles in the PayPal Account.
payflowColor	The HTML hex colour code for the PayPal Checkout's background colour. By default, the colour is white (FFFFFF).
cardBorderColor	The HTML hex colour code for the PayPal Checkout's principal identifying colour. The colour will be blended to white in a gradient fill that borders the cart review area.
hdrImg	URL for the image you want to appear at the top left of the payment page. The image has a maximum size of 750 pixels wide by 90 pixels high. PayPal requires that you provide an image that is stored on a secure (https) server. If you do not specify an image, the business name displays.
logoImg	A URL to your logo image. Use a valid graphics format, such as .gif, .jpg, or .png. Limit the image to 190 pixels wide by 60 pixels high. PayPal crops images that are larger. PayPal places your logo image at the top of the cart review area.
landingPage	Type of PayPal Checkout to display. Possible values are: **Billing** – Non-PayPal account **Login** – PayPal account login
channelType	Type of channel. Possible values are: **Merchant** – Non-auction seller **eBayItem** – eBay auction
solutionType	Type of Checkout flow. Possible values are: **Sole** – Buyer does not need to create a PayPal account to check out. This is referred to as PayPal Account Optional. **Mark** – Buyer must have a PayPal account to check out.
totalType	Type declaration for the label to be displayed in MiniCart for UX. Possible values are: **Total** **EstimatedTotal**
brandName	A label that overrides the business name in the PayPal account on the PayPal Checkout.

Field Name	Description
`customerServiceNumber`	Merchant Customer Service number displayed on the PayPal Checkout.
`buyerEmailOptInEnable`	Enables the buyer to provide an email address on the PayPal pages to be notified of promotions or special events. Possible values are: **0** – Do not enable buyer to provide email. **1** – Enable the buyer to provide email.
`noteToBuyer`	A note from the merchant to the buyer that will be displayed in the PayPal Checkout.
`paymentAction`	Defines how to obtain payment. This can be used to override any `captureDelay` setting that can also be used to indicate a **Sale** or **Authorisation** only. Possible values are: **Sale** – sale with immediate capture. **Authorization** – authorisation subject to later capture (note spelling). **Order** – order subject to later authorisation and capture.
`allowedPaymentMethod`	The payment method type. Specify the value InstantPaymentOnly
`insuranceOptionOffered`	Indicates whether insurance is available as an option that the buyer can choose on the PayPal Review page. Possible values are: **true** – The Insurance option displays 'Yes' and the `insuranceAmount`. If true, the total shipping insurance for this order must be a positive number. **false** – The Insurance option displays 'No'.
`multiShipping`	Indicates whether this payment is associated with multiple shipping addresses. Possible values are: **0** – Single/No shipping address. **1** – Multiple shipping addresses.
`noteText`	Note to the Merchant.
`bucketCategoryType`	The category of a payment. Possible values are: **1** – International shipping **2** – Local delivery **3** – BOPIS, Buy online pick-up in store **4** – PUDO, Pick-up drop-off
`locationType`	Type of merchant location. Required if the items purchased will not be shipped, such as, BOPIS (Buy Online Pick-up In Store) or PUDO (Pick-Up Drop-Off) transactions. Possible values are: **1** – Consumer. **2** – Store, for BOPIS transactions. **3** – PickupDropoff, for PUDO transactions.

Field Name	Description
`locationID`	Location ID specified by the merchant for BOPIS (Buy Online Pick-up In Store) or PUDO (Pick-Up Drop-Off) transactions.
`sellerPayPalAccountID`	Unique identifier for the Merchant. For parallel payments, this field is required and must contain the Payer Id or the email address of the Merchant.
`invNum`	Merchant's invoice or tracking number.
`custom`	Custom field for your own use.
`buyerID`	The unique identifier provided by eBay for this buyer. The value may or may not be the same as the username. In the case of eBay, it is different.
`buyerUsername`	The user name of the user at the marketplaces site.
`buyerRegistrationDate`	Date when the user registered with the marketplace. In UTC/GMT format, for example, 2013-08-24T05:38:48Z.
`allowPushFunding`	Indicates whether the Merchant can accept push funding. Possible values are: **0** – Merchant cannot accept push funding. **1** – Merchant can accept push funding.
`userSelectedFundingSource`	This element could be used to specify the preferred funding option for a guest user. However, the `landingPage` Checkout option must also be set to **Billing**, otherwise it is ignored. Possible values are: **ChinaUnionPay.** **CreditCard.** **ELV.** **QIWI.**
`billingType`	Type of billing agreement for reference transactions. You must have permission from PayPal to use this field. Possible values are: **MerchantInitiatedBilling** – PayPal creates a billing agreement for each transaction associated with buyer. **MerchantInitiatedBillingSingleAgreement** – PayPal creates a single billing agreement for all transactions associated with buyer. Use this value unless you need per-transaction billing agreements.
`billingAgreementDescription`	Description of goods or services associated with the billing agreement. This field is required for each recurring payment billing agreement. PayPal recommends that the description contain a brief summary of the billing agreement terms and conditions. For example, buyer is billed at "9.99 per month for 2 years".

Field Name	Description
paymentType	Type of PayPal payment you require for the billing agreement. Possible values are: **Any** – The merchant accepts any payment method for the billing agreement, even if it could take a few working days for the movement of funds to the merchant account. This includes echeck, in addition to credit or debit cards and PayPal balance. **InstantOnly** – The payment options accepted by the merchant are credit cards, debit cards or PayPal balance only because the merchant expects immediate payment.
taxIDType	Buyer's tax ID type. This field is required for Brazil and used for Brazil only. For Brazil use only: The tax ID type is BR_CPF for individuals and BR_CNPJ for businesses.
taxID	Buyer's tax ID. This field is required for Brazil and used for Brazil only. For Brazil use only: The tax ID is 11 single-byte characters for individuals and 14 single-byte characters for businesses
returnFMFDetails	Flag to indicate whether you want the results returned by Fraud Management Filters when doing a recurring/reference transaction. Possible values are: **0** – Do not receive FMF details (default). **1** – Receive FMF details.
riskSessionCorrelationID	The ID of the risk session for correlation purposes when doing a recurring/reference transaction.
merchantSessionID	The buyer session identification token when doing a recurring/reference transaction.
buttonSource[1]	PayPal Partner's BN Code (if required).

[1] The BN code is the unique button source code provided by PayPal to its partners and added by its partners to the PayPal buttons used by merchants to offer the PayPal Services that are enabled through Partner Product. The button source code provides a means of identifying and tracking referred merchants' payments.

For further information on the options, refer to the PayPal Express Checkout documentation: https://developer.paypal.com/docs/classic/api/merchant/SetExpressCheckout_API_Operation_NVP/.

The options should be passed as either a nested record or serialised record as described in section 1.5.8. The option names are case sensitive.

20.5.4 Purchase details (Hosted and Direct Integrations)

The following request fields may be sent to provide information on the purchased items and to populate the cart on the PayPal Checkout.

Field Name	Mandatory?	Description
`shippingAmount`	No	Shipping costs.
`shippingDiscountAmount`	No	Discount applied to shipping costs.
`handlingAmount`	No	Handling costs.
`insuranceAmount`	No	Insurance costs.
`itemXXDescription`	No	Description of *XX*th item purchased.
`itemXXQuantity`	No	Quantity of *XX*th item purchased.
`itemXXAmount`	No	Gross amount for *XX*th item purchased.
`itemXXTaxAmount`	No	Tax amount for *XX*th item purchased.
`itemXXProductCode`	No	Product code for *XX*th item purchased.
`itemXXProductURL`	No	Shopping cart URL for *XX*th item purchased.
`itemXXSize`	No	Size of *XX*th item purchased in the format 'LengthxWidthxHeight Unit'.
`itemXXWeight`	No	Weight of XX*X*th item purchased in the format 'Weight Unit'.
`items`	No	Nested array of line items.

Refer to section 15.2 for more information on these fields.

Note: The shopping cart items must total to the amount specified in the transaction. If they do not, cart items will not be sent to the PayPal Checkout.

20.6 Response Fields

20.6.1 Initial Response (Direct Integration)

These fields will be returned, in addition to the request fields from section 20.5.1 and the basic response fields in section 2.2 minus any card details.

Field Name	Mandatory?	Description
checkoutRef	Yes	Unique reference required to continue this transaction when the PayPal Checkout has completed.
checkoutName	Yes	Unique name of the Checkout. For PayPal this is the value **paypal**.
checkoutURL	Yes	URL required to load the PayPal Checkout
checkoutRequest	No	Not required for PayPal.
checkoutOptions	No	Any Checkout options passed in the request.
acquirerResponseDetails	Yes	Details about the PayPal response containing any error messages and codes. This can be used together with the normal **responseCode** and **responseMessage** response fields to determine further the reason for any failure.

20.6.2 Continuation Response (Direct Integration)

These fields will be returned, in addition to the request fields from section 20.5.2, the initial response fields in section 20.6.1 and the basic response fields in section 2.2 minus any card details.

Field Name	Mandatory?	Description
checkoutRef	Yes	Provided if **checkoutOnly** was used in the continuation response to indicate that a further request will be sent to finalise the transaction.
checkoutName	Yes	Unique name of the Checkout. For PayPal, this is the value **paypal**.
checkoutDetails	Yes	Record containing options used to customise the PayPal Checkout. Refer to section 20.6.3 for values.
customer*XXXX*	No[1]	Customer details if provided by the PayPal Checkout as documented in section 17.1
delivery*XXXX*	No[1]	Delivery details if provided by the PayPal Checkout as documented in section 17.4
acquirerResponseDetails	Yes	Details about the PayPal response containing any error messages and codes. This can be used together with the normal **responseCode** and **responseMessage** response fields to determine further the reason for any failure.

20.6.3 Checkout Details (Hosted and Direct Integration)

The following details may be provided in the **checkoutDetails** field included in the response.

Field Name	Mandatory?	Description
`correlationID`	No	Correlation ID, which uniquely identifies the transaction to PayPal.
`checkoutStatus`	No	Status of the Checkout session. If payment is completed, the transaction identification number of the resulting transaction is returned. Possible values are: **PaymentActionNotInitiated** **PaymentActionFailed** **PaymentActionInProgress** **PaymentActionCompleted**
`invNum`	No	Merchant's invoice or tracking number, as set sent in `checkoutDetails.invNum` or assigned by the Gateway.
`custom`	No	Merchant's invoice or tracking number, as set sent in `checkoutDetails.custom` or assigned by the Gateway.
`payPalAdjustment`	No	A discount or gift certificate offered by PayPal to the buyer. This amount is represented by a negative amount. If the buyer has a negative PayPal account balance, PayPal adds the negative balance to the transaction amount, which is represented as a positive value.
`buyerMarketingEmail`	No[1]	Buyer's marketing email address.
`note`	No[2]	Buyer's note to the Merchant.
`cartChangeTolerance`	No	Indicates whether a cart's contents can be modified. If this parameter is not returned, then assume the cart can be modified. This will return **NONE** if financing was used in Germany. Possible values are: **NONE** – The cart cannot be changed. **FLEXIBLE** – The cart can be changed.
`payerID`	No	Buyer's PayPal Customer Account ID.

1 The response will include Customer/billing address and delivery address details if provided by the PayPal Checkout.

2 Only available if the leaving of notes was enabled in the initial request using `checkoutOptions.allowNote` option.

Field Name	Mandatory?	Description
`payerStatus`	No	Buyer's PayPal status. Possible values are: **verified** **unverified**
`billingName`	No[1]	Buyer's name. Also returned in `customerName`.
`firstName`	No[2]	Buyer's first name. Also returned in `customerName`.
`middleName`	No[2]	Buyer's middle name. Also returned in `customerName`.
`lastName`	No[2]	Buyer's last name. Also returned in `customerName`.
`suffix`	No[2]	Buyer's name suffix. Also returned in `customerName`.
`business`	No	Buyer's business name. Also returned in `customerCompany`.
`street`	No	Buyer's street first line. Also returned in `customerAddress`.
`street2`	No	Buyer's street second line. Also returned in `customerAddress`.
`city`	No	Buyer's city Also returned in `customerTown`.
`state`	No	Buyer's state. Also returned in `customerCounty`.
`zip`	No	Buyer's postal code. Also returned in `customerPostcode`.
`countryCode`	No	Buyer's country code. (ISO 2 char. code) Also returned in `customerCountryCode`.
`countryName`	No	Buyer's country name.
`phoneNum`	No	Buyer's contact phone number. Also returned in `customerPhone`.
`email`	No	Buyer's email address. Also returned in `customerEmail`.

[1] Permission is needed from PayPal to support this field.
[2] These fields are used when no permission to use `billingName`.

Field Name	Mandatory?	Description
shipToName	No	Name of person/entity to ship to. Also returned in **deliveryName**.
shipToStreet	No	Ship to street first line. Also returned in **deliveryAddress**.
shipToStreet2	No	Ship to street second line. Also returned in **deliveryAddress**.
shipToCity	No	Ship to city. Also returned in **deliveryTown**.
shipToState	No	Ship to state. Also returned in **deliveryCounty**.
shipToZip	No	Ship to postal code. Also returned in **deliveryPostcode**.
shipToCountryCode	No	Ship to country code. (ISO 2 char. code) Also returned in **deliveryCountryCode**.
shipToCountryName	No	Ship to country name.
shipToPhoneNum	No	Ship to phone number. Also returned in **deliveryPhone**.
shipToAddressStatus	No	Status of shipping address on file with PayPal. Possible values are: **none** **Confirmed** **Unconfirmed**
addressNormalizationStatus	No[1]	The PayPal address normalisation status for Brazilian addresses. Possible values are: **None** **Normalized** **Unnormalized** **UserPreferred**
amount	No	Total amount for this order.
itemAmount	No	Total item amount for this order.
taxAmount	No	Tax amount for this order.
exchangeRate	No	Exchange rate for this order.
shippingAmount	No	Shipping amount for this order.
handlingAmount	No	Handling amount for this order.

[1] This field is passed directly to PayPal and therefore the field name and value must be spelt 'ize' and not 'ise'.

Field Name	Mandatory?	Description
`insuranceAmount`	No	Insurance amount for this order.
`shipDiscountAmount`	No	Shipping discount amount for this order.
`desc`	No	Description of items the buyer is purchasing.
`currencyCode`	No	ISO 3-letter currency code.
`isFinancing`	No	Indicates whether the Customer ultimately was approved for and chose to make the payment using the approved instalment credit. Possible values are: **FALSE** – financing not in use **TRUE** – financing approved and used
`financingFeeAmount`	No	The transaction financing fee associated with the payment. This will be set to the instalment fee amount that is the same as the estimated cost of credit or the interest/fees amount the user will have to pay during the lifetime of the loan. This field will only be included in instalment credit orders. In the case of “same as cash” or “no interest” offers, this will be set to 0.
`financingTerm`	No	The length of the financing term, in months. Example values are 6, 12, 18 and 24 months.
`financingMonthlyPayment`	No	This is the estimated amount per month that the Customer will need to pay including fees and interest.
`financingTotalCost`	No	This is the estimated total payment amount including interest and fees that the user will pay during the lifetime of the loan.
`financingDiscountAmount`	No	Discount amount for the buyer if paid in one instalment.
`regularTakeFeeAmount`	No	Fee of the regular take rate on the transaction amount. It could be equal to `financingDiscountAmount` in the case of non-instalment transactions.
`noteText`	No	Note to Merchant.
`transactionID`	No	PayPal transaction ID.
`allowedPaymentMethod`	No	The payment method type as specified in the initial request.
`paymentRequestID`	No	A unique identifier of the specific payment request.
`bucketCategoryType`	No	The category of a payment as specified in the initial request.

Field Name	Mandatory?	Description
`instrumentCategory`	No	Identifies the category of the promotional payment instrument. Possible values are: **1** – PayPal Credit® (formerly Bill Me Later®). **2** – A Private Label Credit Card (PLCC) or co-branded payment card.
`instrumentID`	No	An instrument ID (issued by the external party) corresponding to the funding source used in the payment.
`shippingCalculationMode`	No	Describes how the options that were presented to the buyer were determined. Possible values are: **API** – **Callback** **API** – **Flatrate**
`insuranceOptionSelected`	No	The option that the buyer chose for insurance. Possible values are: **Yes** – opted for insurance. **No** – did not opt for insurance.
`shippingOptionIsDefault`	No	Indicates whether the buyer chose the default shipping option. Possible values are: **true** – chose the default shipping option. **false** – did not choose the default shipping option.
`shippingOptionAmount`	No	The shipping amount that the buyer chose.
`shippingOptionName`	No	The name of the shipping option, such as Air or Ground.
`scheduledShippingDate`	No	The scheduled shipping date is returned only if scheduled shipping options are passed in the request.
`scheduledShippingPeriod`	No	The scheduled shipping period is returned only if scheduled shipping options are passed in the request.
`sellerPayPalAcountID`	No	Unique identifier for the merchant. For parallel payments, this field contains either the Payer ID or the email address of the merchant.
`taxIDType`	No	Buyer's tax ID type. This field is required for Brazil and used for Brazil only. For Brazil use only: The tax ID type is BR_CPF for individuals and BR_CNPJ for businesses.

Field Name	Mandatory?	Description
`taxID`	No	Buyer's tax ID. This field is required for Brazil and used for Brazil only. For Brazil use only: The tax ID is 11 single-byte characters for individuals and 14 single-byte characters for businesses
`billingAgreementID`	No	Identification number of the billing agreement. When the buyer approves the billing agreement, it becomes valid and remains valid until it is cancelled by the buyer.
`billingAgreementAcceptedStatus`	No	Indicates whether the buyer accepted the billing agreement for a recurring payment. Currently, this field is always returned in the response for agreement-based products, such as subscriptions; reference transactions; recurring payments; and regular single payment transactions. **0** – Not accepted. **1** – Accepted.
`paymentStatus`	No	Status of the payment. Possible values are: **None** – No status. **Canceled-Reversal** – A reversal has been cancelled: for example, when you win a dispute and the funds for the reversal have been returned to you. **Completed** – The payment has been completed and the funds have been added successfully to your account balance. **Denied** – You denied the payment. This happens only if the payment was previously pending because of possible reasons described for the `pendingReason` element. **Expired** – The authorisation period for this payment has been reached. **Failed** – The payment has failed. This happens only if the payment was made from your buyer's bank account. **In-Progress** – The transaction has not terminated: for example, an authorisation may be awaiting completion. **Partially-Refunded** – The payment has been partially refunded. **Pending** – The payment is pending. See the `pendingReason` field for more information. **Refunded** – You refunded the payment. **Reversed** – A payment was reversed due to a chargeback or other type of reversal. The funds have been removed from your account balance and returned to the buyer. The reason for the reversal is specified in the `reasonCode` element. **Processed** – A payment has been accepted.

Field Name	Mandatory?	Description
		Voided – An authorisation for this transaction has been voided.
`refundStatus`	No	Status of the refund. Possible value are: **none** – returned if the refund fails **instant** – refund was instant **delayed** – refund was delayed
`pendingReason`	No[1]	The reason the payment is pending. Possible values are: **none** – No pending reason. **address** – The payment is pending because your buyer did not include a confirmed shipping address and your Payment Receiving Preferences is set such that you want to accept or deny each of these payments manually. To change your preference, go to the Preferences section of your Profile. **authorization**[2] – The payment is pending because it has been authorised but not settled. You must capture the funds first. **echeck** – The payment is pending because it was made by an eCheck that has not yet cleared. **intl** – The payment is pending because you hold a non-U.S. account and do not have a withdrawal mechanism. You must manually accept or deny this payment from your Account Overview. **multi-currency** – You do not have a balance in the currency sent, and you do not have your Payment Receiving Preferences set to automatically convert and accept this payment. You must manually accept or deny this payment. **order** – The payment is pending because it is part of an order that has been authorised but not settled. **payment-review** – The payment is pending while it is being reviewed by PayPal for risk. **regulatory-review** – The payment is pending while we make sure it meets regulatory requirements. You will be contacted again from 24 to 72 hours with the outcome of the review. **unilateral** – The payment is pending because it was made to an email address that is not yet registered or confirmed. **verify** – The payment is pending because you are not yet verified. You must verify your account before you can accept this payment. **other** – The payment is pending for a reason other than those listed above. For more information, contact PayPal Customer Service.

[1] `pendingReason` is returned in the response only if `paymentStatus` is **Pending**.
[2] This value is received directly from PayPal and so will use the 'ize' and not 'ise' spelling.

Field Name	Mandatory?	Description
`reasonCode`	No	The reason for a reversal if the transaction type is reversal. Possible values are: **none** – No reason code. **chargeback** – A reversal has occurred on this transaction due to a chargeback by your buyer. **guarantee** – A reversal has occurred on this transaction due to your buyer triggering a money-back guarantee. **buyer-complaint** – A reversal has occurred on this transaction due to a complaint about the transaction from your buyer. **refund** – A reversal has occurred on this transaction because you have given the buyer a refund. **other** – A reversal has occurred on this transaction due to a reason not listed above.
`protectionEligibilityType`	No	The kind of seller protection in force for the transaction. Possible values are: **ItemNotReceivedEligible** – Merchant is protected by PayPal's Seller Protection Policy for Item Not Received. **UnauthorizedPaymentEligible**[1] – Merchant is protected by PayPal's Seller Protection Policy for Unauthorised Payments. **Ineligible** – Merchant is not protected under the Seller Protection Policy. (Multiple values are separated by commas)
`feeAmount`	No	PayPal fee amount charged for the transaction.
`settleAmount`	No	Amount deposited in your PayPal account after a currency conversion.
`storeID`	No	Store identifier as entered in the transaction.
`terminalID`	No	Terminal identifier as entered in the transaction.

[1] This value is received directly from PayPal and so will use the 'ize' and not 'ise' spelling.

The details will be returned as a nested record as described in section 1.5.8. The detail names are case sensitive.

20.7 Transaction Lifecycle

PayPal transactions will use the normal Authorise, Capture life cycle as documented in appendix A-14.1 with the following differences. In addition, the PayPal **paymentAction** option can be included in the **checkoutOptions** field to alter the normal payment lifecycle further, to allow an Order, Authorise, Capture model or a straight Sale model to be specified.

20.7.1 Order

If a **paymentAction** with a value of 'Order' is sent, then PayPal will store the transaction but delay authorising it until instructed. To instruct PayPal to authorise the transaction, a further management request can be sent to the Gateway with an **action** of 'AUTHORISE' and the **xref** of the transaction to authorise. Alternatively, the AUTHORISE command can be selected in the Merchant Management System (MMS). The transaction will be left in the 'received' state.

20.7.2 Authorise

If no **paymentAction** is specified or a **paymentAction** with a value of 'Authorize' is sent, then PayPal will authorise the transaction on receipt as per a standard card transaction and you can capture it later if you used the **captureDelay** field. *Note that the value uses the PayPal spelling 'Authorize', and not the British spelling 'Authorise'.*

For the first three days (by default) of the authorisation, funds are reserved. This is known as the honour period. After the honour period, captures can still be attempted, but may be returned with insufficient funds.

Authorisations have a fixed expiry period of 29 days.

20.7.3 Sale

If a **paymentAction** with a value of 'Sale' is sent, then PayPal will immediately capture the transaction after authorisation. The transaction will be regarded as having been settled and you will not be able to capture it manually and it will not be sent for settlement that evening. The transaction will be left in either the **accepted** or **rejected** terminal states depending on whether PayPal accepted or rejected the transaction.

20.7.4 Capture

Transactions that have been authorised by PayPal and not immediately settled due to a **paymentAction** of 'Sale' will be able to be captured as normal.

Captures are sent to PayPal immediately and the PayPal response and the transaction will be left in either the **accepted** or **rejected** terminal state depending on whether PayPal accepted or rejected the capture request.

There is no need to wait for the nightly settlement batch to run as with normal card transactions. This means that it is not possible to change the amount to be captured or cancel the transaction one a capture has been requested.

Note: PayPal allows multiple captures where they sum the individual capture amounts – i.e. in a different manner from the Gateway's, where only a single capture operation can be processed.

20.7.5 Refund

PayPal transactions can be refunded in the same way as normal card transactions. However, in the same way as capture requests, these will be sent to PayPal immediately and not batched up to be sent as part of the nightly settlement process. This means that the transaction will be left in either the **accepted** or **rejected** terminal state, depending on whether PayPal accepted or rejected the refund request.

Refunds can be made for full or partial amounts, with multiple refunds allowed up to the original authorised amount.

By default, PayPal allows a Merchant up to 60 days from the original authorised transaction date to perform refunds.

20.7.6 Cancel

You should cancel any transactions that you do not wish to capture in order to prevent 'pending' transactions on the Customer's PayPal account.

Authorisations should be cancelled when an initial authorisation was created to confirm the validity of funds during checkout, but the goods will not ship for a significant amount of time (>29 days). Cancelling the transaction will mean that you will have to contact the Customer for an alternative payment method.

All transactions must be completed by being captured or cancelled.

20.7.7 Pending Payments

PayPal may put a transaction into a pending state when flagged for additional fraud review. This state is known to PayPal as payment review or IPR.

IPR transactions will be automatically cancelled by the Gateway and treated as referred transactions with a **responseCode** of **2** and a **responseMessage** indicating the reason that the transaction was put into a pending state. Unlike card referred transactions, an authorisation code cannot be obtained from PayPal verbally and then the transaction resent.

20.8 Reference Transactions

PayPal does not allow ad hoc ‘Card On File’ type repeat or recurring transactions using the **xref** of a reference transaction unless that transaction has specifically started a PayPal Billing Agreement.

If you want to be able to make future repeat or recurring transactions, then the initial transaction must include the **billingType** and **billingAgreementDescription** options in the **checkoutOptions** to identify this transaction as the start of a recurring billing sequence.

This will cause the Gateway to request PayPal to set up a Billing Agreement between you and the Customer. In this case, the PayPal Billing Agreement ID will be returned as part of the **checkoutDetails** and displayed on the Merchant Management System (MMS) as part of the payment details, so that you can easily see which PayPal transactions can be used for recurring billing.

21 Amazon Pay Transaction

21.1 Background

Amazon Pay is an additional payment method that is available to all Merchants using the Gateway that have an Amazon Pay account.

To use Amazon Pay, you will be supplied with a separate Amazon Pay Merchant account that can be grouped with your main Merchant account using the account mapping facility as documented in appendix A-8. This allows transactions to be sent using your main Merchant Account and then routed automatically to the Amazon Pay Merchant Account in the same mapping group.

It allows you to offer payment via Amazon Pay in addition to normal card payments.

Amazon Pay transactions will appear in the Merchant Management System (MMS) alongside any card payments and can be captured, cancelled and refunded in the same way as card payments.

Amazon Pay transactions can also be used for recurring billing but require you to indicate in the initial transaction that it will be the basis for recurring billing and a billing agreement will be entered into between your Customer and Amazon Pay when they agree to the payment.

Amazon Pay transactions cannot be used for ad-hoc ‘Card On File’ repeat transactions unless a billing agreement has been set up.

For more information on how to accept Amazon Pay transactions, please contact customer support.

Amazon Pay is supported by the Hosted and Direct Integrations. It is not supported by the Batch Integration.

21.2 Benefits and Limitations

21.2.1 Benefits

Provides your customers with the flexibility of paying using their Amazon Pay account when this is more suitable to them.

The Amazon Pay Checkout can be added as an overlay on the standard checkout to help improve conversion rates with an easier way to pay without customers leaving your website.

There are no extra costs to add an Amazon Pay Merchant Account. However, you will still be liable for the Amazon Pay transaction fees.

The full Amazon Pay transaction information is available and returned as part of the transaction.

Transactions are controlled within the Merchant Management System (MMS) in the same manner as normal card transactions.

21.2.2 Limitations

You will need an Amazon Pay account.

Recurring transactions are not supported unless part of a prearranged billing agreement.

Independent refunds that are not tied to a previous sale transaction are not supported without prior agreement.

Transactions require a browser in order to display the Amazon Pay Checkout widgets.

21.3 Hosted Implementation

If a transaction is sent to the Hosted Integration using a **merchantID** that is part of a routing group containing an Amazon Pay Merchant, then the Hosted Payment Page will display an Amazon Pay payment button which, when clicked, will open the Amazon Pay Checkout and allow the Customer to pay using their Amazon Pay account.

To customise the Amazon Pay Checkout experience, you may send various options in the **Amazon PayCheckoutOptions** field in your initial request.

Additional information available from Amazon Pay will be made available in the **checkoutDetails** response field.

Note: Custom Hosted Payment Pages might not support the displaying of the Amazon Pay Checkout button. If you have a custom page that doesn't support this, then you would need to contact support to have your Hosted Payment Page upgraded.

21.4 Direct Implementation

Amazon Pay transactions require you to display an Amazon Pay Checkout to your Customer as part of the transaction flow. The transaction must be done in two stages, with the Checkout page being displayed between the stages. They can also optionally be done in three stages, allowing you to display an order confirmation after the Checkout page and before authorising the transaction. You can change the amount at this stage to allow for shipping costs when you know the confirmed delivery address the Customer selected as part of the Amazon Pay Checkout.

Amazon Pay do not provide a ready built Checkout page and require you to create one on your servers using the JavaScript widget toolkit they provide.

Amazon Pay supports the normal payment and management actions. This section explains how to make payment requests. Management requests are performed as detailed in section 3.

21.4.1 Initial Request (Checkout Preparation)

To request that a transaction be processed via Amazon Pay, the request must contain a **paymentMethod** of 'Amazon Pay' In addition, you may send **checkoutOptions** to customise the Checkout experience. When the Gateway receives this **paymentMethod**, assuming there are no other errors with the request, it will attempt to find a suitable Amazon Pay Merchant Account in the current account mapping group.

If the Gateway is unable to find a suitable account, then the transaction will be aborted, and it will respond with a **responseCode** of **66364 (INVALID PAYMENTMETHOD)**.

Otherwise, the Gateway will respond with a **`responseCode`** of **65826 (CHECKOUT REQUIRED)** and the response will include a **checkoutURL** field containing the URL required to load the Amazon Pay JavaScript Widgets; and a **checkoutRequest** containing any data required by those Widgets. The response will also contain a unique **checkoutRef** that must be echoed back in the continuation requests.

At this point, your server must create an Amazon Pay Checkout page using their JavaScipt Widgets. Further details on how to use the Widgets are provided in the Amazon Pay guide at https://developer.Amazon Pay.com/docs/classic/express-checkout/in-context/enable_in_context_checkout/.

21.4.2 Continuation Request (Checkout Details and Authorise)

On completion of the Amazon Pay Widgets, the Merchant should send the information created by the Widgets to the Gateway together with a **status** value. If the Checkout was successful, the status will be 'success'; alternatively, if the Checkout was cancelled, the status will be 'cancel'. Any **accessToken** generated by the Amazon Pay Login Widget; **orderReferenceID**, generated by the Wallet or Address Widgets; and **billingAgreementID** generated by the optional Billing Widget, must be added to the **checkoutResponse** field and sent in a new request to the Gateway. The **checkoutResponse** field can be sent either as a URL query string; as a JSON encoded string; or as an array of parameters. This new request will load the Checkout details, including any purchaser and delivery address details as required, and send the transaction to Amazon Pay for

authorisation, returning the result as in the case of a normal authorisation transaction. The new request must contain the **checkoutRef** received in the initial response.

21.4.3 Separate Checkout Details and Authorisation Requests

You can choose to obtain the Checkout details before sending the transaction for authorisation by sending the **checkoutOnly** field in the above continuation request. If this field is sent with a value of 'Y' then the Gateway will load the Checkout details and then return them to you without sending the request for authorisation. You can then display them and/or adjust the amount, for example, according to delivery charges appropriate to the received delivery address. You should then send a new request containing the **checkoutRef** received to continue the transaction and authorise it.

Note: this stage can be repeated multiple times by including the **checkoutOnly** field with a value of 'Y' each time. To complete the transaction, the final request must not contain the **checkoutOnly** field or it must not have a value of 'Y'.

21.5 Request Fields

21.5.1 Initial Request (Hosted and Direct Integration)

These fields should be sent in addition to basic request fields in section 2.1 excluding any card details.

Field Name	Mandatory?	Description
paymentMethod	Yes[1]	Must contain the value 'Amazon Pay' in lower case letters only.
checkoutRedirectURL	No[2]	Reserved for future use.
checkoutOptions	No[3]	Record containing options used to customise the Amazon Pay Checkout. Refer to section 22.5.3 for values.
AmazonPayCheckoutOptions	No[4]	Record containing options used to customise the Amazon Pay Checkout. Refer to section 22.5.3 for values.

[1] Optional for Hosted Integration
[2] Not required for Hosted Integration.
[3] Direct Integration Only
[4] Hosted Integration Only

21.5.2 Continuation Request (Direct Integration)

These fields may be sent alone[1].

Field Name	Mandatory?	Description
checkoutRef	Yes	Unique reference return in the initial response.
checkoutResponse	Yes	The data received from the Amazon Pay Checkout Widgets together with a status value.
checkoutOnly	No	Pass **Y** to complete the processing as far as the next Checkout stage and then return with the loaded Checkout details.

[1] It is only necessary to send the **checkoutRef** and the **checkoutResponse** in the continuation request because the **checkoutRef** will identify the Merchant Account and initial request. The message does not have to be signed. You can send any of the normal request fields to modify or supplement the initial request – however, in this case the request should be signed. The **checkoutRedirectURL** and **checkoutOptions** fields sent in the initial request cannot be modified and any sent in the second request must match those used in the first request, or the second request will fail with a **responseCode** of **64442 (REQUEST MISMATCH)**.

21.5.3 Checkout Options (Hosted and Direct Integration)

The following options may be sent in the **Amazon PayCheckoutOptions** Hosted Integration field or the **checkoutOptions** Direct Integration field to customise the Amazon Pay Checkout.

Field Name	Description
`billingAgreementRequired`	Can be used to specify that a billing agreement must be started. Alternatively, the `rtAgreementType` standard integration field can be used with a value of 'recurring' or 'instalment'.
`shippingAddressRequired`	Indication that the shipping address is required, and the Address Checkout Widget will be used.
`sellerOrderID`	The Merchant specified identifier for this order. If not sent, then any value in the `merchantOrderRef` standard integration field is used.
`sellerNote`	Represents a description of the order that is displayed in emails to the buyer.
`sellerAuthorizationNote`	A description for the authorisation transaction that is shown in emails to the buyer.
`sellerCaptureNote`	A description for the capture that is displayed in emails to the buyer.
`sellerBillingAgreementID`	The Merchant specified identifier for this billing agreement. If not sent, then any value in the `rtPolicyRef` standard integration field is used.
`customInformation`	Any additional information that you want to include with this order reference
`supplementaryData`	Supplementary data.
`softDescriptor`	The description to be shown on the buyer's payment statement
`billingAgreementRequired`	Can be used to specify that a billing agreement must be started. Alternatively, the `rtAgreementType` standard integration field can be used with a value of 'recurring' or 'instalment'.
`shippingAddressRequired`	Indication that the shipping address is required, and the Address Checkout Widget will be used.

For further information on the options refer to the Amazon Pay API Reference Guide: https://pay.amazon.com/us/developer/documentation/apireference/201751630

The options should be passed as either a nested record or serialised record as described in section 1.5.8. The option names are case sensitive.

21.5.4 Response Fields

21.5.5 Initial Response (Direct Integration)

These fields will be returned in addition to the request fields from section 22.5.1 and the basic response fields in section 2.2 minus any card details.

Field Name	Mandatory?	Description
checkoutRef	Yes	Unique reference required to continue this transaction when the Amazon Pay Checkout has completed.
checkoutName	Yes	Unique name of the Checkout. For Amazon Pay this is the value **Amazon Pay**.
checkoutURL	Yes	URL required to load the Amazon Pay JavaScript Widgets.
checkoutRequest	No	Information required for the Amazon Pay Widgets such as: **merchantID** – Amazon Pay merchant ID **clientID** – Amazon Pay client ID **sandbox** – true if Amazon Pay sandbox **region** – Amazon Pay API region code **scope** – Login Widget scope parameter
checkoutOptions	No	Any Checkout options passed in the request.
acquirerResponseDetails	Yes	Details about the Amazon Pay response containing any error messages and codes. This can be used together with the normal **responseCode** and **responseMessage** response fields to further determine the reason for any failure.

21.5.6 Continuation Response (Direct Integration)

These fields will be returned in addition to the request fields from section 22.5.2, the initial response fields in section 20.6.1 and the basic response fields in section 2.2 minus any card details.

Field Name	Mandatory?	Description
checkoutRef	Yes	Provided if **checkoutOnly** was used in the continuation response to indicate that a further request will be sent to finalise the transaction.
checkoutName	Yes	Unique name of the Checkout. For Amazon Pay this is the value **Amazon Pay**.
checkoutDetails	Yes	Record containing values made available by the Amazon Pay Checkout. Refer to section 21.5.7 for values.
customer*XXXX*	No[11]	Customer details if provided by the Amazon Pay Checkout as documented in section 17.1
delivery*XXXX*	No	Delivery details if provided by the Amazon Pay Checkout as documented in section 17.4
receiver*XXXX*	No	Buyer details if provided by Amazon Pay as documented in section 17.5. Amazon Pay will usually provide the buyer's name, postcode and email only, which are returned in the **receiverName**, **receiverPostcode** and **receiverEmail** fields accordingly
acquirerResponseDetails	Yes	Details about the Amazon Pay response containing any error messages and codes. This can be used together with the normal **responseCode** and **responseMessage** response fields to further determine the reason for any failure.

[11] The response will include Customer/billing address and delivery address details if provided by the Amazon Pay Checkout.

21.5.7 Checkout Details (Hosted and Direct Integration)

The **checkoutDetails** field included in the response above will contain the following values and any further values received from Amazon Pay allowing the Merchant to see the full Amazon Pay order information.

Field Name	Mandatory?	Description
`referenceID`	No	Amazon Pay reference id. Either the `orderReferenceID` or the `billingReferenceID` where appropriate.
`accessToken`	No	Amazon Pay order reference id as sent in the continuation request `checkoutResponse` data.
`bilingAgreementID`	No	Amazon Pay order reference id as sent in the continuation request `checkoutResponse` data.
`orderReferenceID`	No	Amazon Pay order reference id as sent in the continuation request `checkoutResponse` data.

The details will be returned as a nested record as described in section 1.5.8. The detail names are case sensitive.

21.6 Transaction Lifecycle

Amazon Pay transactions will use the normal Authorise, Capture life cycle as documented in appendix A-14.1 with the following differences.

21.6.1 Capture

Captures made by the Direct Integration or Merchant Management System (MMS) are sent to Amazon Pay immediately. The transaction will be left in either the **accepted** or **rejected** terminal state depending on whether Amazon Pay accepted or rejected the capture request. Unlike card payments, captures do not flag the transaction to be included in the nightly settlement batch and therefore when done they cannot be redone. This means that it is not possible to change the amount to be captured or cancel the transaction when a capture has been requested.

Captures that are not explicitly performed such as normal transactions or those with a captureDelay are still done as part of the nightly settlement batch.

Transactions that are not captured within 3 days will be placed in a pending state in the Amazon Pay system which is reflected as the **tendered** state in the Gateway and will show on the Merchant Management System as being settled.

21.6.2 Refund Sale

Amazon Pay transactions can be refunded the same as normal card transactions however, like capture requests, these will be sent to Amazon Pay immediately and not batched up and sent as part of the nightly settlement process. This means the transaction will be left in either the **accepted** or **rejected** terminal state depending on whether Amazon Pay accepted or rejected the refund request.

Refunds can be made for full or partial amounts, with multiple refunds allowed up to the original authorised amount.

21.7 Reference Transactions

Amazon Pay does not allow ad hoc 'Card On File' type repeat or recurring transactions using the **xref** of a reference transaction unless that transaction has specifically started a Amazon Pay Billing Agreement.

If you want to be able to make future repeat or recurring transactions, then the initial transaction must include an **rtAgreementType** of **recurring** or **instalment**. Alternatively, the **`billingAgreementRequired`** option can be included in the **checkoutOptions** to identify this transaction as the start of a recurring billing sequence.

This will cause the Gateway to request Amazon Pay setup a Billing Agreement between you and the Customer. In this case the Amazon Pay Billing Consent Widget must be used in the Checkout and the **billingAgreementID** it creates sent in the **checkoutResponse** data in the continuation request. Any billing agreement ID will be displayed on the Merchant Management System (MMS) as part of the payment details so that you can easily see which Amazon Pay transactions can be used for recurring billing.

22 PPRO Transactions

22.1 Background

PPRO is an additional payment method that is available to all Merchants using the Gateway that have a PPRO account.

To use PPRO, you will be supplied with a separate PPRO Merchant account that can be grouped with your main Merchant Account using the account mapping facility as documented in appendix A-8. This allows transactions to be sent using your main Merchant Account and then routed automatically to the PPRO Merchant Account in the same mapping group.

PPRO is an Acquirer that offers many Alternative Payment Methods (APM), that you can then offer to your Customers.

E-wallets, SMS payments and PSP services are some of the many payment methods PPRO support (e.g. Alipay, EasyPay, Bancontact). This could allow a business to facilitate overseas transactions or alternative payment methods using a different payment method suitable for that country or business plan.

All transactions created with this payment method will appear in the Merchant Management System (MMS) together with the payment method that was used to process the transaction.

PPRO transactions cannot be used for ad-hoc ‘Card On File’ repeat transactions or for recurring billing.

For more information on how to accept PPRO transactions please contact customer support.

PPRO is supported by the Hosted and Direct Integrations. It is not supported by the Batch Integration.

22.2 Benefits and Limitations

22.2.1 Benefits

Multiple alternative payment methods could be used.

Expands range of payment methods for international use.

Supports a variety of e-wallets, SMS and PSP's.

Ability to perform refunds on supported payment methods.

Transactions are controlled within the Merchant Management System (MMS) in the same manner as normal card transactions.

22.2.2 Limitations

You will need a PPRO account.

Payment authorisation is not always instantaneous and may require additional 'QUERY' requests.

An alternative payment method may only support one or a limited set of currencies or countries.

Alternative payment methods require a browser in order to display their Checkout.

22.3 Hosted Implementation

If a transaction is sent to the Hosted Integration using a **merchantID** which is part of a routing group containing a PPRO Merchant Account, then the Hosted Payment Page will show alternative payment method buttons for each payment method listed in the **allowedPaymentMethods** field. When clicked on the Hosted Payment Page may request further details from the Customer before opening the APM Checkout allowing the Customer to pay using that APM.

To customise the alternative payment methods checkout experience, you may send various options in the **pproCheckoutOptions** field in your initial request.

Additional information available from PPRO will be made available in the **checkoutDetails** response field.

Note: Custom Hosted Payment Pages might not support the displaying of the Alternative Payment Methods. If you have a custom page that doesn't support this, then you would need to contact support to have your Hosted Payment Page upgraded.

22.4 Direct Implementation

PPRO transactions require you to display the alternative payment method's Checkout to your Customer as part of the transaction flow. The transaction must be done in two stages with the Checkout being displayed between the stages.

PPRO supports only supports the SALE, REFUND_SALE actions. This section explains how to make payment requests. Management request are performed as detailed in section 3.

22.4.1 Payment Request

To request that a transaction be processed via PPRO the request must contain a **paymentMethod** of 'ppro.*XXXX*', where *XXXX* is the PPRO payment method tag listed in section 22.4.3 below. The request must also have a **checkoutRedirectURL** containing the URL of a page on your server to return to when the alternative payment method's Checkout is closed. In addition, you may send **checkoutOptions** to provide further custom fields requited by the alternative payment method as details in section 22.4.2 below.

When the Gateway receives these fields, assuming there are no other errors with the request, it will attempt to find a suitable PPRO Merchant Account in the current account mapping group.

If the Gateway is unable to find a suitable account, then the transaction will be aborted, and it will respond with a **responseCode** of **66364 (INVALID PAYMENTMETHOD)**.

Otherwise the Gateway will respond with a **`responseCode`** of **65826 (CHECKOUT REQUIRED)** and included in the response will be a **checkoutURL** field containing the URL that the buyer's browser should be redirected to in order to complete the payment. The response will also contain a unique **checkoutRef** which must be echoed back in the continuation requests.

On completion of the third-party payment the browser will be directed to the **checkoutRedirectURL** you provided, complete with information about the payment in a HTTP POST request. The posted data will contain a **checkoutResponse** field that will contain any specific response data for the payment method.

22.4.2 Payment Specific Fields

Most of the information required by the alternative payment methods can be supplied using the standard Gateway request fields. However, there may be specific mandatory fields required by a payment method which are not available using the standard fields. In these cases, these fields can be sent in the **`checkoutOptions`** data.

For example, most European services may require the **`nationalid`** and **`consumerref`** fields.

Recurring transactions will require the use of **`iban`** (optionally **`sequencetype`**) and in follow-up payments; **`mandatereference`**, **`mandatesignaturedate`**, and **`sequencetype`**.

Customer support will be able to help guide you on any missing fields you may find the transaction will come up with a **`responseCode`** of **65550 (PROCESSOR_ERROR -** Invalid request data).

22.4.3 Payment Method Tags

To specify which alternative payment method is required you need to send the **`paymentMethod`** field with a value using the format is 'ppro.*XXXX*' , where *XXXX* is the alternative payment method's tag name as assigned by PPRO.

For example; to use the alternative payment method AstroPay Card that has a tag name of "astropaycard" (all lowercase); the resulting payment method code would be "ppro.astropaycard". This allows the Gateway to know that you're attempting to use AstroPay Card using the PPRO payment method.

The table below is a guide to the tag names available. This list is fluid as PPRO add and remove methods.

If you know of a payment method that is not on this list or the payment method cannot be used; please contact customer support for advice.

Tag	Name
affinbank	Affin bank
alipay	AliPay
ambank	AmBank
argencard	Argencard
astropaycard	AstroPay Card
astropaydirect	AstroPay Direct
aura	Aura
baloto	Baloto
banamex	Banamex
bancodobrasil	Banco do Brasil
bancodechile	Banco de Chile
bancodeoccidente	Banco de Occidente
bancomer	Bancomer
bankislam	Bank Islam
bcmc	Bancontact
bitpay	Bitpay
boleto	Boleto Bancario

bradesco	Bradesco
cabal	Cabal
cartaomercadolivre	Cartao Mercado Livre
carulla	Carulla
ccauth	Credit/Debit Card
ccweb	Credit/Debit Card
cencosud	Cencosud
cimbclicks	CIMB Clicks
cmr	CMR
davivienda	Davivienda
directpay	Sofortüberweisung (Direct Pay)
dragonpay	Dragonpay
easypay	EasyPay
efecty	Efecty
elo	Elo
empresedeenergia	Emprese de Energia
enets	eNETS
entercash	Entercash
eps	EPS
estonianbanks	Estonian Banks
giropay	Giropay
hipercard	Hipercard
hongleongbank	Hong Leong Bank
ideal	iDEAL
instanttransfer	Instant Transfer
int_payout	International Pay-Outs
itau	Itau
latvianbanks	Latvian Banks
lithuanianbanks	Lithuanian Banks

magna	Magna
maxima	Maxima
maybanktwou	Maybank2u
multibanco	Multibanco
mybank	MyBank
myclearfpx	MyClear FPX
naranja	Naranja
narvesen	Narvesen
nativa	Nativa
oxxo	OXXO
p24	Przelewy24
p24payout	Przelewy24 Payout
pagofacil	Pago Facil
paypost	PayPost
paysafecard	Paysafe Card
paysbuy	Paysbuy
paysera	Paysera
payu	PayU
perlas	Perlas Terminals
poli	OLI
presto	Presto
pse	PSE
pugglepay	Pugglepay
qiwi	QIWI
qiwipayout	QIWI Payout
rapipago	Rapipago
redpagos	Redpagos
rhbbank	RHB Bank
safetypay	SafetyPay

santander	Santander
sepadirectdebit	SEPA DirectDebit
sepapayout	SEPA Payout
seveneleven	Seveneleven (7eleven)
singpost	SingPost
skrill	Skrill
surtimax	Surtimax
tarjetashopping	Tarjeta Shopping
trustly	Trustly
trustpay	TrustPay
unionpay	UnionPay
verkkopankki	Verkkopankki – Finish Online Banking
webpay	Webpay
yellowpay	Yellow Pay

22.5 Request Fields

22.5.1 Initial Request (Hosted and Direct Integration)

These fields should be sent in addition to the basic request fields in section 2.1 excluding any card details.

Field Name	Mandatory?	Description
paymentMethod	Yes[1]	Payment method to be used with PPRO (e.g. **ppro.astropay**, **ppro.alipay**, etc.).
checkoutRedirectURL	Yes[2]	URL on Merchant's server to return to when the Alternative Payment Method's Checkout is closed.
checkoutOptions	No[3,4]	Record containing options used to customise the alternative payment methods Checkout. Refer to section 22.5.2 for values.
pproCheckoutOptions	No[5,4]	Record containing options used to customise the alternative payment methods Checkout. Refer to section 22.5.2 for values.

[1] Optional in Hosted Integration.
[2] Not required for Hosted Integration.
[3] Direct Integration only.
[4] Whilst the Gateway does not see this field as mandatory, PPRO may have payment methods that require additional configuration using checkout options.
[5] Hosted Integration Only

22.5.2 Checkout Options (Hosted and Direct Integration)

The following options may be sent in the **pproCheckoutOptions** Hosted Integration field or the **checkoutOptions** Direct Integration field to customise the Checkout.

Field Name	Description
`nationalid`	Consumer's national ID (up to 30 characters).
`consumerref`	Unique reference identifying the consumer within 1 to 20 characters and a format of A-Za-z0-9.%,&/+*$-
`siteid`	Unique site identifier. Required for clients serving multiple points of sale and forwarded onwards whilst using the qiwi payment method.
`iban`	Valid IBAN of consumer/destination account.
`sequencetype`	Sequence type of the direct debit. Possible values are: **oneOff** – The direct debit is executed once (***default***) **first** – First direct debit in a series of recurring ones
`mandatereference`	Mandate reference as returned on the first transaction in the sequence (found from mandatereference in checkoutDetails)
`mandatesignaturedate`	Date of the initial transaction.
`bic`	Valid BIC (8 or 11 alphanumeric letters) – optionally supplied to skip the bank selection page (by using the bank referenced by BIC as supplied)
`clientip`	Optional IP address of the consumer during checkout using Trustly (127.0.0.1 is not allowed!)
`address`	Customer's billing address[1]
`city`	Customer's billing city[1]
`phone`	Customer's phone[1]
`mobilephone`	Customers mobile phone[1]
`dob`	MCC 6012 Date of Birth[1]
`dynamicdescriptor`	Statement narrative[1]

[1] This information is supplied to PPRO by default using the following fields: `customerAddress`, `customerPostcode`, `customerTown`, `customerEmail`, `customerPhone`, `customerMobile`, `receiverDateOfBirth`, `statementNarrative1`.

The options should be passed as either a nested record or serialised record as described in section 1.5.8. The option names are case sensitive.

22.6 Response Fields

22.6.1 Initial Response (Direct Integration)

The fields below will be returned in addition to the basic response fields in section 2.2 for the start of a PPRO transaction and the PPRO checkout process.

Field Name	Mandatory?	Description
`checkoutRef`	Yes	Unique reference required to continue this transaction when the PPRO Checkout has completed.
`checkoutName`	Yes	The `paymentMethod` you used to identify the PPRO payment method.
`checkoutRedirectURL`	Yes	The URL to redirect the Customer to, to start the checkout process.
`checkoutOptions`	Yes	The same `checkoutOptions` used for the request.
`checkoutDetails`	Yes	Additional information provided from the payment method used during checkout.
`checkoutRef`	Yes	The unique reference required to continue the transaction when PPRO checkout is complete.
`checkoutRequest`	Yes	Containing the redirect secret, checksum and request status.

22.6.2 Completion Response (Hosted and Direct Integration)

Fields from the initial response in the previous section may be present as well as the fields below and will not contain any card details.

Field name	Mandatory?	Description
`checkoutResponse`	Yes	Containing additional information provided by the Checkout. Any change in the payment's status will be given in `responseMessage` and `responseCode`[1]
`checkoutStatus`	Yes	A string containing the result of the checkout process. **This is not used to identify the** transaction's payment **status.**

[1] Not all payment methods give an immediate payment status. This will require a further QUERY to the Gateway to see whether this value has changed to a status of 'tendered'.

22.6.3 Notifications and “Tendered” Payments

Whilst some payment methods give an immediate payment status (i.e. direct card payment methods rather than SMS and e-wallet systems), some payments may come back with the status of ‘tendered’. At this time, online shopping modules will not be able to monitor the transaction status. The use of a QUERY request may be of use as seen in section 1.7.8. Please ask customer support in this matter who will be able to give more information and may be able to provide better advice for your situation.

Notifications from PPRO regarding the payment status, seconds, minutes or hours after the checkout will automatically update the transaction status.

23 Digital Wallet Transactions

23.1 Background

The Gateway currently supports payments made using the following Digital Wallets:

Google Pay™
Apple Pay

These are collectively known as ‘The Pays’.

These wallets can be used to enhance mobile purchasing experiences for customers with supported devices and produce a payment token which can be passed to the Gateway instead of the Cardholder’s actual card details.

You can use these wallets with any Merchant Account that has been configured to accept them.

For more information on how to accept payment tokens, please contact customer support.

Digital Wallets are currently supported by the Direct Integration only. They are not supported by the Hosted or Batch Integration.

Note, both Apple Pay and Google Pay are available via accredited Acquirers only.

23.2 Benefits and Limitations

23.2.1 Benefits

The payment details are stored externally to the Gateway and can be used with any Merchant that supports the appropriate payment tokens.

Customers can select from previously stored payment details, making the checkout process more streamlined, resulting in fewer abandoned carts and thus increasing sales.

Compatible with existing card base fraud solutions such as Address Verification Service (AVS), 3-D Secure and third-party fraud providers.

There are no extra costs to add these payment methods to your Gateway account.

The transactions are controlled within the Merchant Management System (MMS) in the same manner as normal card transactions.

23.2.2 Limitations

Your Customer will need a digital wallet enabled device with some stored card details in order to make full use of this payment method.

The device needs to be integrated with the gateway using third-party provided software.

Repeat transactions using the retrieved payment details are supported.

23.3 Configuration

The Merchant Account being used for the payments must be configured with your Digital Wallet credentials so that the Gateway can decrypt the payment token.

23.3.1 Apple Pay configuration

Apple Pay requires the Gateway to generate public/private key pair and then the public key must be shared with your Android Pay enabled application in the guise of an Apple Pay *payment process certificate*.

To configure an Apple Pay *payment processing certificate* you must have enrolled in the Apple Developer Program and created a unique Apple Pay merchant identifier.

The *payment processing certificate* is associated with your merchant identifier and used to encrypt payment information. The certificate expires every 25 months. If the certificate is revoked, you can recreate it.

You would normally use the Merchant Management System (MMS) to configure your payment processing certificate by following the steps outlined below:

1. Open the Apple Developer Certificates, Identifiers & Profiles webpage and select 'Identifiers' from the sidebar.
2. Under 'Identifiers', select 'Merchant IDs' using the filter in the top-right.
3. On the right, select your merchant identifier.
4. Under 'Apple Pay Payment Processing Certificate', click 'Create Certificate'.
5. Download our certificate signing request (CSR) from the MMS and save to a file.
6. Click 'Choose File' and select the CSR you just downloaded.
7. Click 'Continue'.
8. Click 'Download' to download the *payment processing certificate* and save to a file.
9. Upload the payment processing certificate to the MMS.

23.3.2 Google Pay configuration

Google Pay requires no specific configuration however you must use our Gateway identifier of '**crst**' and the correct Merchant Account identifier when configuring your Google Pay enabled application.

Details of Google Pay's brand guidelines, integration checklist and developer documentation can be found on their website:

Google Pay Web developer documentation
Google Pay Web integration checklist
Google Pay Web Brand Guidelines.

blink

23.4 Hosted Implementation

Transactions using Digital Wallet payment methods are currently not supported by the Hosted Integration.

blink

23.5 Direct Implementation

Digital Wallet payments require the secure payment token generated by the wallet enabled application to be sent to the Gateway in the **paymentToken** field. The type of token must be specified by also sending the **paymentMethod** field with a value of '**applepay**' or **'googlepay'**.

23.6 Request Fields

These fields should be sent instead of the standard card details together with the fields in section 2.1.

Field Name	Mandatory?	Description
`paymentMethod`	Yes	The type of payment token sent. **applepay** – to indicate an Apple Pay token **googlepay** – to indicate a Google Pay token
`paymentToken`	Yes	Must contain the secure payment token produced by the wallet enabled application.

23.7 Response Fields

There are no additional response fields.

23.8 Digital Wallet Tokens

Digital Wallet payments operate the same as normal card payments, the main difference is that the card details are passed from the wallet application within an encrypted payment token. Once the Gateway has extracted the card details then it can use it with 3-D Secure and Fraud checking services as normal.

23.8.1 FPAN/DPAN tokens

Apple Pay, Android Pay and Google Pay (Mobile) payment tokens contain an EMV tokenised card number also known as a device-specific number (DPAN) rather than the Cardholder's actual card number (FPAN). With these tokens the expiry date is the date the DPAN expires rather than the value printed on the Cardholder's card. The card mask returned by the Gateway will be the masked DPAN, the Gateway is not able to return the last 4 digits of the FPAN. The card issuing details return should the same as those of the original FPAN.

Google Pay (Web) payment tokens contain the Cardholder's original card number (FPAN) and expiry date. This means that the card mask and expiry date will be those of the original card.

23.8.2 AVS/CV2 Checking

Digital wallet payment tokens do not contain any address or CVV details. The Cardholder's billing address can be passed in the transaction along with the payment token so that address checking can be performed.

The Gateway and Acquirer will not perform CVV checks with these payment tokens effectively disabling CVV checks for the transaction disregarding your preferences.

23.8.3 3-D Secure Authentication

DPAN based tokens will usually contain 3-D Secure data and so the Gateway will send this data to the Acquirer to gain the benefits of an authenticated transaction without the need to challenge the Cardholder. This makes using the digital wallet a much simpler and frictionless method of payment.

FPAN based tokens can passed to the Gateway's 3-D Secure processing and undergo the normal authentication journey as a manually entered card number.

23.8.4 Risk Checking

Both FPAN and DPAN based tokens can be used with risk checking via Kount in the same manner as a normal card transaction.

23.8.5 Transaction Lifecycle and Recurring Transactions

Both FPAN and DPAN based tokens will follow the standard transaction lifecycle and can be cancelled, captured, refunded or used as the basis of subsequent transactions in the same manner as a normal card transaction.

blink

A-1 Response Codes

The Gateway will always issue a **`responseCode`** to report the status of the transaction. These codes should be used rather than the **`responseMessage`** field to determine the outcome of a transaction.

A zero response code always indicates a successful outcome.

Response codes are grouped as follows, the groupings are for informational purposes only and not all codes in a group are used:

Acquirer (FI) Error codes: 1-99	
Code	**Description**
0	Successful / authorised transaction. Any code other than **0** indicates an unsuccessful transaction.
1	Card referred – Refer to card issuer.
2	Card referred – Refer to card issuer, special condition.
4	Card declined – Keep card.
5	Card declined.
30	An error occurred. Check **`responseMessage`** for more detail.

General Error Codes: 65536 - 65791	
Code	**Description**
65536	Transaction in progress. Contact customer support if this error occurs
65537	Reserved for future use. Contact customer support if this error occurs
65538	Reserved for future use. Contact customer support if this error occurs
65539	Invalid Credentials: **`merchantID`** is unknown
65540	Permission denied: caused by sending a request from an unauthorised IP address
65541	Action not allowed: the transaction state or Acquirer doesn't support this action
65542	Request Mismatch: fields sent while completing a request do not match initially requested values. Usually due to sending different card details to those used to authorise the transaction when completing a 3-D Secure transaction or performing a REFUND_SALE transaction.
65543	Request Ambiguous: request could be misinterpreted due to inclusion of mutually exclusive fields
65544	Request Malformed: couldn't parse the request data
65545	Suspended Merchant account

General Error Codes: 65536 - 65791	
Code	**Description**
65546	Currency not supported by Merchant
65547	Request Ambiguous, both `taxValue` and `discountValue` provided when should be one only
65548	Database error
65549	Payment processor communications error
65550	Payment processor error
65551	Internal Gateway communications error
65552	Internal Gateway error
65553	Encryption error.
65554	Duplicate request. Refer to Section 14.
65555	Settlement error.
65556	AVS/CV2 Checks are not supported for this card (or Acquirer)
65557	IP Blocked: Request is from a banned IP address
65558	Primary IP blocked: Request is not from one of the primary IP addresses configured for this Merchant Account
65559	Secondary IP blocked: Request is not from one of the secondary IP addresses configured for this Merchant Account
65560	Reserved for future use. Contact customer support if this error occurs
65561	Unsupported Card Type: Request is for a card type that is not supported on this Merchant Account
65562	Unsupported Authorisation: External authorisation code `authCode` has been supplied and this is not supported for the transaction or by the Acquirer
65563	Request not supported: The Gateway or Acquirer does not support the request
65564	Request expired: The request cannot be completed as the information is too old
65565	Request retry: The request can be retried later
65566	Test Card Used: A test card was used on a live Merchant Account
65567	Unsupported card issuing country: Request is for a card issuing country that is not supported on this Merchant Account
65568	Unsupported payment type: Request uses a payment type which is not supported on this Merchant Account

3-D Secure Error Codes: 65792 - 66047	
Code	**Description**
65792	3-D Secure transaction in progress. Contact customer support if this error occurs
65793	Unknown 3-D Secure Error
65794	3-D Secure processing is required for the given card
65795	3-D Secure processing is not required for the given card
65796	3-D Secure processing is unavailable. Merchant account doesn't support 3-D Secure.
65797	Error occurred during 3-D Secure enrolment check
65798	Reserved for future use. Contact customer support if this error occurs
65799	Reserved for future use. Contact customer support if this error occurs
65800	Error occurred during 3-D Secure authentication check
65801	Reserved for future use. Contact customer support if this error occurs
65802	3-D Secure authentication is required for this card
65803	3-D Secure enrolment or authentication failure and Merchant 3-D Secure preferences are to STOP processing

Missing Request Field Error Codes: 66048 - 66303	
Code	**Description**
66048	Missing request. No data posted to integration URL
66049	Missing `merchantID` field
66050	Reserved for future use. Contact customer support if this error occurs
66051	Reserved for internal use. Contact customer support if this error occurs
66052	Reserved for internal use. Contact customer support if this error occurs
66053	Reserved for internal use. Contact customer support if this error occurs
66054	Reserved for internal use. Contact customer support if this error occurs
66055	Missing `action` field
66056	Missing `amount` field
66057	Missing `currencyCode` field
66058	Missing `cardNumber` field
66059	Missing `cardExpiryMonth` field

Missing Request Field Error Codes: 66048 - 66303	
Code	**Description**
66060	Missing `cardExpiryYear` field
66061	Missing `cardStartMonth` field (reserved for future use)
66062	Missing `cardStartYear` field (reserved for future use)
66063	Missing `cardIssueNumber` field (reserved for future use)
66064	Missing `cardCVV` field
66065	Missing `customerName` field
66066	Missing `customerAddress` field
66067	Missing `customerPostCode` field
66068	Missing `customerEmail` field
66069	Missing `customerPhone` field (reserved for future use)
66070	Missing `countyCode` field
66071	Missing `transactionUnique` field (reserved for future use)
66072	Missing `orderRef` field (reserved for future use)
66073	Missing `remoteAddress` field (reserved for future use)
66074	Missing `redirectURL` field
66075	Missing `callbackURL` field (reserved for future use)
66076	Missing `merchantData` field (reserved for future use)
66077	Missing `origin` field (reserved for future use)
66078	Missing `duplicateDelay` field (reserved for future use)
66079	Missing `itemQuantity` field (reserved for future use)
66080	Missing `itemDescription` field (reserved for future use)
66081	Missing `itemGrossValue` field (reserved for future use)
66082	Missing `taxValue` field (reserved for future use)
66083	Missing `discountValue` field (reserved for future use)
66084	Missing `taxDiscountDescription` field (reserved for future use)
66085	Missing `xref` field (reserved for future use)

Missing Request Field Error Codes: 66048 - 66303	
Code	**Description**
66086	Missing **type** field. The transaction type is missing. Possible values are: **1** – E-commerce (ECOM) **2** – Mail Order/Telephone Order (MOTO). **9** – Continuous Authority (CA).
66087	Missing **signature** field (field is required if message signing is enabled)
66088	Missing **authorisationCode** field (reserved for future use)
66089	Missing **transactionID** field (reserved for future use)
66090	Missing **threeDSRequired** field (reserved for future use)
66091	Missing **threeDSMD** field (reserved for future use)
66092	Missing **threeDSPaRes** field
66093	Missing **threeDSECI** field
66094	Missing **threeDSCAVV** field
66095	Missing **threeDSXID** field
66096	Missing **threeDSEnrolled** field
66097	Missing **threeDSAuthenticated** field
66098	Missing **threeDSCheckPref** field
66099	Missing **cv2CheckPref** field
66100	Missing **addressCheckPref** field
66101	Missing **postcodeCheckPref** field
66102	Missing **captureDelay** field
66103	Missing **orderDate** field
66104	Missing **grossAmount** field
66105	Missing **netAmount** field
66016	Missing **taxRate** field
66016	Missing **taxReason** field
66160	Missing **cardExpiryDate** field
66161	Missing **cardStartDate** field

Invalid Request Field Error Codes: 66304 - 66559	
Code	**Description**
66304	Invalid request
66305	Invalid `merchantID` field
66306	Reserved for future use. Contact customer support if this error occurs
66307	Reserved for internal use. Contact customer support if this error occurs
66308	Reserved for internal use. Contact customer support if this error occurs
66309	Reserved for internal use. Contact customer support if this error occurs
66310	Reserved for internal use. Contact customer support if this error occurs
66311	Invalid `action` field
66312	Invalid `amount` field
66313	Invalid `currencyCode` field
66314	Invalid `cardNumber` field
66315	Invalid `cardExpiryMonth` field
66316	Invalid `cardExpiryYear` field
66317	Invalid `cardStartMonth` field
66318	Invalid `cardStartYear` field
66319	Invalid `cardIssueNumber` field
66320	Invalid `cardCVV` field
66321	Invalid `customerName` field
66322	Invalid `customerAddress` field
66323	Invalid `customerPostCode` field
66324	Invalid `customerEmail` field
66325	Invalid `customerPhone` field
66326	Invalid `countyCode` field
66327	Invalid `transactionUnique` field (reserved for future use)
66328	Invalid `orderRef` field (reserved for future use)
66329	Invalid `remoteAddress` field

Invalid Request Field Error Codes: 66304 - 66559	
Code	**Description**
66330	Invalid `redirectURL` field
66331	Invalid `callbackURL` field (reserved for future use)
66332	Invalid `merchantData` field (reserved for future use)
66333	Invalid `origin` field (reserved for future use)
66334	Invalid `duplicateDelay` field. Refer to Section 14.
66335	Invalid `itemQuantity` field
66336	Invalid `itemDescription` field
66337	Invalid `itemGrossValue` field
66338	Invalid `taxValue` field
66339	Invalid `discountValue` field
66340	Invalid `taxDiscountDescription` field (reserved for future use)
66341	Invalid `xref` field
66342	Invalid `type` field
66343	Invalid `signature` field
66344	Invalid `authorisationCode` field
66345	Invalid `transactionID` field
66356	Invalid `threeDSRequired` field
66347	Invalid `threeDSMD` field
66348	Invalid `threeDSPaRes` field
66349	Invalid `threeDSECI` field
66350	Invalid `threeDSCAVV` field
66351	Invalid `threeDSXID` field
66352	Invalid `threeDSEnrolled` field
66353	Invalid `threeDSAuthenticated` field
66354	Invalid `threeDSCheckPref` field
66355	Invalid `cv2CheckPref` field

Invalid Request Field Error Codes: 66304 - 66559	
Code	**Description**
66356	Invalid `addressCheckPref` field
66357	Invalid `postcodeCheckPref` field
66358	Invalid `captureDelay` field.
66359	Invalid `orderDate` field
66360	Invalid `grossAmount` field
66361	Invalid `netAmount` field
66362	Invalid `taxRate` field
66363	Invalid `taxReason` field
66416	Invalid card expiry date. Must be a date sometime in the next 10 years
66417	Invalid card start date. Must be a date sometime in the last 10 years

A-2 AVS / CV2 Check Response Codes

The AVS/CV2 Check Response Message field **`avscv2ResponseMessage`** is sent back in the raw form that is received from the Acquiring bank and can contain the following values:

Response	Description
ALL MATCH	AVS and CV2 match
SECURITY CODE MATCH ONLY	CV2 match only
ADDRESS MATCH ONLY	AVS match only
NO DATA MATCHES	No matches for AVS and CV2
DATA NOT CHECKED	Supplied data not checked
SECURITY CHECKS NOT SUPPORTED	Card scheme does not support checks

The AVS/CV2 Response Code **`avscv2ResponseCode`** is made up of six characters and is sent back in the raw form that is received from the Acquiring bank. The first 4 characters can be decoded as below, the remaining 2 characters are currently reserved for future use:

Position 1 Value	Description
0	No Additional information available.
1	CV2 not checked
2	CV2 matched.
4	CV2 not matched
8	Reserved

Position 2 Value	Description
0	No Additional information available.
1	Postcode not checked
2	Postcode matched.
4	Postcode not matched
8	Postcode partially matched

Position 3 Value	Description
0	No Additional Information
1	Address numeric not checked
2	Address numeric matched
4	Address numeric not matched
8	Address numeric partially matched

Position 4 Value	Description
0	Authorising entity not known
1	Authorising entity – merchant host
2	Authorising entity – acquirer host
4	Authorising entity – card scheme
8	Authorising entity – issuer

blink

A-3 3-D Secure Enrolment/Authentication Codes

The 3-D Secure enrolment check field **`threeDSEnrolled`** can return the following values:

Y - Enrolled: The card is enrolled in the 3-D Secure program and the payer is eligible for authentication processing.

N - Not Enrolled: The checked card is eligible for the 3-D Secure (it is within the card association's range of accepted cards) but the card issuing bank does not participate in the 3-D Secure program. If the Cardholder later disputes the purchase, the issuer may not submit a chargeback to you.

U - Unable To Verify Enrolment: The card associations were unable to verify whether the Cardholder is registered. As the card is ineligible for 3-D Secure, Merchants can choose to accept the card nonetheless and precede the purchase as non-authenticated and submits authorisation with ECI 7. The Acquirer/Merchant retains liability if the Cardholder later disputes making the purchase. *(v1 only)*

E - Error Verify Enrolment: The Gateway encountered an error. This card is flagged as 3-D Secure ineligible. The card can be accepted for payment, yet you may not claim a liability shift on this transaction in case of a dispute with the Cardholder. *(v1 only)*

The 3-D Secure authentication check field **`threeDSAuthenticated`** can return the following values:

Y - Authentication Successful: The Issuer has authenticated the Cardholder by verifying the identity information or password. A CAVV and an ECI of 5 is returned. The card is accepted for payment.

N - Not Authenticated: The Cardholder did not complete authentication and the card should not be accepted for payment.

U - Unable To Authenticate: The authentication was not completed due to technical or another problem. A transmission error prevented authentication from completing. The card should be accepted for payment, but no authentication data will be passed on to authorisation processing and no liability shift will occur.

A - Attempted Authentication: A proof of authentication attempt was generated. The Cardholder is not participating, but the attempt to authenticate was recorded. The card should be accepted for payment and authentication information passed to authorisation processing.

E - Error Checking Authentication: The Gateway encountered an error. The card should be accepted for payment, but no authentication information will be passed to authorisation processing and no liability shift will occur.

R – Authentication rejected by Issuer: Authentication/account verification rejected by the Issuer. *(v2 only)*

D – Decoupled challenge required: Challenge required; decoupled authentication confirmed. *(v2 only)*

C – Challenge required: Additional authentication is required using a challenge. *(v2 only)*

I – Challenge preference acknowledged: Information only. 3DS Requestor challenge preference acknowledged. *(v2 only)*

A-4 3-D Secure Enrolment/Authentication Only

Usually, the Gateway will perform most of the 3-D Secure processing in the background leaving the only the actual contacting of the issuers Access Control Server (ACS) to the Merchant.

However, there may be times when you may wish to gain more control over the Enrolment and Authentication process. The following field allows the request processing to stop after the 3-D Secure enrolment check or authentication check and return:

Field Name	Mandatory?	Description
`threeDSOnly`	No	Complete the processing as far as the next 3-D Secure stage and then return with the appropriate response fields for that stage.

As this stop is requested then a **`responseCode`** is returned as **0 (Success)** however it will be recorded in the Merchant Management System (MMS) as **65792 (3DS IN PROGRESS)** indicating that the transaction has been prematurely halted expecting it to be continued to the next 3-D Secure stage when required. In order to continue the process, the **`threeDSRef`** field is returned together with any relevant 3-D Secure response fields suitable for that stage in the processing.

If this flag is used when 3-D Secure is not enabled on the account or after the 3-D Secure process has been completed for the request (i.e. when the authentication step has completed), then passing the flag will cause the transaction to abort with a **`responseCode`** of **65795 (3DS PROCESSING NOT REQUIRED)**. This ensures that the transaction does not go on to completion by accident while trying do 3-D Secure enrolment or authentication only.

3-D Secure Enrolment/Authentication Only is supported by the Direct Integration only.

blink

A-5 SCA Exemptions

There are several exemptions to SCA that may be requested:

Low Value Exemption
Transactions below 30 EUR are considered low value and are generally exempt from authentication. However, the velocity limits below must be met:

- The cumulative limit of consecutive transactions without the application of SCA must not exceed 100 EUR: or
- The number of consecutive transactions since the last application of SCA must not exceed five.

The Merchant can request this exemption, or it may be automatically applied by the Issuer.

Trusted Beneficiary Exemption
The payer may add a trusted Merchant to a whitelist of trusted beneficiaries held by their Issuer, completing an SCA challenge in the process, to prevent further SCA application on subsequent transactions with the trusted Merchant.

The Merchant can request this exemption to allow this trust to be taken into consideration.

Trusted Risk Analysis (TRA) Exemption
SCA is not mandated where a PSP, having in place effective risk analysis tools, assesses that the fraud risk associated with a remote payment transaction is low. The ability for a payment to be considered low risk is based on the average fraud levels of the card issuer and acquirer processing the transaction.

The Merchant can request this exemption if agreed to by the PSP.

Secured Corporate Payment Exemption
Transactions initiated by a business rather than a Consumer and processed through a secured dedicated payment protocol can be exempt from SCA provided alternative controls are sufficiently secure.

The Merchant can request this transaction to indicate such a secure transaction.

Delegated Authentication Exemption
If the Merchant already requires the Consumer to perform sufficient authentication on their website, such as secure account logins etc., then they can use this exemption to request that further SCA is not required.

A-6 3-D Secure Legacy API

A 6.1 Background

The 3-D Secure Legacy API was used for 3-D Secure integration prior to the introduction of 3-D Secure version 2. The API details are documented for backwards compatibility and should not be used for new integrations.

A 6.2 Direct Implementation

If your Merchant account is setup for 3-D Secure the Gateway will require further authentication details provided by the 3-D Secure system.

To ensure the Gateway uses the legacy 3-D Secure API you must not pass any of the following fields used by the current API:

threeDSRef
threeDSRedirectURL
threeDSURL
threeDSRequest
threeDSResponse

The API versions must not be mixed between the initial and continuation requests otherwise an error will occur.

A-6.2.1 Initial Request (Direct Integration)

If no 3-D Secure authentication details are provided in the initial request the Gateway will determine whether the transaction is eligible for 3-D Secure by checking whether the card is enrolled in the 3-D Secure scheme.

If the Gateway determines that the transaction is not eligible for 3-D Secure, then it will continue and process it as normal transaction without 3-D Secure unless the **`threeDSRequired`** request field indicates that the transaction should be aborted instead.

If the Gateway determines that the transaction is eligible it will respond with a **`responseCode`** of **65802 (3DS AUTHENTICATION REQUIRED)** and included in the response will be a **`threeDSACSURL`** field containing the URL required to contact the ACS on and a **`threeDSMD`** and **`threeDSPaReq`** to send to the provided URL. The latter two values must be posted to the provided ACS URL as the fields **MD** and **PaReq** together with a **TermUrl** field provided by yourself which must contain the URL of a page on your server to return to when authentication has been completed.

A-6.2.2 Continuation Request (Direct Integration)

On completion of the 3-D Secure authentication the ACS will post the original **MD** together with a **PaRes** value to the **TermUrl** provided. These values should then be sent to the Gateway in the **`threeDSMD`** and **`threeDSPaRes`** fields of a new request. This new request will check the 3-D Secure authentication and then either complete or abort the transaction depending on the authentication result and your preferences, either sent in the **`threeDSPref`** field or set in the Merchant Management System (MMS).

A-6.2.3 Initial Request (Direct Integration)

These fields should be sent in addition to basic request fields in section 2.1 of the main integration guide.

Field Name	Mandatory?	Description
merchantName	No[1]	Merchant name to use on 3DS form.
merchantWebsite	No[1]	Merchant website to use on 3DS form. The website must be a fully qualified URL and include at least the scheme and host components.
threeDSRequired	No[2]	Is 3DS required for this transaction? Possible values are: **N** – 3DS is not required. **Y** – Abort if 3DS is not enabled.
threeDSCheckPref	No[1]	List of **threeDSCheck** response values that are to be accepted, any other value will cause the transaction to be declined. Value is a comma separated list containing one or more of the following values: 'not known', 'not **checked', ' not authenticated', 'attempted authentication**', 'authenticated' .

[1] If the value is not supplied, then the default account preferences will be used.
[2] The default value is **Y** if 3-D Secure is enabled on the Merchant Account, otherwise **N.**

A-6.2.4 Continuation Request (Direct Integration)

These fields may be sent alone[1].

Field Name	Mandatory?	Description
threeDSMD	Yes	The value of the **threeDSMD** field in the initial Gateway response.
threeDSPaRes	Yes	The value of the PaRes field POSTed back from the Access Control Sever (ACS).

[1] It is only necessary to send the `threeDSMD` and the `threeDSPaRes` in the continuation request as the `threeDSMD` will identify the Merchant Account and initial request. The message does not need to be signed. However, you can send any of the normal request fields to modify or supplement the initial request. Any card details and transaction amount sent in the second request must match those used in the first request, or the second request will fail with a `responseCode` of **64442 (REQUEST MISMATCH).**

A-6.3.1 Initial Response (Direct Integration)

These fields should be sent in addition to the basic request fields in 2.2 of the main integration guide.

Field Name	Returned?	Description
threeDSEnabled	Always	Is 3DS enabled for this Merchant Account? Possible values are: **N** – Merchant Account is not enabled. **Y** – Merchant Account is enabled.
threeDSXID	If 3DS enabled	The unique identifier for the transaction in the 3DS system.
threeDSVETimestamp	If 3DS enabled	The time the card was checked for 3DS enrolment.
threeDSEnrolled	If 3DS enabled	The 3DS enrolment status for the credit card. Refer to appendix A-3 for details. Possible values are: **Y** – Enrolled. **N** - Not Enrolled. **U** - Unable to Verify. (v1 only) **E** - Error Verifying Enrolment. (v1 only)
threeDSMD	If 3DS enabled	Value to return in the continuation request. Can be sent to the Access Control Server (ACS) in its MD field or stored locally by your server.
threeDSACSURL	If 3DS enrolled	The URL of the Access Control Server (ACS) to which the Payer Authentication Request (PaReq) should be sent.
threeDSPaReq	If 3DS enrolled	Payer Authentication Request (PaReq) that is sent to the Access Control Server (ACS) in order to verify the 3DS status of the credit card.

A-6.3.2 Continuation Response (Direct Integration)

These fields will be returned in addition to the request fields, the initial response fields and the basic response fields in section 2.2 of this guide.

Field Name	Returned?	Description
`threeDSPaRes`	If 3DS enrolled	Payer Authentication Response (PaRes) that is returned from the Access Control Server (ACS) determining the 3DS status of the credit card.
`threeDSCATimestamp`	If 3DS enrolled	The time the card was checked for 3DS authentication.
`threeDSAuthenticated`	If 3DS enrolled	The 3DS authentication status for the credit card. Refer to appendix A-3 for details. Possible values are: **Y** - Authentication Successful. **N** - Not Authenticated. **U** - Unable to Authenticate. **A** - Attempted Authentication. **E** - Error Checking Authentication. For 3DS version 2.2 only: **R** – Authentication rejected by Issuer. **C** – Challenge required. **D** – Decoupled challenge required. **I** – Acknowledges request not to challenge cardholder.
`threeDSECI`	If 3DS authenticated	This contains a two-digit Electronic Commerce Indicator (ECI) value, which is to be submitted in a credit card authorisation message. This value indicates to the processor that the Customer data in the authorisation message has been authenticated. The data contained within this property is only valid if the `threeDSAuthenticated` value is **Y** or **A**.
`threeDSCAVV`	If 3DS authenticated	This contains a 28-byte Base-64 encoded Cardholder Authentication Verification Value (CAVV). The data contained within this property is only valid if the `threeDSAuthenticated` value is **Y** or **A**.
`threeDSCAVVAlgorithm`	If 3DS authenticated	This contains the one digit value that indicates the algorithm used by the Access Control Server (ACS) to generate the CAVV. The data contained within this property is only valid if the `threeDSAuthenticated` value is **Y** or **A**.

Field Name	Returned?	Description
`threeDSErrorCode`	If 3DS error	Any error response code returned by the Access Control Server (ACS) if there is an error in determining the card's 3DS status.
`threeDSErrorDescription`	If 3DS error	Any error response description returned by the Access Control Server (ACS) if there is an error in determining the card's 3DS status.

A-6.4 3-D Secure Enrolment/Authentication Only (Direct Only)

3-D Secure Enrolment/Authentication checks can be performed using the legacy 3-D Secure API, except that the Gateway will return a **`threeDSMD`** field instead of a **`threeDSRef`** field. This field can be passed in the next request to continue the transaction.

A-7 Request Checking Only

Sometimes, you may wish to submit a request to the Gateway in order for it to be 'validated only' and not processed or sent to the Acquirer. In these cases, the following flag can be used that will stop the processing after the integrity verification has been performed:

Field Name	Mandatory?	Description
`checkOnly`	No	Check the request for syntax and field value errors only. Do not attempt to submit the transaction for honouring by the Merchant's financial institution.

If the request is OK, then a `responseCode` is returned as **0 (Success)**; otherwise the code that would have prevented the request from completing is returned.

Note: *in these cases, the request is not stored by the Gateway and is not available within the Merchant Management System (MMS).*

blink

A-8 Merchant Account Mapping

Merchant Accounts can be grouped together so that if a transaction is sent to an account that doesn't support either the requested card type or currency, then it can be automatically routed to another account in the same group that does support them.

For example, you can group a Merchant Account that only supports American Express cards with a Merchant Account that only supports Visa cards. Then, if a request using an American Express card is sent to the Visa only Merchant Account, the Gateway will automatically route it to the American Express Merchant Account.

This prevents you from needing to know the card type in advance in order to send the request to the correct Merchant Account. This is important when using the Hosted integration, because you don't know the card type at the time you send the request.

It is usual for you to have one master account to which you direct all requests and then group all your accounts together.

Any Gateway routing of the transaction can be seen from the following additional response fields:

Field Name	Returned?	Description
`requestMerchantID`	Always	ID of Merchant Account request was sent to (usually same as `merchantID`).
`processMerchantID`	Always	ID of Merchant Account request was processed by.

A-9 Velocity Control System (VCS)

The Gateway allows you to configure velocity controls using the Merchant Management System (MMS). These can be used to email you declined transactions automatically, where they exceed these controls.

For example, you can set up a control that stops a certain card number from being used more than twice in the space of a few minutes.

If one or more of these controls are broken by a transaction, then the following response fields will show the problem.

If a transaction is declined through breach of one or more of these rules, then a **responseCode** of **5 (VCS DECLINE)** will be returned.

Field Name	Returned?	Description
`vcsResponseCode`	Always	VCS error code. Normally **5**. Refer to appendix A-1 for details.
`vcsResponseMessage`	Always	Description of above response code or list of rules broken by this transaction.

blink

A-10 Capture Delay

Capture Delay enables you to specify a delay between the authorisation of a payment and its capture. This allows you time to verify the order and choose whether to fulfil it or cancel it. This can be very helpful in preventing chargebacks due to fraud.

When NOT using capture delay, payments are authorised and captured immediately - funds are automatically debited from the Customer's credit or debit card at that time.

When using capture delay, the payment is authorised only at the time of payment - funds are reserved against the credit or debit card and will not be debited until the payment is captured; or not at all if you cancel.

The Customer experience with capture delay is exactly the same as when capture delay is not used. The Customer will not know whether you are using capture delay or not.

If you choose to use capture delay, you specify the number of days for which capture is delayed, within a range of 0 - 30 days. Payments will automatically be captured after that delay unless you manually cancel the transaction (either using the Hosted Integration or via the Merchant Management System (MMS)). (Note that some cards require capture within 4-5 days - if payment is not automatically captured within that 4-5 day period, the transaction will expire and the reserved funds will be released to the Customer.)

Why Use Capture Delay?

Capture delay allows you to accept online orders normally but allows you to cancel any transactions that you cannot or will not fulfil, thereby reducing the risks of chargeback. If you receive an order that appears to be fraudulent or that you cannot or do not wish to fulfil, you can simply cancel the transaction.

Note: Cancelling a transaction will not reverse the authorisation and will not release the funds back to the Customer. The authorisation will be left to expire and release reserved funds. The time taken for this varies between cards.

Some Acquirers do not support delayed capture, in which case the Hosted Integration will return a `responseCode` *of* ***66358 (INVALID CAPTURE DELAY)****.*

blink

A-11 Types of card

The following is a list of primary card types supported by the Gateway.

Card Code	Card Type
MC	Mastercard Credit
MD	Mastercard Debit
MA	Mastercard International Maestro
MI	Mastercard/Diners Club
MP	Mastercard Purchasing
MU	Mastercard Domestic Maestro (UK)
VC	Visa Credit
VD	Visa Debt
EL	Visa Electron
VA	Visa ATM
VP	Visa Purchasing
AM	American Express
JC	JCB
CU	China UnionPay (generic)
CC	China UnionPay Credit
CD	China UnionPay Debit

The Gateway primarily supports Mastercard, Visa and American Express branded cards. Some Acquirers may support JCB cards. Not all Acquirers support all types.

Where cards are provided by a single card scheme, then the primary card code is also used as a code to identify the card scheme (referred to as the `cardSchemeCode` in the transaction response). For example, cards issued by VISA will use the code '**VC**'; cards issued by Mastercard will use the code '**MC**'; and so on. China UnionPay credit '**CC**' and debit '**CD**' will use the scheme code '**CU**'.

The following is a list of secondary card types recognised by the Gateway.

Card Code	Card Type
CF	Clydesdale Financial Services
BC	BankCard
DK	Dankort
DS	Discover
DI	Diners Club
DE	Diners Club Enroute
DC	Diners Club Carte Blanche
FC	FlexCache
LS	Laser
SO	Solo
ST	Style
SW	Switch
TP	Tempo Payments
IP	InstaPayment
XX	Unknown/unrecognised card type

These cards may be returned in response to a card lookup, but they are either deprecated or most likely not supported by any current Acquirer.

blink

A-12 Integration Testing

You can perform test transactions using one of the test Merchant IDs below and using test card details.

For non 3-D Secure testing use Merchant ID **136784**
For 3-D Secure Testing use Merchant ID **135828**

Test Merchant Accounts are not connected to an Acquirer and for that reason simulate their response, depending on the request **amount**, as follows:

Amount range from	Amount range to	Expected authorisation response	Expected settlement outcome
100 (£1.00)	2499 (£24.99)	AUTH CODE: XXXXXX (0)	ACCEPTED
2500 (£25.00)	4999 (£49.99)	AUTH CODE: XXXXXX (0)	REJECTED
5000 (£50.00)	9999 (£99.99)	CARD REFERRED (1)	N/A
10000 (£100.00)	14999 (£149.99)	CARD DECLINED (5)	N/A
15000 (£150.00)	19999 (£199.99)	CARD DECLINED – KEEP CARD (4)	N/A
20000 (£200.00)	24999 (£249.99)	CARD DECLINED – SCA REQUIRED (65) AUTH CODE XXXXX (0)	N/A
25000 (£250.00)	29999 (£299.99)	CARD DECLINED – SCA REQUIRED (65) CARD DECLINED (5)	N/A

Any other amount will return a **`responseCode`** of **66311 (Invalid Test Amount)**.

The range 20000 to 29999 can be used to test SCA soft declines. If the transaction is eligible[1] to request SCA then the Simulator will return a **`responseCode`** of **65 (SCA REQUIRED**). If not, then it will return a **`responseCode`** of **0 (SUCCESS)** for the range 20000 to 24999 or **5 (DO NOT HONOR)** for the range 25000 to 29999**.**

[1] A cardholder-initiated ecommerce sale or verify transaction that is enabled for 3-D Secure but is not already authenticated. SCA exemptions are not supported by the simulator and so cannot be used to request that SCA is not required.

A-12.1 Test Card Details

DO NOT USE THESE TEST CARDS ON LIVE MERCHANT ACCOUNTS. THEY ARE FOR TEST PURPOSES ONLY.

The expiry date used for each test card should be December of the current year, in two-digit format.

A-12.1.1 Visa Credit

Card Number	CVV	Address
4929 4212 3460 0821	356	Flat 6 Primrose Rise 347 Lavender Road Northampton NN17 8YG
4543 0599 9999 9982	110	76 Roseby Avenue Manchester M63X 7TH
4543 0599 9999 9990	689	23 Rogerham Mansions 4578 Ermine Street Borehamwood WD54 8TH

A-12.1.2 Visa Debit

Card Number	CVV	Address
4539 7910 0173 0106	289	Unit 5 Pickwick Walk 120 Uxbridge Road Hatch End Middlesex HA6 7HJ
4462 0000 0000 0003	672	Mews 57 Ladybird Drive Denmark 65890

A-12.1.3 Mastercard Credit

Card Number	CVV	Address
5301 2500 7000 0191	419	25 The Larches Narborough Leicester LE10 2RT
5413 3390 0000 1000	304	Pear Tree Cottage

		The Green Milton Keynes MK11 7UY
5434 8499 9999 9951	470	34a Rubbery Close Cloisters Run Rugby CV21 8JT
5434 8499 9999 9993	557	4-7 The Hay Market Grantham NG32 4HG

A-12.1.4 Mastercard Debit

Card Number	CVV	Address
5573 4712 3456 7898	159	Merevale Avenue Leicester LE10 2BU

blink

UK Maestro

Card Number	CVV	Address
6759 0150 5012 3445 002	309	The Parkway 5258 Larches Approach Hull North Humberside HU10 5OP
6759 0168 0000 0120 097	701	The Manor Wolvey Road Middlesex TW7 9FF

JCB

Card Number	CVV	Address
3540 5999 9999 1047	209	2 Middle Wallop Merideth-in-the-Wolds Lincolnshire LN2 8HG

Electron

Card Number	CVV	Address
4917 4800 0000 0008	009	5-6 Ross Avenue Birmingham B67 8UJ

American Express

Card Number	CVV	Address
3742 4545 5400 001	4887	The Hunts Way Southampton SO18 1GW

Diners Club

Card Number	CVV Number
3643 2685 2602 94	**111**

Diners Club do not support AVS. For testing purposes use a separate MID with AVS turned off.

blink

Visa Test Cards

Card Number	CVV	Address	Postcode	Amount	Test Scenario
4909 6300 0000 0008				£12.01	Card range not participating
4012 0100 0000 0000 009				£12.02	Card registered with VbV (automated ACS response – click on Submit button)
4012 0010 3714 1112	083	16	155	£12.03	Card registered with Visa (automated ACS response – click on Submit button)
4012 0010 3748 4447	450	200	19	£12.04	Failed authentication – issuer database unavailable
4015 5011 5000 0216				£12.05	Attempts processing (automated ACS response – click on Submit button)

blink

Mastercard Test Cards

Note: These test cards are controlled by Mastercard and won't always act as expected. The 3-D Secure passwords can be changed by anyone during the 3-D Secure testing, which means that the password won't then work for the next person. The standard fall-back password is dog33cat. Use Visa's 3-D Secure test cards if these below are not behaving as expected.

Card Number	CVV	Address	Postcode	Amount	Test Scenario
5033 9619 8900 0008 18	332	31	18	£11.01	Enrolled International Maestro account number – valid SecureCode (multiple cardholder). Select 'MEGAN SANDERS' with SecureCode password: secmegan1
5453 0100 0007 0789	508	20	52	£11.02	Enrolled account number - valid SecureCode (single) SecureCode password: sechal1
5453 0100 0007 0151	972	22	08	£11.03	Enrolled account number – mixed SecureCode (multi) SecureCode password: Hannah – sechannah1 (bad) Haley – sechaley1 (good)
5453 0100 0007 0284	305	35	232	£11.04	Enrolled account number – invalid SecureCode Invalid SecureCode password: invseccode
5453 0100 0008 4103	470	73	170	£11.05	Attempts processing
5453 0100 0007 0888	233	1	248	£11.06	Account number not enrolled
5199 9923 1264 1465	006	21	14	£11.07	Card range not participating

blink

A-12.2 3-D Secure Testing

You test accounts are connected to our 3-D Secure Product Integration Testing (PIT) system rather than to the production 3-D Secure servers.

You can use any of the test cards provided in section A -12.1 with this PIT system and can test various enrolment and authentication scenarios as follows.

A-12.2.1 3-D Secure version 1

For 3-D Secure v1 all the standard test card numbers will show as enrolled except for:

Card Number	Enrolment	Simulation
4012 0010 3844 3335	**N**	Unenrolled card
4012 0010 3848 8884	**U**	Unknown enrolment status
4012 0010 3627 5556	**E**	Error due timeout communicating with the Directory Server
4012 0010 3629 8889	**E**	Error due to corrupt response from the Directory Server

The desired authentication status (**`threeDSAuthenticated`**) can be selected on the challenge dialog shown by the PIT Access Control Server.

A-12.2.2 3-D Secure version 2

For 3-D Secure v2 all the standard test cards will show as enrolled, and the authentication status returned by the Directory Server (for frictionless flow simulation) can be selected using the value of the card expiry month as follows:

Card Expiry Month	Auth Status	Simulation (Frictionless)
01 - Jan	`Y`	Fully authenticated
02 - February	`N`	Not authenticated
03 - March	`U`	Unknown authentication status
04 - April	`A`	Attempted authentication
05 - May	`D`	Decoupled authentication
06 - June	`R`	Transaction rejected (do not attempt to send for authorisation)
07 – July	`E`	Unknown error performing 3-D Secure checks
08 - August	`E`	Error due to timeout communicating with the Directory Server
09 – September	`E`	Error due to corrupt response from the Directory Server.
10 – October	`I`	Information only
11 – November	`U`	Unknown authentication due to Cardholder not enrolled (error 13)
12 - December	`C`	Frictionless not possible, challenge Cardholder

If the month required has passed for the current year the card will show as expired. Use the month required and the next years date to receive the expected response.

An expiry month of 12 will simulate the non frictionless flow and desired authentication status (**`threeDSAuthenticated`**) can be selected on the challenge dialog shown by the PIT Access Control Server.

A-12.3 PayPal Sandbox Accounts

PayPal testing is available on the standard **100001** test Merchant account. However, you may wish to contact customer support to have your own PayPal test Merchant account created that connects to your own PayPal sandbox account, thus enabling you to view the transactions as they are sent to PayPal.

A-12.4 Amazon Pay Sandbox Accounts

Amazon Pay testing is available on the standard **100001** test Merchant account. However, you may wish to contact customer support to have your own Amazon Pay test Merchant account created that connects to your own Amazon Pay sandbox account, thus enabling you to view the transactions as they are sent to Amazon Pay.

blink

A-13 Sample Signature Calculation

It is required that transactions are protected using message signing. The signing process offers a quick and simple way to ensure that the message came from an authorised source and has not been tampered with during transmission.

Signing, however, must be completed on your servers and never left for the Customer's browser to complete in JavaScript, as this would mean revealing your secret signature code to anyone who viewed the JavaScript code in the browser.

Signatures are especially important when a transaction is sent from a browser's payment form via the use of hidden form fields, because the Customer can easily use tools built into their browser to modify these hidden fields and change items such as the amount they should be charged.

The section below gives a step by step example of how to sign a transaction, complete with coding examples using the PHP language.

Example Signature Key:

```
$key = 'DontTellAnyone'
```

Example Transaction:

```
$tran = array (
  'merchantID' => '100001',
  'action' => 'SALE',
  'type' => '1',
  'currencyCode' => '826',
  'countryCode' => '826',
  'amount' => '2691',
  'transactionUnique' => '55f025addd3c2',
  'orderRef' => 'Signature Test',
  'cardNumber' => '4929 4212 3460 0821',
  'cardExpiryDate' => '1213',
)
```

The transaction used for signature calculation must not include any 'signature' field as this will be added after signing when its value is known.

blink

Step 1 - Sort transaction values by their field name

Transaction fields must be in ascending field name order according to their numeric ASCII value.

```
ksort($tran);
```

```
array ( 'action' => 'SALE', 'amount' => '2691', 'cardExpiryDate' => '1213',
'cardNumber' => '4929 4212 3460 0821', 'countryCode' => '826', 'currencyCode' => '826',
'merchantID' => '100001', 'orderRef' => 'Signature Test', 'transactionUnique' =>
'55f025addd3c2', 'type' => '1' )
```

Step 2 - Create url encoded string from sorted fields

Use RFC 1738 and the application/x-www-form-urlencoded media type, which implies that spaces are encoded as plus (+) signs.

```
$str = http_build_query($tran, '', '&');
```

```
action=SALE&amount=2691&cardExpiryDate=1213&cardNumber=4929+4212+3460+0821&countryCode=
826&currencyCode=826&merchantID=100001&orderRef=Signature+Test&transactionUnique=55f025
addd3c2&type=1
```

Step 3 - Normalise all line endings in the url encoded string

Convert all CR NL, NL CR, CR character sequences to a single NL character.

```
$str = str_replace(array('%0D%0A', '%0A%0D', '%0D'), '%0A', $str);
```

```
action=SALE&amount=2691&cardExpiryDate=1213&cardNumber=4929+4212+3460+0821&countryCode=
826&currencyCode=826&merchantID=100001&orderRef=Signature+Test&transactionUnique=55f025
addd3c2&type=1
```

Step 4 - Append your signature key to the normalised string

The signature key is appended to the normalised string with no separator characters.

```
$str .= 'DontTellAnyone'
```

```
action=SALE&amount=2691&cardExpiryDate=1213&cardNumber=4929+4212+3460+0821&countryCode=
826&currencyCode=826&merchantID=100001&orderRef=Signature+Test&transactionUnique=55f025
addd3c2&type=1DontTellAnyone
```

Step 5 - Hash the string using the SHA-512 algorithm

The normalised string is hashed to a more compact value using the secure SHA-512 hashing algorithm[1].

```
$signature = hash('SHA512', $str);
```

```
da0acd2c404945365d0e7ae74ad32d57c561e9b942f6bdb7e3dda49a08fcddf74fe6af6b23b8481b8dc8895
c12fc21c72c69d60f137fdf574720363e33d94097
```

Step 6 - Add the signature to the transaction form or post data

The signature should be sent as part of the transaction in a field called 'signature'.

```
<input type="hidden" name="signature" value="<?=$signature?>">
```

or

```
$tran['signature'] = $signature;
```

A-14 Transaction Life cycle

Each transaction received by the Gateway follows a pre-determined life cycle from receipt to completion. The stages in the life cycle are determined by the type of transaction and its success or failure at different stages in its life.

A-14.1 Authorise, Capture and Settlement

The key stages in the transaction's life cycle can be grouped into the Authorisation, Capture and Settlement stages as follows:

A-14.1.1 Authorisation

An authorisation places a hold on the transaction amount in the Cardholder's issuing bank. No money changes hands yet. For example, let's say that you are going to ship a physical product from your website. First, you authorise the amount of the transaction; then you ship the product. You only capture the transaction after the product is shipped.

A-14.1.2 Capture

A capture essentially marks a transaction as ready for settlement. As soon as the product is shipped, you can capture an amount up to the amount of the authorisation. Usually, the full amount is captured. An example of a situation in which the whole amount is not captured is where the Customer ordered multiple items and one of them is unavailable.

The Gateway will normally automatically capture all authorisations as soon as they are approved, freeing up you from having to do this.

However, it is usually more desirable to delay the capture either for a period of time or indefinitely. The **`captureDelay`** field can be used for this purpose and will allow you to state the number of days to delay any automatic capture or never to automatically capture. For more details on delayed capture, refer to appendix A-10.

A-14.1.3 Settlement

Within 24 hours, the Gateway will instruct your Acquirer to settle the captured transaction. The Acquirer then transfers the funds between the Cardholder's and your accounts.

A-14.2 Transaction States

At any time during the transaction's life cycle, it is in one of a number of states as follows:

A-14.2.1 Received

The transaction has been received by the Gateway and stored away. This is the first stage. The Gateway will examine the transaction and pass it on to the next stage, as appropriate.

A-14.2.2 Approved

The transaction has been sent to the Acquirer for authorisation and the Acquirer has approved it and is holding the Cardholder's funds.

This is an intermediate state and follows the **received** state.

A-14.2.3 Verified

The transaction has been sent to the Acquirer for verification and the Acquirer has confirmed that the account is valid.

This is a terminal state and follows the **received** state. The transaction will never be settled and no funds will ever be transferred

A-14.2.4 Declined

The transaction has been sent to the Acquirer for authorisation and the Acquirer declined it. The Acquirer will not usually give any reason for a decline and will not have held any funds.

The transaction has now completed its life cycle and no more processing will be done on it.

This is a terminal state and follows the **received** state. The transaction will never be settled and no funds will ever be transferred. The transaction **responseCode** will be **5 (Declined)**.

A-14.2.5 Referred

The transaction has been sent to the Acquirer for authorisation and the Acquirer referred it for verbal approval.

You can choose not to seek verbal approval and treat these transactions the same as a normal 'declined' authorisation.

To seek verbal approval, you must phone the Acquirer and ask for an authorisation code. They will probably ask for more information about the transaction and might require you to gather other forms of identification from the Cardholder. If an authorisation code is provided, then a new transaction can be sent to the Gateway specifying the **xref** of this transaction and the received **authorisactionCode**. This new request will not be sent for authorisation and will be in the 'approved' state ready for capture and settlement.

This is a terminal state and follows the **received** state. The transaction will never be settled and no funds will ever be transferred. The transaction **responseCode** will be **2 (Referred)**.

A-14.2.6 Reversed

The transaction was sent to the Acquirer for authorisation and the Acquirer approved it. However, the transaction has been voided and the approval reversed. The Acquirer will have been asked to reverse any approval previously received, effectively cancelling the authorisation and returning any held funds back to the Cardholder.

The gateway will reverse an authorisation if it declines the transaction post authorisation due to any AVS/CV checking. The PREAUTH action will also automatically reverse an authorisation before return.

This is a terminal state and follows the **approved** state. The transaction will never be settled and no funds will ever be transferred.

If the transaction was reversed due to AVS/CV2 checking, then the transaction **responseCode** will be **5 (AVS/CV2 Declined)**.

A-14.2.7 Captured

The transaction has been captured and the Acquirer will be asked to capture the approved held funds when the settling process next runs. The settling process usually runs each evening but the Acquirer may take up to 3 days to transfer the funds.

The **capture** state can either be entered automatically if the transaction requested an immediate or delayed capture; or it can be manually requested by sending a CAPTURE request. You are free to change the amount to be captured to a value less than that initially approved by issuing one or more CAPTURE commands. When captured, there is no way to un-capture a transaction. If not explicitly cancelled, it will be sent for settlement at the next opportunity.

This is an intermediate state and follows the **approved** state.

A-14.2.8 Tendered

The transaction has been sent to the Acquirer for settlement by the settling process and is awaiting confirmation that it has been accepted.

At this point, the transaction can no longer be cancelled or re-captured.

This is an intermediate state and follows the **captured** state.

A-14.2.9 Deferred

The transaction could not be settled due to some temporary problem such as a communications loss. It will be attempted again the next time the settling process runs – usually first thing the next day.

This is an intermediate state and follows the **tendered** state. It will normally be accompanied by a transaction response that indicates why the settlement process could not settle the transaction.

A-14.2.10 Accepted

The transaction has been accepted for settlement by the Acquirer. The held funds will be transferred between the Merchant and Cardholder in due course.

The transaction has now completed its life cycle and no more processing will be done on it, unless it is subject to a rejection while the Acquirer is settling it.

This is a terminal state and follows the **tendered** state.

A-14.2.11 Rejected

The transaction has been rejected for settlement by the Acquirer. The held funds will not be transferred between the Merchant and Cardholder.

Only a few Acquirers inform the Gateway that they have rejected a transaction: they usually inform you directly. Therefore, a transaction may show as **accepted** even if was ultimately rejected or it may change from **accepted** to **rejected** if the Acquirer does inform the Gateway.

The transaction has now completed its life cycle and no more processing will be done on it.

This is a terminal state and follows the **tendered** or **accepted** states. The transaction response will normally indicate the reason the transaction was rejected.

A-14.2.12 Canceled

The transaction has been cancelled by the Merchant by sending a cancellation request to the Gateway either using the CANCEL action or via the Merchant Management System (MMS).

You can cancel any transaction that is not in a terminal state or in the ‘tendered’ state. When cancelled, any further processing that would have normally taken place will be halted. Cancelling a transaction may or may not release any funds held on the Cardholder’s card, depending on support from the Acquirer and card scheme. Note: the state is spelt American style, with a single ‘l’ as **canceled**.

This is a terminal state and follows any non-terminal state that occurs before the transaction reaches the **tendered** state.

A-14.2.13 Finished

The transaction has finished and reached the end of its lifespan but did not reached one of the other terminal states. Usually this indicates that a problem has occurred with the transaction that prevents it continuing with its normal life cycle.

This is a terminal state and can follow any other state. The transaction response will normally indicate the reason that the transaction failed.

A-15 Transaction types

The Gateway only supports card not present (CNP) types of transactions, made where the Cardholder does not or cannot physically present the card for a your visual examination at the time that an order is placed and payment effected.

The type of transaction required is specified using the type request field when performing a new payment transaction.

A-15.1 E-commerce (ECOM)

E-commerce transactions are supported by the Gateway by using a transaction **type** of **1**. They are designed for you to accept payments via a website, such as a shopping cart payment. E-commerce transactions can use advance fraud detection, such as 3-D Secure.

In accordance with Mastercard stipulations, the Gateway will not allow Maestro cards to be used for new e-commerce transactions without the use of 3-D Secure.

A-15.2 Mail Order/Telephone Order (MOTO)

Mail Order/Telephone Order transactions are supported by the Gateway by using a transaction **type** of **2**. They are designed for you to build your own virtual terminal system to enter remote order details. You will need to ensure when processing such transactions, that your Acquirer understands that the transaction is a MOTO transaction. This is because your Acquirer will have different requirements in order to classify a transaction as secure: e.g. 3-D Secure is often required for internet transactions, but impossible for MOTO transactions.

A-15.3 Continuous Authority (CA)

Continuous Authority transactions are supported by the Gateway by using a transaction **type** of **9**. They are designed for you to make subscription transactions. For further details on how to use Continuous Authority transactions, please refer to appendix A-17.2.

The Gateway offers a means of automating the taking of regular CA transactions using Recurring Transaction Agreements (RTA) as detailed in section 13.

A-16 Payment Tokenisation

All new transactions stored by the gateway are assigned a unique reference number that is referred to as the cross reference and returned in the **xref** response field. This cross reference is displayed on the Merchant Management System (MMS) and used whenever a reference to a previous transaction is required.

The cross reference can be sent as part of a transaction request, in the **xref** request field, to tell the Gateway to perform an action on an existing transaction. This is usually for management actions such as **CANCEL** or **CAPTURE**.

The cross reference can also be sent with new transactions such as **PREAUTH**, **SALE**, and **REFUND** actions, to request that the Gateway uses the values from the existing transaction if they have not been specified in the new request. For more information on how the existing values are used, please refer to appendix A-18. This allows an existing transaction to be effectively repeated without your needing to know the original card number. The only exception to this is the card's security code (CVV) which the Gateway cannot store, due to PCI DSS restrictions. Accordingly, it will have to be supplied in the new request (unless the new request is a Continuous Authority transaction, refer to appendix A-15.3).

The use of cross references to perform repeat transactions is referred to as Payment Tokenisation and should not be confused with Card Tokenisation which is a separate service offered by the Gateway.

Refer to section 13 for details on how to instruct the Gateway to repeat a payment automatically.

The way each action handles any supplied **`xref`** is as follows:

A-16.1 PREAUTH, SALE, REFUND, VERIFY requests

These requests will always create a new transaction.

The **`xref`** field can be provided to reference an existing transaction, which will be used to complete any missing fields in the current transaction. The previous transaction will not be modified. For more information on how the existing values are used, please refer to appendix A-18. If the existing transaction cannot be found, then an error will be returned and recorded against the new transaction

The request is expected to contain any transaction information required.

The **`xref`** will only be used to complete any missing card and order details, relieving you from having to store card details and reducing your PCI requirements.

A-16.2 REFUND_SALE requests

These requests will always create a new transaction.

The **`xref`** field can be provided to reference an existing transaction that is going to be refunded. This existing transaction will be marked as have been fully or partially refunded and the amounts will be tallied to ensure that you cannot refund more than the original amount of this existing transaction. If the existing transaction cannot be found, then an error will be returned and recorded against the new transaction.

The request is expected to contain any transaction information required.

The **`xref`** will not only be used to find the transaction to be refunded: additionally, that transaction will be used to complete any missing card and order details, relieving you from having to store card details and reducing your PCI requirements.

A-16.3 CANCEL or CAPTURE requests

These requests will always modify an existing transaction.

The **`xref`** field must be provided to reference an existing transaction, which will be modified to the desired state. If the existing transaction cannot be found, then an error is returned but no record of the error will be recorded against any transaction.

The request must not contain any new transaction information any attempt to send any new transaction information will result in an error. The exception is that a CAPTURE request can send in a new lesser **`amount`** field when a lesser amount, than originally authorised, must be settled.

A-16.4 QUERY requests

These requests will not create or modify any transaction.

The **`xref`** field must be provided to reference an existing transaction, which will be returned as if it had just been performed. If the existing transaction cannot be found, then an error is returned but no record of the error will be recorded against any transaction.

The request must not contain any new transaction information and any attempt to send any new transaction information will result in an error.

A-16.5 SALE or REFUND Referred Authorisation requests

These will always create a new transaction.

The **`xref`** field must be provided to reference an existing transaction, which must be of the same request type and be in the **referred** state. A new transaction will be created based upon this transaction. If the existing transaction cannot be found or is not in the **referred** state, then an error will be returned and recorded against the new transaction.

The new transaction will be put in the **approved** state and captured when specified by the existing or new transaction details. It will not be sent for authorisation again first.

The request may contain new transaction details, but any card details or order amount must be the same as the existing transaction. Any attempt to send different card details or order details will result in an error.

NB: This usage is very similar to a normal SALE or REFUND request sent with an **`authorisationCode`** included. The difference is that the **`xref`** must refer to an existing **referred** transaction whose full details are used if required and not simply an existing transaction whose card details are used if required.

This means it is not possible to create a pre-authorised SALE or REFUND request and use a **`xref`** (i.e. to use the card and order details from an existing transaction). As a soon as the **`xref`** field is seen, the Gateway identifies that it is a **referred** transaction that you wish to authorise.

A-17 Repeat Transactions

The Gateway supports two main types of repeat transactions and the option for the Gateway to take the repeat transactions automatically on behalf of the Merchant.

Repeat transactions take advantage of the Payment Tokenisation feature of the Gateway as described in appendix A-16, where each transaction is assigned a unique cross reference and allows the details from a previous transaction to be used in a later transaction.

Refer to section 13 for information on how the Gateway can be instructed to take repeat payments automatically, according to a pre-determined schedule.

A-17.1 MOTO Transactions

A Mail Order/Telephone Order (MOTO) repeat transaction, is where the Merchant makes a repeat transaction using card details that have been captured as part of a previous transaction without the Cardholder giving permission to continue to take money from their debit or credit card.

Merchants who use this system to implement billing or subscription type payments are encouraged to use the Continuous Payment Agreement method, as described in section A-17.2, to comply with Card Payment Scheme practices. *Your Acquirer may refuse to accept the repeat transactions if they are not subject to an agreement between yourself and your Customer.*

.

A-17.1.1 Initial Transaction

The initial transaction can be any transaction that has successfully stored valid credit card details and returned a **xref** response field. The transaction does not have to have resulted in a successful authorisation but would normally be a successful VERIFY, PREAUTH or SALE request.

A-17.1.2 Repeat Transaction

The repeat transaction would send the **xref** returned by the initial transaction (or previous repeat transaction) as the **xref** request field. This transaction should use a **type** of **2** (MOTO) indicating that it is a Merchant initiated transaction.

The repeat transaction would be a clone of the cross referenced transaction, including any payment details with the exception of any new data provided in the repeat transaction. The **cloneFields** request field can also be used to control which fields in the cross referenced transaction are used in the repeat transaction (refer to appendix A-18).

Because the card CVV number is never stored, repeat transactions will either require the Cardholder to re-enter their CVV or the transaction must be performed with no CVV. In such cases, the Gateway will automatically suppress CVV checking. However, some Acquirers will not allow transactions to be performed with no CVV.

A-17.2 Continuous Payment Agreements

A Continuous Payment Authority (CPA), which is sometimes referred to as a recurring payment or a 'continuous payment transaction', is where the Cardholder gives a Merchant permission to take money regularly from their debit or credit card, whenever they consider that they are owed money. Often, payday loan companies, online DVD rental subscriptions, magazine subscriptions and gym memberships use this method of payment.

A-17.2.1 Initial Transaction

The initial transaction must be any successful VERIFY, PREAUTH or SALE request. If no payment is required at the same time, then a Merchant must use a VERIFY request.

The initial transaction must be subject to the highest level of authentication supported. This therefore means that eCommerce transactions must use 3-D Secure when available.

In order to indicate that the initial transaction is the first in a Continuous Payment Authority, then the type of agreement between the Merchant and the Cardholder must be specified, using the **rtAgreementType** field.

The **rtAgreementType** can be one of the following values:

recurring – this is used when each recurring payment may be for a variable or fixed amount and the agreement shall not have a specified end date.

instalment – this is used when each recurring payment may be for a variable or fixed amount but the total of all the recurring payments will be for a fixed amount that shall be specified in the agreement with the Cardholder. Therefore, the agreement has a specified end date and the total amount to be paid is known.

A-17.2.2 Repeat Transaction

The repeat transaction would send the **xref** returned by the initial transaction (or previous repeat transaction) as the **xref** request field. This transaction must use a **type** of **9** (CA) indicating that it is a Continuous Authority transaction.

The repeat transaction would be a clone of the cross referenced transaction, including any payment details with the exception of any new data provided in the repeat transaction. The **cloneFields** request field can also be used to control which fields in the cross referenced transaction are used in the repeat transaction (refer to appendix A-18).

Because the card CVV number is never stored, repeat transactions will not require a card CVV to be supplied.

Acquirers may insist that a separate acquiring account must be used for any Continuous Authority payment, in which case this would be associated with a different Merchant Account. In such cases, the initial transaction would be performed against your normal Merchant Account and the repeat transactions would be performed against your Continuous Authority Merchant Account.

It is the responsibility of the Merchant to regulate the transaction values and frequencies. Please be aware that as a rule of thumb, the banks expect Continuous Authority payments to be a predictable transaction amount on a regular or predictable frequency. Any deviation from this can be viewed as an abuse of the Merchant's Continuous Authority acquiring account. You must also

only ever process a Continuous Authority transaction on a card for which you have obtained full authorisation and authentication via your normal Merchant Account.

Mastercard stipulates that the Gateway will not allow Maestro cards to be used with Continuous Authority transactions.

blink

A-18 Transaction Cloning

If a new transaction request is received with the Cross Reference (**xref**) of an existing transaction, then the values of certain fields in the existing transaction will be used to initialise the new transaction where new values have not been provided in the new request. This copying of fields from a base transaction is termed 'transaction cloning', and the copied-over value is termed the 'cloned value'.

Appendix A-18.1 shows all the fields whose values can be copied over from the existing transaction. To allow for easy addition of future fields, the fields are grouped into logical groupings and each group is given a name (as show in brackets after the group title).

Certain groups of fields, such as address fields, can only be copied as a whole entity and any new value provided in the new request will prevent the whole group from being copied from the existing transaction. Please note that line item data (items) cannot be merged.

By default, the values of all the fields listed in appendix A-18.1 are copied from the existing transaction where appropriate. However, you can control exactly which fields are copied using the **cloneFields** field in the new request. The value of **cloneFields** should be a comma separated list of field names or group names that should be copied over. Alternatively, if you wish to specify a list of fields not to copy, then prefix the list with a single exclamation mark (!).

Field Name	Mandatory?	Description
`cloneFields`	N	Comma separated list of field names or group names whose values should be cloned.

Examples

To copy over only the value of **customerName** and any values for the fields in the ***customerAddressFields*** group:

cloneFields="customerName, customerAddressFields"

To copy over the values of all supported fields apart from the value of **customerName** and **merchantName**:

cloneFields="!customerName,merchantName"

A-18.1 Cloned Fields

Transaction fields currently cloned are as follows:

Order Details Fields (**orderFields**)
- **type**
- **countryCode**
- **currencyCode**
- **amount**
- **grossAmount**
- **netAmount**
- **taxRate**
- **taxAmount**
- **taxReason**
- **discountAmount**
- **discountReason**
- **handlingAmount**
- **insuranceAmount**

Order Reference Fields (**orderRefFields**)
- **transactionUnique**
- **orderRef**
- **orderDate**

Card Fields (**cardFields**)
- **paymentMethod**
- **cardToken**
- **cardNumber**
- **cardExpiryDate**
- **cardExpiryMonth**
- **cardExpiryYear**
- **cardStartDate**
- **cardStartMonth**
- **cardStartYear**
- **cardIssueNumber**

Cardholder Fields (**cardholderFields**)
- **customerName**
- **customerAddress**
- **customerPostcode**
- **customerEmail**
- **customerPhone**

Purchase Fields (**purchaseFields**)
- **Items**

Statement Narrative Fields (**narrativeFields**)
- **statementNarrative1**
- **statementNarrative2**

3D Secure Fields (**threedsFields**)[1]
- **threeDSRequired**
- **threeDSCheckRef**

AVS/CV2 Fields (**avscv2Fields**)
- **avscv2Required**
- **cv2CheckPref**
- **addressCheckPref**
- **postcodeCheckPref**
- **customerAddress**
- **customerPostcode**

Merchant Email Notification Fields (**notifyFields**)
- **notifyEmail**

Customer Receipt Fields (**cReceiptFields**)
- **customerReceiptRequired**
- **customerEmail**

Merchant Information Fields (**merchantFields**)
- **merchantName**
- **merchantCompany**
- **merchantAddress***
- **merchantTown***
- **merchantCounty***
- **merchantPostcode***
- **merchantCountryCode***
- **merchantPhone**
- **merchantMobile**
- **merchantFax**
- **merchantEmail**
- **merchantWebsite**
- **merchantData**
- **merchantOrderRef**
- **merchantCustomerRef**
- **merchantTaxRef**
- **merchantOriginalOrderRef**
- **merchantCategoryCode**
- **merchantType**

Customer Information Fields (**customerFields**)
- **customerName**
- **customerCompany**
- **customerAddress***
- **customerTown***
- **customerCounty***
- **customerPostcode***
- **customerCountryCode***
- **customerPhone**

[1] 3D Secure fields are only cloned if both the existing and new transaction support 3-D Secure.

- **customerMobile**
- **customerFax**
- **customerEmail**
- **customerOrderRef**
- **customerMerchantRef**
- **customerTaxRef**

Supplier Information Fields (**supplierFields**)

- **supplierName**
- **supplierCompany**
- **supplierAddress***
- **supplierTown***
- **supplierCounty***
- **supplierPostcode***
- **supplierCountryCode***
- **supplierPhone**
- **supplierMobile**
- **supplierFax**
- **supplierEmail**

Receiver Information Fields (**receiverFields**)

- **receiverName**
- **receiverCompany**
- **receiverAddress***
- **receiverTown***
- **receiverCounty***
- **receiverPostcode***
- **receiverCountryCode***
- **receiverPhone**
- **receiverMobile**
- **receiverFax**
- **receiverEmail**
- **receiverAccountNo**
- **receiverDateOfBirth**

Delivery Information Fields (**deliveryFields**)

- **deliveryName**
- **deliveryCompany**
- **deliveryAddress***
- **deliveryTown***
- **deliveryCounty***
- **deliveryPostcode***
- **deliveryCountryCode***
- **deliveryPhone**
- **deliveryMobile**
- **deliveryFax**
- **deliveryEmail**

Shipping Information Fields (**shippingFields**)

- **shippingMethod**
- **shippingTrackingRef**

- **shippingAmount**
- **shippingGrossAmount**
- **shippingNetAmount**
- **shippingTaxRate**
- **shippingTaxAmount**
- **shippingTaxReason**
- **shippingDiscountAmount**
- **shippingDiscountReason**

MCC 6012 Additional Authorisation Data (**mcc6012Fields**)

- **receiverName**
- **receiverPostcode**
- **receiverAccountNo**
- **receiverDateOfBirth**

Payment Facilitator Data (**facilitatorFields**)[1]

- **subMerchantID**
- **facilitatorID**
- **facilitatorName**

[1] Payment facilitator fields are only cloned if the existing transaction uses the same **`merchantID`** as the new transaction.

A-18.2 Cloned Groups

To allow for easy future addition of new fields, the existing fields are grouped into logic groupings. Each group is given a name (as shown in brackets after the group title). It is recommended that this group name be used in any **cloneFields** value instead of listing all the fields separately.

A-18.2.1 Compound Groups

To help maintain transaction integrity, certain groups of fields, such as address fields, can only be copied as a whole entity and any new value provided in the new request will prevent the whole group from being copied from the existing transaction.

These compound fields are marked with an asterisk in appendix A-18.1 and can be referred to in **cloneFields** as logical groups using the following group names; ***merchantAddressFields***, ***customerAddressFields***, ***deliveryAddressFields***, ***supplierAddressFields*** and ***receiverAddressFields***.

A-18.2.2 Line Item Data

Any line item data (**items**) is copied over in its entirety and there is no way to merge the line item from an existing transaction with any sent in a new transaction.

A-18.2.3 Amount Consistency

The Gateway does not validate that the various sub-amount fields, such as **netAmount**, **grossAmount,** all add up to the actual requested **amount**. Therefore, these fields are currently not treated as a compound group.

If a new **`amount`** value is passed that is different from the value in the existing transaction, then the following fields should also be passed so that they tally with the new amount.

`grossAmount`
`netAmount`
`taxRate`
`discountAmount`

blink

A-19 Stored Credentials Framework

To make sure merchants use their customers' details responsibly, Visa and Mastercard have introduced a new framework for the storing of card details and new rules for any associated transactions. This framework identifies stored credentials as Credentials on File (COF) and classifies the transaction that use them as either Consumer Initiated Transactions (CIT) or Merchant Initiated Transactions (MIT).

If you process transactions using stored credentials, you may need to make changes to comply with these rules.

Currently the only credentials stored are card details and so the terms Consumer, Customer and Cardholder can be used interchangeably.

Field Name	Mandatory?	Description
`initiator`	N	Indicate who initiated the transaction. Possible values are: consumer – consumer initiated (CIT) merchant – merchant initiated (MIT)
`rtAgreementType`	No	Consumer/Merchant agreement type. Possible values are: cardonfile – credential storage agreed (CIT/MIT). recurring – recurring type CPA agreed (CIT/MIT). instalment – instalment type CPA agreed (CIT/MIT). unscheduled – adhoc COF payment (MIT) incremental[1] – authorisation amount increment (MIT) resubmission – failed authorisation retry (MIT) reauthorisation – expired authorisation refresh (MIT). delayedcharges – post authorisation charges (MIT). noshow – missed reservation penalty (MIT)

For backwards compatibility, the Gateway will try to automatically identify if a transaction is a Consumer Initiated Transaction or a Merchant Initiated Transaction from the value provided for the **`action`**, **`type`** and **`rtAgreementType`** fields.

You may also pass the **`initiator`** field in the request to force a classification. This can be used if the Gateway is unable to correctly determine classify the transaction. If, however, the requested classification is incompatible with the provided request fields then the transaction will fail with a **`responseCode`** of **66944** (INVALID INITIATOR).

The **`initiator`** field will be returned in the response with either the value passed in the request or the automatically identified value.

[1] MIT type **incremental** is not currently supported but reserved for future use.

A-19.1 Credentials on File (CoF)

Credentials on File (CoF) is the process when the Consumer authorises you to store their credentials (including, but not limited to, an account number or payment token) for future transactions. This includes for future Recurring or Instalment payments and Unscheduled ad-hoc payments, where the Consumer does not need to enter their payment credentials again.

These transactions must always be identified with the reason for storing or using the stored credentials and who initiated the transaction - Consumer (CIT) or Merchant (MIT).

You may store the credentials and send them with the future transaction, or you may store the details in the Gateway's Wallet as described in section 18 or by taking advantage of the Payment Tokenisation feature of the Gateway as described in appendix A-16. Either way you must tell the Gateway of your intentions, we will not assume that just because you have asked, for example, to store credentials in the Wallet that those are legitimate stored credentials and follow all the requirements laid out below.

If you store credentials on file, then you must:

- Disclose to consumers how those credentials will be used.
- Obtain consumers' consent to store the credentials.
- Notify consumers when any changes are made to the terms of use.
- Inform the card issuer via a transaction that payment credentials are now stored on file.
- Identify transactions with appropriate **`rtAgreementType`** when using stored credentials.
- Perform a PREAUTH, SALE or VERIFY transaction during the initial credential setup.

Note: Credentials stored to complete a single transaction (or a single purchase) for a Consumer, including multiple authorisations related to that particular transaction or future refunds are not considered stored credentials and can be stored and used without the following the above rules.

A-19.2 Consumer Initiated Transactions (CIT)

Consumer Initiated Transactions (CIT) are any transaction where the Consumer is actively participating in the transaction. This can be either through a checkout experience online, via a mail order or telephone order, with or without the use of an existing stored credential.

A Consumer Initiated Transaction is one whose **`action`** field is one of **PREAUTH**, **SALE** or **VERIFY** and whose **`type`** is one of **1** (ECOM) or **2** (MOTO).

To indicate that the card details are to be stored as, or were stored as, Credentials on File then send the **`rtAgreementType`** field as one of the following values:

cardonfile – card details stored as Credential on File
recurring – initial payment as the start of a recurring payment agreement.
instalment – initial payment as the start of an instalment payment agreement.

If the card details are cloned from an existing transaction or loaded from a Gateway Wallet which also stored the Credentials on File then the transaction will be flagged as subsequent use of stored credentials rather than first use of them[1].

Refer to section 13 for more information on **recurring** or **instalment** payment agreements.

[1] For flagging of subsequent use the existing credentials will usually need to have been stored with the same Acquirer.

A-19.3 Merchant Initiated Transactions (MIT)

Merchant Initiated Transactions (MIT) are any transaction where you have performed the transaction without the active participation of the Consumer. This would always be as a follow-up to a previous Consumer Initiated Transaction (CIT).

Merchant Initiated Transactions are broken down in to two categories as follows.

A-19.3.4 Standing Instruction MITs

Merchant Initiated Transactions defined under this category are performed to address pre-agreed standing instructions from the Consumer for the provision of goods or services.

The following transaction types are standing instructions transactions:

Instalment Payments: A transaction in a series of transactions that use a stored credential and that represent Consumer agreement for the merchant to initiate one or more future transactions over a period for a single purchase of goods or services.

Recurring Payments: A transaction in a series of transactions that use a stored credential and that are processed at fixed, regular intervals (not to exceed one year between transactions), representing Consumer agreement for the merchant to initiate future transactions for the purchase of goods or services provided at regular intervals.

Unscheduled Credential on File (UCOF): A transaction using a stored credential for a fixed or variable amount that does not occur on a scheduled or regularly occurring transaction date, where the Consumer has provided consent for the merchant to initiate one or more future transactions. An example of such transaction is an account auto-top up transaction.

A-19.3.5 Industry-Specific Business Practice MIT

Merchant Initiated Transactions defined under this category are performed to fulfil a business practice as a follow-up to an original Consumer-Merchant interaction that could not be completed with one single transaction. Not every industry practice Merchant Initiated Transaction requires a stored credential, for example, if you store card details for a single transaction or a single purchase, it is not considered as a stored credential transaction.

The following transaction types are industry specific transactions[1]:

Incremental[2]: Incremental authorizations can be used to increase the total amount authorised if the authorised amount is insufficient. An incremental authorization request may also be based on a revised estimate of what the Consumer may spend.

Resubmission: You can perform a resubmission in cases where it requested an authorization but received a decline due to insufficient funds; however, the goods or services were already delivered to the Consumer. In such scenarios, you can resubmit the request to recover outstanding debt from Consumers.

Reauthorization: You can initiate a reauthorization when the completion or fulfilment of the original order or service extends beyond the authorization validity limit set by the card scheme.

Delayed Charges: Delayed charges are performed to make a supplemental account charge after original services have been rendered and payment has been processed.

No Show: Consumers can use their payment credentials to make a guaranteed reservation with certain merchant segments. A guaranteed reservation ensures that the reservation will be honoured and allows you to perform a No Show transaction to charge the Consumer a penalty according to your cancellation policy. If no payment is made to guarantee a reservation, then it is necessary to perform a VERIFY Consumer Initiated Transaction at the time of reservation to be able perform a No Show transaction later.

[1] Not all Acquirers support all transaction types.
[2] The Gateway does not currently support incremental authorisations.

A Merchant Initiated Transaction is one whose **`action`** field is one of **PREAUTH**, **SALE** or **VERIFY** and whose **`type`** is one of **2** (MOTO) or **9** (CA) depending on the category.

To indicate the type of MIT, send the **`rtAgreementType`** field as one of the following values:

recurring – subsequent payment as the start of a recurring payment agreement (CA).
instalment – subsequent payment as the start of an instalment payment agreement (CA).
unscheduled – subsequent payment not to a fixed schedule (MOTO)
incremental – subsequent payment to increment initial amount authorised (MOTO)
resubmission – subsequent payment due to failed initial payment (MOTO)
reauthorisation – subsequent payment to refresh expired initial payment (MOTO)
delayedcharges – subsequent payment for additional charges (MOTO)
noshow – subsequent payment as penalty for missed reservation (MOTO)

The **`xref`** of the initial Consumer Initiated Transaction must be provided as follows:

For standing order MITs the initial authorisation must have been a successful Consumer Initiated Transaction with Credentials on File. This MIT will be a subsequent use of those Credentials on File. For **recurring** and **instalment** MITs the initial authorisation must have used the same **`rtAgreementType`**. The xref can be to the previous MIT in which case the Gateway will follow the chain of transactions back to the initial CIT.

For industry practice MITs the initial authorisation must be successful (apart from for a **resubmission**) but need not have Credentials on File. For example, it may not be known at the time of the initial authorisation that the MIT would be required and so the initial authorisation would not necessarily have stored the Credentials on File. This is an example of when an industry practice Merchant Initiated Transaction does not require a stored credential

Note: For compatibility with existing practices, Instalment Payments and Recurring Payments MITs use Continuous Authority (CA) **`type`** transactions while other MITs Mail Order/Telephone Order (MOTO) **`type`** transactions. This use of MOTO is different to its use with a Consumer Initiated Transaction (CIT).

Refer to section 13 for more information on **recurring** or **instalment** Continuous Authority payment agreements.

A-20 Integration Libraries

We can provide a range of libraries to help you to integrate with the Gateway.

These libraries include simple sever-side classes in many popular programming languages through to client-side scripts to help with the integration of the Hosted Payment Page or Hosted Payment Fields.

The server-side libraries can be obtained by contacting customer support.

The client-side libraries can be downloaded directly from the Gateway server.

A-20.1 Gateway Integration Library

A simple server-side integration library is available to simplify the preparation and transmission of Hosted and Direct Integration requests.

The library is available in many popular programming languages and is based around a single class: the **`Gateway`** class.

The Gateway integration library does not currently support the preparation and transmission of Batch Integration requests.

A-20.1.1 Library Namespace

To avoid polluting the global namespace, the library uses the 'P3/SDK' namespace where supported by the language.

A-20.1.2 Gateway Configuration

Before you can use the **`Gateway`** class, you will need to configure the following properties to match your integration parameters and authentication parameters documented in section 1.6.

Property Name	Type	Description
`hostedURL`	string	Absolute URL provided for the Hosted Integration. [Default: Gateway's Hosted Integration URL]
`directURL`	string	Absolute URL provided for the Direct Integration. [Default: Gateway's Direct Integration URL]
`merchantID`	string	Your unique Merchant ID to be passed in the `merchantID` integration field. [Default: 100001]
`merchantPwd`	string	Any password configured on your Merchant Account as per section 1.6.1. [Default: null]
`merchantSecret`	string	Any secret configured on your Merchant Account as per section 1.6.2. [Default: Circle4Take40Idea]
`proxyUrl`	string	Absolute URL to any proxy required for connections. (eg https://www.proxy.com:3128) [Default: null]
`debug`	boolean	True to enable debugging output. [Default: false]

A-20.1.3 Gateway Methods

The follow methods are made available by the **`Gateway`** class:

`string hostedRequest(mixed[] request, string[] options)`

Return an HTML fragment that can be included in your webpage to render a <form> which will send the provided request data to the Gateway's Hosted Integration when submitted.

The **`request`** parameter should be an associative array containing the request fields required to be sent. The request fields are not validated.

The following class properties are used unless alternative values are provided in the **`request`** array: **`directUrl`**, **`merchantID`**, **`merchantPwd`**, **`merchantSecret`**.

The **`options`** parameter is an optional associative array containing options that can be used to modify the returned HTML fragment as follows:

`formAttrs`	– string containing additional attributes to include in the form tag.
`submitAttrs`	– string containing additional attributes to include in the submit button tag.
`submitImage`	– string containing the URL to use as the submit button.
`submitHtml`	– string containing HTML to use as the label on the submit <button>.
`submitText`	– string containing text to use as the label on the submit <input>.

The **`submitImage`**, **`submitHtml`** and **`submitText`** options are mutually exclusive and will be checked for in that order. If none is provided, then a **`submitText`** value of 'Pay Now' is assumed.

If a **`merchantSecret`** is provided, then the method will add the correct **`signature`** field to the request.

An exception is thrown if the HTML fragment cannot be composed.

The **`verifyResponse()`** method can be used to validate and decode any response POSTed back to your website.

Please refer to appendix A-23.1.1 for an example of how to use this method.

Returns a string containing the HTML fragment if successful; throws an exception otherwise.

`mixed[] directRequest(mixed[] request, string[] options)`

Return the response received when sending the provided request to the Gateway's Direct Integration.

The **`request`** parameter should be an associative array containing the request fields required to be sent. The request fields are not validated.

The following class properties are used unless alternative values are provided in the **`request`** array: **`directUrl`**, **`merchantID`**, **`merchantPwd`**, **`merchantSecret`**.

The **`options`** parameter is not used and reserved for future use.

If a **`merchantSecret`** is provided, then the method will add the correct **`signature`** field to the request and check the **`signature`** field on the response.

An exception is thrown if the request cannot be sent; or the response cannot be received; or if the response's **`signature`** is incorrect.

Please refer to appendix A-23.1.2 for an example of how to use this method.

Returns an associative array containing the received response fields; otherwise, throws an exception.

`void prepareRequest(mixed[] &request, string[] &options,`
`string &secret, string &direct_url, string &hosted_url)`

Prepare a request for sending to the Gateway's Direct Integration.

The **`request`** parameter should be a reference to an associative array containing the request fields required to be sent. The request fields are not validated.

The **`merchantSecret`**, **`directUrl and hostedUrl`** configuration properties will be returned in the **`secret`**, **`direct_url`** and **`hosted_url`** method parameters. These properties can be overridden by providing them in the **`request,`** in which case they will be extracted and removed from the request.

The **`merchantID`** and **`merchantPwd`** configuration properties will be added to the **`request`**.

A few known Gateway response fields will be removed from the request, if present, to avoid confusion, notably the **`responseCode`**, **`responseMessage`**, **`responseStatus`**, **`state`** fields.

An exception will be thrown if the request does not contain an action element or a merchantID element (and none could be inserted).

`void verifyResponse(mixed[] &response, string secret)`

Verify a response received from the Gateway's Hosted or Direct Integration.

The **`response`** parameter should be a reference to an associative array containing the response received from the Gateway, either from the Direct Integration or as POSTed from the Hosted Integration.

The **`secret`** parameter should be any Merchant secret to use when checking the response's **`signature`** element. If not provided, then the value of the **`merchantSecret`** property is used.

Any **`signature`** element is removed from the **`response`**.

An exception is thrown if the response is not valid, does not contain a **`responseCode`** element or its **`signature`** is incorrect.

Please refer to appendix A-23.1.1 for an example of how to use this method.

`string sign(mixed[] request, string secret, mixed partial = false)`

Return the signature for the provided request data.

The **`request`** parameter should be a reference to an associative array containing the request fields required to be sent. The request fields are not validated.

The **`secret`** parameter should be the Merchant secret to use when signing the request.

The **`partial`** parameter should be either the boolean **false** or comma separated string; or an array of strings containing the names of the request elements to sign.

Returns a string containing the correct signature for the request.

A-20.2 Hosted Payment Page Library

A simple client-side script is available to simplify the displaying of the Hosted Payment Page in a lightbox overlaying your website.

The library is available as a JavaScript script and is based around a single class: the **`Form`** class. The script is compatible with most modern web browsers.

The script can be loaded directly from our Gateway server as follows[1]:

```
1.  <script src="https://gateway.example.com/sdk/web/v1/js/hostedforms.min.js"></script>
```

If the script detects the presence of the jQuery API, then it will extend the jQuery object with its own plugin method. However, jQuery is not needed in order to use the script.

A-20.2.1 Hosted Payment Pages

Hosted Payment Pages are a prebuilt webpage residing on our server that you can use to collect sensitive payment details without those details' touching your server. The standard Hosted Payment Page is designed so that it can be displayed in a transparent overlay over your website, thus making the Customer feel as though they never left your shopping cart.

The standard Hosted Integration examples redirect the Customer's browser to the Hosted Payment Page, resulting it appearing on a new browser page and not overlaying your website. The Hosted Payment Page library provides the scripting necessary to result in the redirection, causing the Hosted Payment Page to appear in an overlay and not a new browser page, without your having to make any modifications to your website. The library can also simplify the creation of the Hosted Integration redirection FORM if required.

Note: Use /hosted for Hosted Form v2 and /payment form for Hosted Form v2.

A-20.2.2 Library Namespace

To avoid polluting the global namespace, the library extends the global `window` object with a **`hostedForms`** object containing the following properties:

- **`forms`** – array containing all the instantiated **`Form`** objects.
- **`classes`** – array containing all the instantiable classes.
  - **`form`** – **`Form`** class prototype.

[1] Please use the correct hostname as provided in section 1.6.

A-20.2.3 Form Construction

The construction method can be used to build and prepare a HTML FORM element for use with the modal Hosted Payment Page; or to prepare an existing element. The method signature is as follows:

```
Form(element, data)
```

The **element** parameter should be either the id or DOM node of an existing FORM or DIV DOM element.

If the **element** is a DIV node, then the data is used to create a new FORM node within the **element**.

If the **element** is a FORM node, then the data is used to modify the existing FORM **element**.

The **data** parameter should be an object containing construction details and can contain the following optional properties:

- **id** – string containing the value to use as the FORM tag's id attribute.
- **url** – string containing the URL to use as the FORM tag's src attribute.
- **attrs** – object whose properties are added as additional attributes on the FORM tag.
- **modal** – boolean indicating that the HPP should open in a modal overlay.
- **data** – object whose properties are added as hidden input elements in the FORM.
- **submit** – object containing details for a submit button that should be added to the FORM.
  - **type** – type of submit button, either 'auto', 'image', 'button', 'input'
  - **id** – string containing the value to use as the submit button's id attribute.
  - **attrs** – object whose properties are added as additional attributes on the submit button.
  - **label** – string containing button label (or 'alt' attribute for 'image' buttons)
  - **src** – string containing image URL for 'image' buttons.

The constructor will submit the FORM immediately after preparation if the **data.submit.type** property is 'auto'; or if the existing FORM **element** has a `data-hostedform-autosubmit` attribute. Otherwise, an event handler will be attached to the submit button to disable it automatically when clicked, to help prevent your Customer from clicking it twice.

The constructor will prepare the FORM so that the Hosted Payment Page (HPP) will be opened in a modal overlay if the **data.modal** property is true; or if the existing FORM **element** has a `data-hostedform-modal` attribute; or has an `action` attribute containing the string 'modal/' or ending in the string 'modal'.

The modal overlay is automatically created as a semi-opaque IFRAME element that fills the browser display. The Hosted Payment Page is then loaded into this IFRAME and, being semi-opaque, your shopping cart will remain visible beneath, but greyed out and noninteractive. When the Customer closes the Hosted Payment Page, then their browser will be redirected to the URL you provided using the **redirectURL** parameter. This will cause the original page and IFRAME to be replaced by your new page.

A-20.2.4 Form Methods

The follow methods are made available by the `Form` class:

`void destroy()`

Destroys the `Form`, reverting its `element` back to its original state.

A-20.2.5 jQuery Plugin

If the jQuery API has been loaded into the browser before the script, then it will extend the jQuery object with its own plugin method.

Construction and destruction can then be done as follows:

```
$(element).hostedForm(data);
$(element).hostedForm('destroy');
```

A-20.3 Hosted Fields Library

A simple client-side script is available to support the displaying of Hosted Payment Fields in your payment form.

The library is available as a JavaScript script and is based around two classes: the **`Form`** and **`Field`** classes. The script is compatible with most modern web browsers.

The script can be loaded directly from our Gateway server as follows[1]:

```
1.  <script src="https://gateway.example.com/sdk/web/v1/jsFfields.min.js"></script>
```

The script requires the jQuery API, which must be loaded prior to the script.

A-20.3.1 Hosted Fields

Hosted Payment Fields are a set of prebuilt JavaScript UI components that can be used by your website's HTML payment form to collect sensitive payment details without those details touching your server. They provide you with the PCI benefits of using a Hosted Payment Page, while allowing you the ability to design and implement your own payment forms.

There are 6 predefined Hosted Payment Fields available as follows:

cardNumber	– collects the card number.
cardCVV	– collects the card cvv.
cardExpiryDate	– collects the card expiry month and year.
cardStartDate	– collects the card start/issue month and year.
cardIssueNumber	– collects the card issue number.
cardDetails	– collects the card number, expiry date and cvv in a single field.

The **cardNumber** field is designed to collect a card number, including an icon used to display the card type. The field will only accept digits and spaces and validate that any entered value is a correctly formatted card number and insert spaces at the correct positions for the card type as the number is typed.

The **cardCVV** field is designed to collect a card CVV. The field will only accept digits and will validate that any entered value is a correctly formatted CVV, taking into account the card type as determined by an associated **cardNumber** field.

The **cardExpiryDate** and **cardStartDate** fields are designed to collect a card expiry date and card issue date respectively. The fields can render as a pair of select controls containing the months and a suitable range of years; or as an input control that will only allow digits to be entered and automatically formatted as a month / year entry. The field will validate that any entered value is a valid month and year combination.

[1] Please use the correct hostname as provided in section 1.6.

The **cardIssueNumber** field is designed to collect a card issue number. The field will only accept digits and will validate that any entered value is a correctly formatted issue number.

The **cardDetails** field is designed to collect all of the essential card details. It combines the **cardNumber**, **cardExpiryDate** and **cardCVV** fields into a single line compound field design to allow easy entry of the card details and to complement the look of your checkout.

The field type is either: passed as the value of the **`type`** option the **`Field`** construction, provided by the HTML element's meta data; or provided via the HTML element's type attribute (prefixed with the 'hostedfield:' name space).

The following example shows all three approaches to specifying the field type:

```
1.  <input type="hostedfield:cardNumber" name="card-number">
2.  <div class="hostedfield" data-hostedfield-type="cardExpiryDate"></div>
3.  <input data-hostedfield='{"type":"cardCVV"}'>
```

It is highly recommended that you adopt a single approach as above and don't mix and match.

Each field type has its own additional configuration options, as detailed in section A-20.3.6.

A-20.3.2 Library Namespace

To avoid polluting the global namespace, the library extends the global `window` object with a **`hostedFields`** object containing the following properties:

- **`forms`** – array containing all the instantiated **`Form`** objects.
- **`classes`** – array containing all the instantiable classes.
  - **`form`** – **`Form`** class prototype.

A-20.3.3 Form Construction

The construction method can be used to prepare a HTML FORM for use with Hosted Payment Field components. The method signature is as follows:

```
Form(element, options)
```

The **element** parameter should be the DOM node of an existing FORM tag.

The **options** parameter should be object containing one of more of the following optional properties:

autoSetup	– boolean indicating whether setup should be handled automatically.
autoSubmit	– boolean indicating whether submission should be handled automatically.
merchantID	– string containing the **merchantID** the payment request is for.
stylesheet	– string containing DOM selector for any stylesheets to be used.
tokenise	– string/array/object specifying fields whose values should be tokenised.
fields	– object containing field configuration by field type.
locale	– string containing the desired locale.
classes	– object containing names of extra CSS classes to use.
submitOnEnter	– boolean indicating whether the enter key should cause the form to submit.
nativeEvents	– boolean indicating that native browser events should be fired.

Any **options** parameter will be merged with those provided via meta data supplied, using `data-hostedfield` and/or `data-hostedfield-<option>` attributes; or via existing attributes or properties of the **element**.

The **autoSetup** option can be used to disable the automatic creation of **Field** objects for the FORM child controls by calling the **autoSetup()** method during the **Form** construction. If automatic setup is disabled, then you must manually instantiate **Field** objects and attach them to the **Form** as required, using the **addField()** method. This option or manually calling the **autoSetup()** method minimises the amount of JavaScript you have to write. Automatic operation is good if you don't need to customise the operation or can't customise it by reacting to the **Form** or **Field** events. The option defaults to true and cannot be changed once the **Form** has been created.

The **autoSubmit** option can be used to disable the automatic handling of the FORM submission via the **autoSubmit()** method. If automatic submission is disabled, then you must manually retrieve the sensitive payment details by calling **getPaymentDetails()** and include them in the form submission data. This option or manually calling the **autoSubmit()** method minimises the amount of JavaScript you have to write. Automatic operation is good if you don't need to customise the operation or can't customise it by reacting to the **Form** or **Field** events. The option defaults to true and cannot be changed once the **Form** has been created.

The **merchantID** option can be used to specify the **merchantID** with which the final **paymentToken** will be used. The option defaults to the value of any child INPUT node whose name is 'merchantID' and can be changed at runtime by calling the **setMerchantID()** method or by altering the options using the jQuery **hostedForm()** plugin method.

The **stylesheet** option can be used to specify a DOM selector used to locate stylesheets that should be parsed for styles related to the Hosted Payment Fields. Refer to section A-20.3.10 for

how to style the Hosted Payment Fields using CSS stylesheets. The option defaults to the DOM selector string 'link.hostedfield[rel=stylesheet], style.hostedfield' and can be changed at runtime by calling the **`setStylesheet()`** method; or by altering the options using the jQuery **`hostedForm()`** plugin method.

The **`tokenise`** option can be used to specify addition FORM controls whose values, as returned by the `jQuery.val()` method, should be included in the final **`paymentToken`**.
The option's value must be either:

- A string containing a DOM selector used to select one or more controls.
- An array containing values used to `jQuery.filter()` down to one or more controls.
- An object whose properties are the name of fields to tokenise and whose values are objects containing a **`selector`** property used to select a control.

For the first two, the tokenised field's name be will be taken from the controls `data-hostedfield-tokenise` attribute or `name` attribute. For the third, the name is property name in the **`tokenise`** object. If the field's name is of the format 'paymentToken[<name>]', then only the '<name>' part is used. The option defaults to the DOM selector string 'INPUT.hostedfield-tokenise:not(:disabled), INPUT[data-hostedfield-tokenise]:not(:disabled), INPUT[name^="paymentToken["]:not(:disabled)' and cannot be changed once the **`Form`** has been created.

The **`fields`** options can be used to specify default options for the different types of Hosted Payment Fields. The option's value should be an object whose properties are the fields type or the wildcard type 'any' and whose values are objects whose properties are the default options for fields of that type. The values can also contain a **`selector`** property containing a DOM selector that is used during the automatic setup stage to select a FORM's child element to add as a **`Field`** of the specified type automatically. The option has no default value and cannot be changed once the **`Form`** has been created.

The **`locale`** option can be used to specify the language that should be used by the Hosted Payment Fields attached to this **`Form`**. The option defaults to the value provided by any `lang` attribute on the **element** or closest ancestor and cannot be changed once the **`Form`** has been created.

The **`classes`** options can be used to specify additional CSS class names to add in addition to the default classes documented in section A-20.3.9. The value is an object whose properties are the default class name and whose values are a string containing the additional class name(s) to use. The option has no default and cannot be changed once the **`Form`** has been created.

The **`submitOnEnter`** option can be used to specify if pressing the enter key when typing a **`Field`** value should cause the **`Form`** to submit. The option defaults to false and cannot be changed once the **`Form`** has been created.

The **`nativeEvents`** option can be used to specify that any associated native event should be fired when a 'hostedField:' prefixed **`Field`** event is fired (as documented in section A-20.3.8). For example, when enabled if the 'hostedfield:mouseover' event is fired, then the native 'mouseover' event is also fired. The option defaults to false and cannot be changed once the **`Form`** has been created.

If not explicitly constructed, a **`Form`** object will be automatically instantiated and attached to the FORM DOM node as soon as any **`Field`** object is instantiated on a child DOM node.

Example **`Form`** construction is as follows:

```
var form = new window.hostedFields.classes.Form(document.forms[0],{
    // Auto setup the form creating all hosted fields (default)
    autoSetup: true,

    // Auto validate, tokenise and submit the form (default)
    autoSubmit: true,

    // Additional fields to tokenise
    tokenise: '.add-to-token',

    // Stylesheet selection
    stylesheets: '#hostedfield-stylesheet',

    // Optional field configuration (by type)
    fields: {
        any: {
            nativeEvents: true
        },
        cardNumber: {
            selector: $('#form2-card-number'),
            stylesheet: $('style.hostedform, style.hostedform-card-number')
        }
    },

    // Additional CSS classes
    classes: {
        invalid: 'error'
    }
});
```

Or using meta data on the HTML FORM element:

```
<form data-hostedfields='{"autoSetup":true,"autoSubmit":true,"tokenise":".add-to-
token","stylesheets":"#hostedfield-
stylesheet","fields":{"any":{"nativeEvents":true},"cardNumber":{"selector":"#form2-card-
number","stylesheet":"style.hostedform, style.hostedform-card-
number"}},"classes":{"invalid":"error"}}' method="post" novalidate="novalidate" lang="en">
<script>
var form = new window.hostedFields.classes.Form{document.forms[0]);
</script>
```

A-20.3.4 Form Methods

The follow methods are made available by the **`Form`** class:

`void autoSetup()`

Automatically setup the form by scanning the Form element for child nodes to control as Hosted Payment Fields. Child nodes are selected if they:

- have a `type` attribute with a `hostedfield:<type>` value *(INPUT nodes only)*.
- have a `data` attribute with a `hostedfield.<type>` property.
- match a DOM selector provided by the `fields.<type>.selector` option.

If multiple selection criteria are present, then they must all specify the same **`Field`** type or an exception is thrown.

This method is called during the **`Form`** construction unless the **`autoSetup`** option is false.

`void autoSubmit()`

Automatically handles any attempted FORM submission by checking the FORM's controls are valid by calling the **`validate()`** method; and then requesting the **`paymentToken`** using the **`getPaymentDetails()`** method; and finally adding the token to the forms fields using the **`addPaymentToken()`** method. Failure to validate or request the payment token will cause the form submission to be stopped.

You can affect the automatic submission stages by listening for events and preventing their default actions. The full list of events is documented in section A-20.3.5.

This method is attached to the FORM submit event during the **`Form`** construction unless the **`autoSubmit`** option is false, or the **`autoSubmit`** option is null and the **`autoSetup`** option is false.

If automatic submission is disabled, then you must react to the FORM's submit event and then request the **`paymentToken`** using the **`getPaymentDetails()`** method and ensure that the token is sent as part of the form's data.

`boolean addField(Field f)`

Add a hosted **`Field`** to the `Form`.

Returns true if successful, false otherwise.

`boolean delField(Field f)`

Remove a hosted **`Field`** from the **`Form`**.

Returns true if successful, false otherwise.

promise validate(boolean submitting)

Validate all **Field** values on the **Form**, either during submission or not.

Returns a promise that will be resolved when the validation is complete.

object[] getInvalidElements()

Get details about all invalid FORM controls (not just invalid hosted **Field** elements).

Returns an array of objects containing the following properties:

element	– DOM element.
message	– DOM elements validationMessage property or 'Invalid value'.
label	– associated LABEL text.
field	– **Field** instance (if DOM element is a hosted **Field**).

object getValidationErrors()

Get the validation errors for all invalid FORM controls (not just invalid hosted **Field** elements).

Returns an object whose properties are the associated labels, names or id of the invalid FORM controls and whose values are the error message for that control.

promise getPaymentDetails(object tokenData, boolean validate)

Gets the payment details, generating a **paymentToken** containing the hosted Field values; any values specified by the **tokenise** option; and any passed **tokenData**. The Form will be validated first if required.

Returns a promise that will be resolved when the payment details have been obtained, passing the details as an object containing the following properties:

success	– boolean true if successful, false otherwise.
message	– string containing message to display if not successful.
errors	– object containing details about invalid payment data.
invalid	– object as returned by **getValidationErrors()** method.
paymentToken	– string containing generated **paymentToken**.

void addPaymentToken(string token)

Add the payment token as the value of a Form child INPUT whose name is 'paymentToken', creating the control if needed. Any created control will be given a type of 'hidden'.

void setMerchantID(string merchantID)

Set the **merchantID** used by the payment form.

`void setStylesheet(string selector)`

Set the DOM selector used to select the stylesheet(s) used by the **`Form`**.

`object defaultFieldOptions(string type)`

Get any default field options specified via the **`fields`** option, resulting from the merger of its optional **`any`** and ***`<type>`*** properties.

Returns an object whose properties are the default options.

`void forceSubmit()`

Forcefully submit the FORM **`element`** as if a child submit button had been clicked.

`void reset()`

Reset all the **`Form`**, setting all **`Field`** values back to their initial values.

`void destroy()`

Destroys the **`Form`**, reverting its **`element`** back to its original state.

blink

A-20.3.5 Form Events

The following events may be fired by the **`Form`** object and you can use these to hook into and modify the object's behaviour:

Event Name[1]	Description
`create`	Fired when a **`Form`** has been created.
`destroy`	Fired when a **`Form`** has been destroyed.
`presubmit`	Fired by the **`autoSubmit()`** method prior to handling the submission. You can prevent the handling of the submission and handle it yourself by calling the Events `preventDefault()` method.
`valid`	Fired by the **`autoSubmit()`** method if the FORM contains valid data prior to requesting the payment details. You can prevent the continued handling of the submission and handle it yourself by calling the Events `preventDefault()` method or by invalidating the FORM.
`submit-invalid`	Fired by the **`autoSubmit()`** method if the FORM contains invalid data prior to displaying the validity using the DOM `reportValidity()` method. You can prevent the `reportValidity()` call and display the validity yourself by calling the Events `preventDefault()` method.
`submit`	Fired by the **`autoSubmit()`** method prior to submitting the FORM. You can prevent the FORM from submitting by calling the Events `preventDefault()` method.
`error`	Fired by the **`autoSubmit()`** method if an exception is caught prior to displaying the error, using the JavaScript `alert()` function. You can prevent the `alert()` call and display the error yourself by calling the Events `preventDefault()` method.

[1] Event names are prefixed with the 'hostedform:' namespace not shown in the table.

The **`presubmit`**, **`valid`**, **`submit-invalid`**, **`submit`** and **`error`** events fired by the **`autoSubmit()`** method the payload is an object with the following properties:

`success` – boolean false.
`message` – error message if **`error`** otherwise null.
`invalid` – result of **`getValidationErrors()`** method if **`Form`** invalid.
`submitting` – boolean true.

.

A-20.3.6 Field Construction

The construction method can be used to prepare a HTML INPUT control as a Hosted Payment Field or to create a new field in HTML DIV container. The method signature is as follows:

```
Field(element, options)
```

The **element** parameter should be the DOM node of an existing INPUT or DIV tag.

The **options** parameter should be object containing one of more of the following optional properties:

- **type** – string containing the desired field type.
- **value** – string containing the initial value.
- **placeholder** – string containing any placeholder text.
- **style** – string containing any inline CSS styles.
- **stylesheet** – string containing DOM selector for any stylesheets to be used.
- **disabled** – boolean indicating if initially disabled.
- **required** – boolean indicating if the value is required.
- **readOnly** – boolean indicating if initially read only.
- **validity** – boolean or string indicating the initial validity.
- **locale** – string containing the desired locale.
- **classes** – object containing names of extra CSS classes to use.
- **submitOnEnter** – boolean indicating if the enter key should cause the form to submit.
- **nativeEvents** – boolean indicating that native browser events should be fired.
- **validationMessages** – object containing alternative validation messages.
  - **required** – string containing validation message to use when a value is required.
  - **invalid** – string containing validation message to use when a value is invalid.
- **format** – string containing select option format for date fields.
- **minYear** – integer containing minimum year (relative to current year) for date fields.
- **maxYear** – integer containing maximum year (relative to current year) for date fields.

Any **options** parameter will be merged with those provided via meta data supplied using `data-hostedfield` and/or `data-hostedfield-<option>` attributes, or via existing attributes or properties of the **element** or provided via the **getDefaultOptions()** method of the parent **Form**.

The **type** option can be used to specify the type of Hosted Payment Field required. It defaults to the value provided by any `type` attribute on the **element** (prefixed with the 'hostedfield:' namespace). The option cannot be changed once the **Field** has been created. Valid types are **cardDetails**, **cardNumber**, **cardCVV**, **cardExpiryDate**, **cardStartDate**, **cardIssueNumber**.

The **value** option can be used to specify any initial value that should be used by the **Field**. It defaults to the value provided by any `value` attribute or property on the **element**. Obviously, due to the purpose of the Hosted Payment Fields, any initial value is not wise for card number and CVV fields. The option can be changed at runtime by calling the **setValue()** method.

The **placeholder** option can be used to specify any initial text that should be used as a placeholder by the **Field**. It defaults to the value provided by any `placeholder` attribute or property on the **element**. When used with the **CardDetails** type **Field** the placeholder contains three parts separated by a pipe character, the first part contains the **cardNumber** placeholder, the second part contains the **cardExpiry** placeholder, and the third part contains the **cardCVV** placeholder. The option can be changed at runtime by calling the **setPlaceholder()** method or by altering the options using the jQuery **hostedForm()** plugin method.

The **style** option can be used to specify any initial inline CSS style that should be used by the **Field**. It defaults to the value provided by any `style` attribute or property on the **element**. The option can be changed at runtime by calling the **setStyle()** method or by altering the options using the jQuery **hostedForm()** plugin method.

The **stylesheet** option can be used to specify a DOM selector used to locate stylesheets that should be parsed for styles related to this **Field**. Refer to section on styling fields. The option can be changed at runtime by calling the **setStylesheet()** method or by altering the options using the jQuery **hostedForm()** plugin method.

The **disabled** option can be used to specify if the **Field** should be initially disabled. It defaults to the value provided by any `disabled` attribute or property on the **element**. The option can be changed at runtime by calling the **setDisabled()** method or by altering the options using the jQuery **hostedForm()** plugin method.

The **required** option can be used to specify if the **Field** value is required. It defaults to the value provided by any `required` attribute or property on the **element**. The option can be changed at runtime by calling the **setRequired()** method or by altering the options using the jQuery **hostedForm()** plugin method.

The **readOnly** option can be used to specify if the **Field** should be initially read-only. It defaults to the value provided by any `readOnly` attribute or property on the **element**. The option can be changed at runtime by calling the **setReadOnly()** method or by altering the options using the jQuery **hostedForm()** plugin method.

The **validity** option can be used to specify if the **Field** should be initially marked as invalid. It defaults to the value provided by any `validity` property on the **element**. The option can be changed at runtime by calling the **setValidity()** method or by altering the options using the jQuery **hostedForm()** plugin method.

The **locale** option can be used to specify the language that should be used by the **Field**. It defaults to the value provided by any `lang` attribute or property on the **element** or closest ancestor. The option cannot be changed once the **Field** has been created.

The **classes** options can be used to specify additional CSS class names to add in addition to the default classes documented in section A-20.3.9. The value is an object whose properties are the default class name and whose values are a string containing the additional class name(s) to use. This option will be merged with any classes option provided to the **Form** constructor. The option cannot be changed once the **Form** has been created.

The **submitOnEnter** option can be used to specify if pressing the enter key when typing the **Field** value should cause the **Form** to submit. The option defaults to false and cannot be changed once the **Field** has been created.

The **nativeEvents** option can be used to specify that any associated native event should be fired when a 'hostedfield:' prefixed event is fired. Events are documented in section A-20.3.8. For example, when enabled if the 'hostedfield:mouseover' event is fired then the native 'mouseover' event is also fired. The option defaults to false and cannot be changed once the **Field** has been created.

The **validationMessages** option can be used to specify alternative validation messages that should be displayed when a value is required or invalid. The option defaults to suitable messages depending on the locale and cannot be changed once the **Field** has been created.

The **dropdown** option can be used to specify that a **cardStartDate** or **cardExpiryDate Field** should be displayed as a pair of select controls to select the month and year, otherwise the month and year are entered via a formatted input box instead. The option defaults to false and cannot be changed once the **Field** has been created.

The **format** option can be used in conjunction with the **dropdown** option to specify the format used to display the month and year in the dropdowns. The month and year parts of the format are separated by a pipe character. The option defaults to 'N – M | Y' (e.g. '01 – January | 2020') and cannot be changed once the **Field** has been created.

The following formatting characters are understood:

- **n** – month number (no zero prefixing).
- **N** – month number (zero prefixed to two digits when required).
- **m** – short month name (e.g. Jan, Feb, Mar)
- **M** – long month name (e.g. January, February, March)
- **y** – two digit year number.
- **Y** – four digit year number.

The **minYear** and **maxYear** options can be used in conjunction with the **dropdown** option to specify the minimum and maximum years that are included in the year dropdown. The option defaults to minus 20 to zero for a **cardStartDate Field** or zero to plus 20 for a **cardExpiryDate Field** and cannot be changed once the **Field** has been created.

Example **`Field`** construction is as follows:

```
var field = new window.hostedFields.classes.Field(document.forms[0].elements[0], {
    // Field type
    type: 'cardNumber',

    // Stylesheet selection
    stylesheets: '#hostedfield-stylesheet',

    // Additional CSS classes
    classes: {
        invalid: 'error'
    }
});
```

Or using meta data on the HTML INPUT element:

```
<input type="hostedfield:cardNumber" data-hostedfields='{"stylesheet":"style.hostedform, style.hostedform-card-number"}},"classes":{"invalid":"error"}}'>
<script>
var field = new window.hostedFields.classes.Field{document.forms[0].elements[0]);
</script>
```

A-20.3.7 Field Methods

The follow methods are made available by the **`Field`** class:

`promise validate()`

Validate the **`Field`** value. This will normally be called automatically when the **`Field`** loses focus or the form is submitted, or when an invalid value is modified.

Returns a `promise` that will be resolved when the validation is complete.

`boolean isEmpty()`

Check if the **`Field`** has a value.

Returns true if the field has a value, false otherwise.

`boolean isComplete()`

Check if the **`Field`** has a complete, but not necessarily valid, value. This is mainly used by compound fields such as **cardDetails**, **cardExpiryDate**, **cardStartDate**, which contain multiple input controls and are deemed complete when all their required input controls have values.

Returns true if the value is complete, false otherwise.

`void setStyle()` / **`string getStyle()`**

Set or gets the field's inline CSS style data.

Returns void when setting, or a CSS style string when getting.

`void setStylesheet(string selector)` / **`string getStylesheet()`**

Sets or gets the DOM selector used to select the stylesheet(s) used by the **`Field`**. When setting, the stylesheets are parsed and applied to the **`Field`**.

Returns void when setting, or a DOM selector string when getting.

`void setPlaceholder(string text)` / `string getPlaceholder()`

Sets or gets the placeholder text to be shown when the **`Field`** has no value.

When used with the **CardDetails** type **`Field`** the placeholder contains three parts separated by a pipe character, the first part contains the **cardNumber** placeholder, the second part contains the **cardExpiry** placeholder, and the third part contains the **cardCVV** placeholder.

Returns void when setting, or a text string when getting.

`void setDisabled(boolean disabled)` / `string getDisabled()`

Sets or gets the disabled state of the **`Field`**. When disabled, the field will be greyed out and not be focusable and thus will not react to any input events.

A disabled **`Field`** will have the 'hf-disabled' class added otherwise the 'hf-enabled' class is added.

Returns void when setting, or a boolean representing the state when getting.

`void setRequired(boolean required)` / `string getRequired()`

Sets or gets the required state of the **`Field`**. When required, the field will be invalid if it contains no value or a blank value.

A required **`Field`** will have the 'hf-required' class added otherwise the 'hf-optional' class is added.

Returns void when setting, or a boolean representing the state when getting.

`void setReadOnly(boolean read_only)` / `string getRequired()`

Sets or gets the read-only state of the **`Field`**. When read-only, the field will be not be focusable and thus will not react to any input events.

A read-only **`Field`** will have the 'hf-readonly' class added otherwise the 'hf-readwrite' class is added.

Returns void when setting, or a boolean representing the state when getting.

`void setFocused(boolean focused)`

Moves the browser's focus to the **`Field`**. When focused, the field will react input events.

A focused **`Field`** will have the 'hf-focus' class added otherwise the 'hf-blur' class is added.

Returns void when setting, or a boolean representing the state when getting.

`void setValidity(string validity)` / `string getValidity()`

Sets or gets the validity of the **`Field`**. When valid, the validity will be true or a blank string. When invalid, the validity will be an error message explaining the reason the value is invalid.

When used with the **CardDetails** type **`Field`** the error message contains three parts separated by a pipe character, the first part contains the **cardNumber** value, the second part contains the **cardExpiry** value, and the third part contains the **cardCVV** value.

A valid **`Field`** will have the 'hf-valid' and 'hf-user-valid' classes added otherwise the 'hf-invalid' and 'hf-user-invalid' classes are added.

Returns void when setting, or an error message string when getting.

`void setValue()` / `string getValue()`

Set or gets the **`Field`** value. Because Hosted Payment Fields are designed for the entry of sensitive payment details, then these methods are not normally used. There is no means to retrieve the actual sensitive data and so any returned value will be an empty string if the field has no value or a single asterisk if the field has a value.

When used with the **CardDetails** type **`Field`** the value contains three parts separated by a pipe character, the first part contains the **cardNumber** value, the second part contains the **cardExpiry** value, and the third part contains the **cardCVV** value.

Returns void when setting, or a mask string when getting.

`void getState()`

Get the current state of the **`Field`** as an object with the following boolean properties:

`isReady`	– the **`Field`** has been created, initialised and is ready for use.
`isValid`	– the value is valid (refer to the **`setValidity()`** method).
`isEmpty`	– the value is empty (refer to the **`isEmpty()`** method).
`isComplete`	– the value is complete (refer to the **`isComplete()`** method).
`isDisabled`	– the value is complete (refer to the **`setDisabled()`** method).
`isRequired`	– the value is complete (refer to the **`setRequired()`** method).
`isReadOnly`	– the value is complete (refer to the **`setReadOnly()`** method).

Returns an object containing the states.

`void reset()`

Reset **`Field`** value back to the initial value.

`void destroy()`

Destroys the **`Form,`** reverting its **`element`** back to its original state.

Note: A field's *options or properties cannot be changed while a field is initialising: that is between construction and firing of the* 'ready' event. *Attempts to change field options or properties before this will be ignored.*

blink

A-20.3.8 Field Events

The following events may be fired by the **`Field`** object and you can use these to hook into and modify the object's behaviour:

Event Name[1]	Description
`create`	Fired when a **`Field`** has been created.
`destroy`	Fired when a **`Field`** has been destroyed.
`ready`	Fired when a **`Field`** style is has finished initialising and is ready.
`style`	Fired when a **`Field`** style is changed.
`autofill`	Fired when a **`Field`** has a value auto filled by the browser.
`autofillcancel`	Fired when a **`Field`** has an auto filled value removed.
`valid`	Fired when a **`Field`** is checked for validity and passes the check.
`invalid`	Fired when a **`Field`** is checked for validity and fails the check.
`uservalid`	Fired when the valid event is fired but only after user interaction has occurred, such as focusing a **`Field`**, leaving a **`Field`** or attempting to submit a **`Form`**.
`userinvalid`	Fired when the invalid event is fired but only after user interaction has occurred, such as focusing a **`Field`**, leaving a **`Field`** or attempting to submit a **`Form`**.
`disabled`	Fired when a **`Field`** changes to disabled.
`enabled`	Fired when a **`Field`** changes from disabled.
`required`	Fired when a **`Field`** changes to required.
`optional`	Fired when a **`Field`** changes from required.
`readonly`	Fired when a **`Field`** changes to read-only.
`readwrite`	Fired when a **`Field`** changed from read-only.
`focus`	Fired when a **`Field`** receives focus.
`blur`	Fired when a **`Field`** loses focus.
`mouseenter`	Fired when a pointing device is moved into the **`Field`**.
`mouseleave`	Fired when a pointing device is moved out of the **`Field`**.
`mouseover`	Fired when a pointing device is moved into the **`Field`**.
`mouseout`	Fired when a pointing device is moved out of the **`Field`**.
`mousemove`	Fired when a pointing device is moved over the **`Field`**.
`keydown`	Fired when a key is pressed in the **`Field`**.
`keyup`	Fired when a key is released in a **`Field`**.

`keypress`	Fired when a key except Shift, Fn, CapsLock is in a pressed position in a **`Field`**.
`change`	Fired when an alteration to the value of a **`Field`** is committed by the user.
`input`	Fired when the value of a **`Field`** is changed.

[1] Event names are prefixed with the 'hostedfield:' namespace not shown in the table.

A-20.3.9 Field CSS Classes

The following CSS class names will be added to a **Field** object depending on its state and you can use these to style the object as required:

Event Name	Description
hostedfield	Present on all **Field elements**.
hf-autofill	Present when the value was auto filled by the browser.
hf-invalid	Present when in the invalid state.
hf-valid	Present when in the valid state.
hf-user-invalid	Present when in the invalid state and user interaction has occurred, such as focusing a **Field**, leaving a **Field** or attempting to submit a **Form**.
hf-user-valid	Present when in the valid state and user interaction has occurred, such as focusing a **Field**, leaving a **Field** or attempting to submit a **Form**.
hf-disabled	Present when in the disabled state.
hf-enabled	Present when not in the disabled state.
hf-required	Present when in the required state.
hf-optional	Present when not in the required state.
hf-readonly	Present when in the read-only state.
hf-readwrite	Present when not in the read-only state.
hf-focus	Present when in the focused state.
hf-blur	Present when not in the focused state.
hf-empty	Present when in the empty state.
hf-complete	Present when in the complete state.
hf-hover	Present when a pointing device is over the **Field**.
hf-placeholder-shown	Present when the placeholder text is displayed.

In addition to these class names, the **Field** will add any corresponding class names provided by the **classes** option provided when the **Field** is constructed.

For example if the **Field** is constructed with a **classes** option as follows '`{disabled: 'text-blur text-grey', enabled: 'text-normal'}`', then the 'text-blur' and 'text-grey' class names will be present whenever the 'hf-disabled' class is present and the 'text-normal' class name will be present whenever the 'hf-enabled' class name is present.

blink

A-20.3.10 Field Styling

The Hosted Payment Fields are styled using CSS as normal.

However, styles have to be transferred from your website to the controls served from our website, therefore styles must be isolated and easily identifiable. To aid with identification, all styles intended for a **`Field`** must contain the 'hostedfield' class name in their selector or '-hostedfield' extension on any id in the selector.

As a website may contain lots of stylesheets, a **`Field`** cannot be expected to parse every stylesheet present on the page and therefore it only parses those selected using the stylesheets construction option or using the **`setStylesheet()`** method. By default, this is any stylesheet referenced via a <link> tag or <style> tag with the 'hostedfield' class name: ie any HTML node that matches the following DOM selector 'link.hostedfield[rel=stylesheet], style.hostedfield'.

CSS styles using the **`Field`** state classes, pseudo classes and pseudo elements are supported as follows:

:focus
.hf-focus
:hover
.hf-hover
:enabled
.hf-enabled,
:disabled
.hf-disabled
:valid
.hf-valid
:invalid
.hf-invalid
:user-valid
.hf-user-valid
:user-invalid
.hf-user-invalid
:required
.hf-required
:optional
.hf-optional
:empty
.hf-empty
:complete
.hf-complete
:autofill
.hf-autofill
:placeholder-shown
.hf-placeholder-shown
:readonly
.hf-readonly
:readwrite
.hf-readwrite
:-webkit-auto-fill
.hf-icon
::placeholder
::-moz-placeholder
::-webkit-input-placeholder
::-ms-input-placeholder

The styles can contain any valid CSS rules and will be used to style both the public elements and internal private elements. For security only, styles that relate to the textual representation of the **`Field`** are passed to the internal private elements. This include styles such as colours, font weights and text decorations. At present, it is not possible to specify custom fonts as they would require the font files to be available on our servers.

The following list are the best web safe fonts for HTML and CSS:
Arial (sans-serif)
Verdana (sans-serif)
Helvetica (sans-serif)
Tahoma (sans-serif)
Trebuchet MS (sans-serif)
Times New Roman (serif)
Georgia (serif)
Garamond (serif)
Courier New (monospace)
Brush Script MT (cursive)

blink

The following styles can be used to style the **`Field`** internal private elements:

- caret-color
- color
- cursor
- direction
- fill
- filter
- font
- font-family
- font-feature-settings
- font-kerning
- font-language-override
- font-size
- font-size-adjust
- font-smooth
- font-stretch
- font-style
- font-synthesis
- font-variant
- font-variant-alternates
- font-variant-caps
- font-variant-east-asian
- font-variant-ligatures
- font-variant-numeric
- font-variant-position
- font-weight
- letter-spacing
- line-height
- stroke
- text-align
- text-decoration
- text-decoration-color
- text-decoration-line
- text-decoration-style
- text-emphasis
- text-emphasis-color
- text-emphasis-position
- text-emphasis-style
- text-indent
- text-rendering
- text-shadow
- text-transform
- text-underline-position
- -moz-osx-font-smoothing
- -webkit-font-smoothing
- -webkit-text-fill-color

The '.hf-icon' class name can be used to target the icon sub element in a **`cardDetails Field`**.

Individual controls can be targeted by using DOM ids, which will have a '-hostedfield' extension added to the DOM id of the original **`element`**.

It is advisable to keep CSS selectors and rules as simple as possible to avoid styling errors caused by a failure to parse and filter the rules.

blink

Example stylesheet:

```
<style class="hostedfield">
    /*
     * Style hosted field internals
     * - only accept font, foreground and background styling
     */

    /* Copy of Bootstrap styles */
    .hostedfield:disabled {
        cursor: not-allowed;
        background-color: #eee;
        opacity: 1;
    }

    /* Change text to red when invalid */
    .form-control:invalid,
    .hostedfield:invalid {
        border-color: #a94442 !important;
        color: #a94442 !important;
    }

    /* Change text to light grey when readonly */
    .form-control:readonly,
    .hostedfield:readonly {
        color: lightgrey !important;
    }

    /* Emulate webkit auto fill style */
    .form-control.hf-autofill,
    .hostedfield.hf-autofill {
        background-color: rgb(250, 255, 189) !important;
        background-image: none !important;
        color: rgb(0, 0, 0) !important;
    }

    /* Add pink placeholder */
    .form-control::placeholder,
    .hostedfield::placeholder {
        color: pink;
    }

    /* Show hovering over the control */
    .form-control.hf-hover,
    .hostedfield.hf-hover {
        font-style: italic;
    }

    /* Style by id (hosted field will have '-hostedfield' appended to the id) */
    #form1-card-details.hostedfield, #form1-card-details-hostedfield {
        color: blue;
    }

</style>
```

A-20.3.11 jQuery Plugin

The script will extend the jQuery object with its own plugin methods to allow easy access to **`Form`** and **`Field`** objects attached to an **`element`** as follows:

```
$(element).hostedForm(options);
$(element).hostedForm('instance');
$(element).hostedForm('options', options);
$(element).hostedForm(method, parameters);
$(element).hostedForm('destroy');

$(element).hostedField(options);
$(element).hostedField('instance');
$(element).hostedField('options', options);
$(element).hostedField(method, parameters);
$(element).hostedField('destroy');
```

The script will also add a ':hostedfield' pseudo selector allowing **`Field`** elements to be selected using the following example notation:

```
$('INPUT:hostedfield')
```

:

A-21 Example HTTP Requests

A-21.1 Hosted Integration

A-21.1.1 Transaction Request HTTP Headers

The following HTTP headers are sent for transaction request:

HTTP Header	Mandatory	Description
`content-type`	**Y**	Content type of the request. This must be 'application/x-www-form-urlencoded', A charset parameter is optional and any non UTF-8 request will be converted to UTF-8.

A-21.1.2 Transaction Response HTTP Headers

The following HTTP headers are received for a transaction response:

HTTP header	Description
`Status`	HTTP status header. Possible value are: **200** – Transaction request processed **500** – Internal Server Error **503** – Service Temporarily Unavailable
`content-type`	Content type of the response. This will be 'application/x-www-form-urlencoded'

A-21.1.3 Submission Example

The following shows an example of a transaction request:

```
HTTP/1.1 200 OK
POST /hosted/ HTTP/1.1
Host: gateway.example.com
Accept: */*
Content-Length: 314
Content-Type: application/x-www-form-urlencoded

merchantID=100001&action=SALE&type=1&currencyCode=826&countryCode=826&amount=680&transactionUnique=5de651c7c5379&orderRef=Test+Transaction&redirectURL=https%3A%2F%2Fmyshop.com&signature=ba12b0766a3412782448f154be15e8f61eea390387b1b23d4688c82c9f28f81df593de5890426546cca365943cc7b5c4897c9721b663a0e17680e1e796f1ad55
```

The following shows an example of a transaction response:

```
HTTP/1.1 200 OK
Date: Tue, 01 Jan 2019 09:30:45 GMT
Server: Apache/2.4.23 (Win64) OpenSSL/1.0.2k-fips PHP/5.4.12
Vary: Host
X-Powered-By: PHP/5.4.12
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 4129
Content-Type: text/html

<!DOCTYPE html>
<html>
--- Hosted Payment Page HTML Removed ---
</html>
```

A-21.2 Direct Integration

A-21.2.1 Transaction Request HTTP Headers

The following HTTP headers are sent for transaction request:

HTTP Header	Mandatory	Description
`content-type`	**Y**	Content type of the request. This must be 'application/x-www-form-urlencoded', A charset parameter is optional and any none UTF-8 request will be converted to UTF-8.

A-21.2.2 Transaction Response HTTP Headers

The following HTTP headers are received for a transaction response:

HTTP header	Description
`Status`	HTTP status header. Possible value are: 200 – Hosted Payment Form returned 500 – Internal Server Error 503 – Service Temporarily Unavailable
`content-type`	Content type of the response. This will be 'text/html'

blink

A-21.2.3 Submission Example

The following shows an example of a transaction request:

```
POST /direct/ HTTP/1.1
Host: gateway.example.com
Accept: */*
Content-Length: 397
Content-Type: application/x-www-form-urlencoded

merchantID=100001&action=SALE&type=1&currencyCode=826&countryCode=826&amount=680&transactionUnique=5de65b552499e&orderRef=Test+Transaction&cardNumber=4929+4212+3460+0821&cardCVV=356&cardExpiryDate=1219&threeDSRequired=N&avscv2CheckRequired=N&duplicateDelay=0&signature=06b01e06c8fc761943d676d5f3aa2e9264758fed72e7bcb058a2a35cf23e8e45403099537bb0363054d6bc8ea951ce1ad86e582dbf0b435855b9c97507fcf844
```

The following shows an example of a transaction response:

```
HTTP/1.1 200 OK
Date: Tue, 01 Jan 2019 09:30:45 GMT
Server: Apache/2.4.23 (Win64) OpenSSL/1.0.2k-fips PHP/5.4.12
Vary: Host
X-Powered-By: PHP/5.4.12
Content-Length: 2532
Content-Type: text/html

merchantID=100001&threeDSEnabled=Y&avscv2CheckEnabled=Y&riskCheckEnabled=N&caEnabled=Y&rtsEnabled=Y&cftEnabled=Y&threeDSCheckPref=not+known%2Cnot+checked%2Cauthenticated%2Cattempted+authentication&cv2CheckPref=matched&addressCheckPref=not+known%2Cnot+checked%2Cmatched%2Cpartially+matched&postcodeCheckPref=not+known%2Cnot+checked%2Cmatched%2Cpartially+matched&cardCVVMandatory=Y&riskCheckPref=not+known%3Dfinished%2Cnot+checked%3Ddecline2%2Capprove%3Dcontinue%2Cdecline%3Ddecline1%2Creview%3Ddecline2%2Cescalate%3Dfinished&notifyEmail=an.operator%40merchant.com&customerReceiptsRequired=Y&eReceiptsEnabled=Y&eReceiptsApiKey=C282ZTF885MN0BPL80Q3&eReceiptsStoreID=2&merchantCategoryCode=6013&surchargeEnabled=Y&surchargeRequired=N&surchargeRules%5B0%5D%5BcardType%5D=CC&surchargeRules%5B0%5D%5Bsurcharge%5D=10.1235&surchargeRules%5B1%5D%5BcardType%5D=CC&surchargeRules%5B1%5D%5Bcurrency%5D=GBP&surchargeRules%5B1%5D%5Bsurcharge%5D=2.5000&surchargeRules%5B2%5D%5BcardType%5D=VC&surchargeRules%5B2%5D%5Bsurcharge%5D=3.5000&surchargeRules%5B3%5D%5BcardType%5D=VC&surchargeRules%5B3%5D%5Bcurrency%5D=GBP&surchargeRules%5B3%5D%5Bsurcharge%5D=4.5000&surchargeRules%5B4%5D%5BcardType%5D=DD&surchargeRules%5B4%5D%5Bsurcharge%5D=5.5000&action=SALE&type=1¤cyCode=826&countryCode=826&amount=680&transactionUnique=5de65b552499e&orderRef=Test+Transaction&cardExpiryDate=1219&threeDSRequired=N&avscv2CheckRequired=N&duplicateDelay=0&requestID=5de65b562496f&responseCode=0&responseMessage=AUTHCODE%3A347414&state=captured&requestMerchantID=100001&processMerchantID=100001&paymentMethod=card&cardType=Visa+Credit&cardTypeCode=VC&cardScheme=Visa+&cardSchemeCode=VC&cardIssuer=BARCLAYS+BANK+PLC&cardIssuerCountry=United+Kingdom&cardIssuerCountryCode=GBR&cardFlags=8323072&cardNumberMask=492942%2A%2A%2A%2A%2A0821&cardNumberValid=Y&xref=19120312NG55CM51QH35JRL&cardExpiryMonth=12&cardExpiryYear=19&authorisationCode=347414&transactionID=10018201&responseStatus=0×tamp=2019-12-03+12%3A55%3A52&amountApproved=680&amountReceived=680&amountRetained=680&avscv2ResponseCode=244100&avscv2ResponseMessage=SECURITY+CODE+MATCH+ONLY&avscv2AuthEntity=merchant+host&cv2Check=matched&addressCheck=not+matched&postcodeCheck=not+matched&notifyEmailResponseCode=0&notifyEmailResponseMessage=Email+successfully+queued&vcsResponseCode=0&vcsResponseMessage=Success+-+no+velocity+check+rules+applied&currencyExponent=2&signature=e5c65e5d0340e0ec0de8782affcb6caba2e4d202a6873a1677ddbf6415cb1dd52cc597e43c758b233afd121367d300a57d0faade7abf6b4b88a1a1b974e55d33
```

A-21.3 Batch Integration

A-21.3.1 Submission Request HTTP Headers

The following HTTP headers are sent for batch submission request:

HTTP Header	Mandatory	Description
`content-type`	**Y**	Content type of the batch request. This must be 'multipart/mixed' and contain a boundary parameter to separate each transaction request. A charset parameter is optional and any none UTF-8 request will be converted to UTF-8.
`content-encoding`	**N**	Optional content encoding applied to the request. The value should be a comma separated list of one or more: **x-gzip**, **gzip**, **base64**.
`authorization`	**N**	Optional username and password to authenticate the submitter

The following HTTP headers are sent on each individual part request:

HTTP Header	Mandatory	Description
`content-type`	**Y**	Content type of the individual request. This must be 'application/x-www-form-urlencoded', A charset parameter is optional and any none UTF-8 request will be converted to UTF-8.
`content-encoding`	**N**	Optional content encoding applied to the request. The value should be a comma separated list of one or more: **x-gzip**, **gzip**, **base64**.
`content-id`	**N**	Optional identifier for each individual transaction with the batch. The Gateway will return this identifier in the submission response. If not sent, the Gateway will generate a unique identifier for each transaction.

blink

A-21.3.2 Submission Response HTTP Headers

The following HTTP headers are received for batch submission response:

HTTP header	Description
`status`	HTTP status header. Possible value are: **200** – Batch submission status response ok **201** – Batch submission received and stored **400** – Batch submission invalid **401** – Unauthorised (none or incorrect credentials) **405** – HTTP method was not POST/PUT or GET **500** – Internal Gateway error
`location`	URL to use to monitor the status of the batch. A unique batch reference number will be provided in the URL in the format: XXXX-XXXX-XXXX-XXXX (e.g. 1A23-B4C5-DEF6-G7HI) This reference number is used to request information about the status of a batch via HTTP GET requests to the URL endpoint as outlined in section 1.3.3.
`x-p3-token`	If user authentication was sent in the initial request, this header will contain a token that can be used for future requests for the status of the batch instead of having to use a username/password.
`content-type`	Content type of the HTTP batch request. This will be 'multipart/mixed' and contain a boundary parameter to separate each transaction request.

The following HTTP headers are received on each individual part response:

HTTP header	Description
`content-type`	Content type of the individual request. This will be 'application/x-www-form-urlencoded', A charset parameter is optional and any none UTF-8 request will be converted to UTF-8.
`content-id`	The content ID sent in the initial request as outlined in A-21.3.1. If no content-id header was sent, the Gateway will return a unique content ID per transaction.
`x-transaction-id`	The Gateway transaction ID. This will be empty if the transaction is currently pending in this stage.
`x-transaction-response`	A message containing the current status of the transaction. Possible value are: **skipped** – insufficient permissions to view transaction **pending** – queued for processing **success** – (Response Message) **failure** – (Response Message)

blink

A-21.3.3 Status Request HTTP Headers

The following HTTP headers are used during a batch status request:

HTTP Header	Mandatory	Description
authorization	**Y**	Mandatory username and password to authenticate the submitter

A-21.3.4 Status Response HTTP Headers

The batch status response is identical to the submission status response as documented in section A-21.3.2.

A-21.3.5 Submission Example

The following shows an example of a batch submission request:

```
PUT /batch/?validate=0 HTTP/1.1
Authorization: Basic bmljay50dXJuZXI6dGVzdGluZzI=
Host: gateway.example.com
Accept: */*
Content-type: multipart/mixed; charset=UTF-8; boundary=5de63a42507a9
Content-length: 1404

--5de63a42507a9
Content-Id: TX5de63a42507ac
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

merchantID=100001&action=SALE&type=1&currencyCode=826&countryCode=826&amount=680&transactionUnique=5de63a42507ac&orderRef=Test+Transaction&cardNumber=4929+4212+3460+0821&cardExpiryDate=1219&duplicateDelay=0&signature=3cd686fdd40449ef33534baa62732c95fc127ff591fae3b5b611ccb38573ad921d199396e27cffd14faa4f46df8dde310252920fd1b33607b029b9b6ff669e2b

--5de63a42507a9
Content-Id: TX5de63a42af062
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

merchantID=100001&action=SALE&type=1&currencyCode=826&countryCode=826&amount=681&transactionUnique=5de63a42af062&orderRef=Test+Transaction&cardNumber=4929+4212+3460+0821&cardExpiryDate=1219&duplicateDelay=0&signature=55f411d40954be7f7089e84fe489438f09fc1b37c0964e46b0fab8bdcb44e13ed3ea11b9deb9da89a6d7b45133709a126bd3581f6329bf888b83231184597231

--5de63a42507a9
Content-Id: TX5de63a42ca9cd
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

merchantID=100001&action=SALE&type=1&currencyCode=826&countryCode=826&amount=682&transactionUnique=5de63a42ca9cd&orderRef=Test+Transaction&cardNumber=4929+4212+3460+0821&cardExpiryDate=1219&duplicateDelay=0&signature=c296266cb9bc8082957c700da9651d98add176dd8bd62eb3b7098566c7d8e23a3426b776de815e99149c6681978b1addedac762339563732d8a449b6cca3a3c2

--5de63a42507a9--
```

The following shows an example of a batch submission response:

```
1.  HTTP/1.1 201 Created
2.  Date: Tue, 01 Jan 2019 09:30:45 GMT
3.  Server: Apache/2.4.23 (Win64) OpenSSL/1.0.2k-fips PHP/5.4.12
4.  X-Powered-By: PHP/5.4.12
5.  x-p3-token: YTo1OntzOjc6InZlcnNpb24iO3M6ODoiUDNUSy8yLjAiO3M6NzoicHVycG9zZSI7czo0OiJhdXRoIjtzOjc6ImNyZWF0b3IiO3M
    6NToiQkFUQ0giO3M6NzoiY3JlYXRlZCI7aToxNTc1MzY5Mjg1O3M6NzoiZXhwaXJlcyI7aToxNTc1MzcyODg1O30.czozOiI2MjkiOw.zdfxxXY
    tC2Wc4yyk-lEos-wZ99pEJtPGYpXR4KCiWW_56nmOysarOaMucrWPIt-NzwFzgq3-7u4Ud6uYkQcWBQ
6.  Location: /batch/2D6D-AC2C-BF55-2A8C
7.  Content-disposition: attachment; filename="batch-2D6D-AC2C-BF55-2A8C"
8.  Content-Length: 1857
9.  Content-Type: multipart/mixed; charset=UTF-8; boundary=5de63a5c1a071
10.
11. Transaction 'TX5de63a42507ac' - pending - queued for processing
12. Transaction 'TX5de63a42af062' - pending - queued for processing
13. Transaction 'TX5de63a42ca9cd' - pending - queued for processing
14.
15. --5de63a5c1a071
16. Content-Id: TX5de63a42507ac
17. Content-Type: application/x-www-form-urlencoded; charset=UTF-8
18. X-Transaction-ID:
19. X-Transaction-Response: pending - queued for processing
20.
21. merchantID=100001&action=SALE&type=1¤cyCode=826&countryCode=826&amount=680&transactionUnique=5de63a42507ac&orde
    rRef=Test+Transaction&cardNumber=492942%2A%2A%2A%2A%2A0821&cardExpiryDate=1219&duplicateDelay=0&signature=03
    84bbf6ca0fc153e1e27a0cfc51f3b1cd1c2cff7a49aa4e9439bba38262183e9ac7d156f218eba1ef8d04f9e6a7fa6fbc9c2b3ab990c70e0
    6dc7c6923e5b27b
22.
23. --5de63a5c1a071
24. Content-Id: TX5de63a42af062
25. Content-Type: application/x-www-form-urlencoded; charset=UTF-8
26. X-Transaction-ID:
27. X-Transaction-Response: pending - queued for processing
28.
29. merchantID=100001&action=SALE&type=1¤cyCode=826&countryCode=826&amount=681&transactionUnique=5de63a42af062&orde
    rRef=Test+Transaction&cardNumber=492942%2A%2A%2A%2A%2A0821&cardExpiryDate=1219&duplicateDelay=0&signature=1e
    13e23c2b90a30f4403d604ac20302b5504b886b0b5c9ace0764fc8d966d120f5a1beca975805292780c22953b4e6ca71f67f499804f19d2
    718518463a598c4
30.
31. --5de63a5c1a071
32. Content-Id: TX5de63a42ca9cd
33. Content-Type: application/x-www-form-urlencoded; charset=UTF-8
34. X-Transaction-ID:
35. X-Transaction-Response: pending - queued for processing
36.
37. merchantID=100001&action=SALE&type=1¤cyCode=826&countryCode=826&amount=682&transactionUnique=5de63a42ca9cd&orde
    rRef=Test+Transaction&cardNumber=492942%2A%2A%2A%2A%2A0821&cardExpiryDate=1219&duplicateDelay=0&signature=c4
    56aa211f8e3e568a40051bfd38406be02566fcd72d3bb1547f4d43e75db1d069eaa4158aa035337cac084633df945a13471db6b1a3fcd6c
    0749626d9bc0044
38.
39. --5de63a5c1a071--
```

blink

A-22 Example Integration Code

The follow section provides samples of how to integrate with the Gateway using the PHP scripting language to communicate directly with the API without the use of any our SDKs.

A-22.1 Hosted Integration

A-22.1.1 Sale Transaction

The following example PHP code shows how to send a SALE transaction:

```php
<?PHP

// Signature key entered on MMS. The demo account is fixed to this value,
$key = 'Circle4Take40Idea';

// Gateway URL
$url = 'https://gateway.example.com/hosted/';


if (!isset($_POST['responseCode'])) {
    // Send request to gateway

    // Request
    $req = array(
        'merchantID' => '100001',
        'action' => 'SALE',
        'type' => 1,
        'countryCode' => 826,
        'currencyCode' => 826,
        'amount' => 1001,
        'orderRef' => 'Test purchase',
        'transactionUnique' => uniqid(),
        'redirectURL' => ($_SERVER['HTTPS'] == 'on' ? 'https' : 'http') . '://' . $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI'],
    );

    // Create the signature using the function called below.
    $req['signature'] = createSignature($req, $key);

    echo '<form action="' . htmlentities($url) . '" method="post">' . PHP_EOL;

    foreach ($req as $field => $value) {
        echo '    <input type="hidden" name="' . $field . '" value="' . htmlentities($value) . '">' . PHP_EOL;
    }

    echo '    <input type="submit" value="Pay Now">' . PHP_EOL;
    echo '</form>' . PHP_EOL;

// Check the return signature
    if (!$signature || $signature !== createSignature($res, $key)) {
        // You should exit gracefully
        die('Sorry, the signature check failed');
    }

    // Check the response code
    if ($res['responseCode'] === "0") {
        echo "<p>Thank you for your payment.</p>";
    } else {
        echo "<p>Failed to take payment: " . htmlentities($res['responseMessage']) .  "</p>";
    }

}

```

```php
// Function to create a message signature
function createSignature(array $data, $key) {
    // Sort by field name
    ksort($data);

    // Create the URL encoded signature string
    $ret = http_build_query($data, '', '&');

    // Normalise all line endings (CRNL|NLCR|NL|CR) to just NL (%0A)
    $ret = str_replace(array('%0D%0A', '%0A%0D', '%0D'), '%0A', $ret);

    // Hash the signature string and the key together
    return hash('SHA512', $ret . $key);
}

?>
```

A-22.2 Direct Integration

A-22.2.1 Sale Transaction (with 3-D Secure)

The following example PHP code shows how to send a SALE transaction with support for 3-D Secure:

```php
<?PHP

// Signature key entered on MMS. The demo account is fixed to this value,
$key = 'Circle4Take40Idea';

// Gateway URL
$url = 'https://gateway.example.com/direct/';

// Setup PHP session as use it to store data between 3DS steps
if (isset($_GET['sid'])) {
    session_id($_GET['sid']);
}

session_start();

// Compose current page URL (removing any sid and acs parameters)
$pageUrl = ((isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] == 'on') ? 'https://' : 'http://')
    . $_SERVER['SERVER_NAME'] . ($_SERVER['SERVER_PORT'] != '80' ? ':' . $_SERVER['SERVER_PORT'] : '')
    . preg_replace('/(sid=[^&]+&?)|(acs=1&?)/', '', $_SERVER['REQUEST_URI']);

// Add back the correct sid parameter (used as session cookie may not be passed when the page is redirected fro
m an IFRAME)
$pageUrl .= (strpos($pageUrl, '?') === false ? '?' : '&') . 'sid=' . urlencode(session_id());


// If ACS response into the IFRAME then redirect back to parent window
if (!empty($_GET['acs'])) {
    echo silentPost($pageUrl, array('threeDSResponse' => $_POST), '_parent');
    exit();
}

if (!isset($_POST['threeDSResponse'])) {
    // Initial request

    // Gather browser info - can be done at any time prior to the checkout
    if (!isset($_POST['browserInfo'])) {
        echo collectBrowserInfo();
        exit();
    }

    // Direct Request
    $req = array(
        'merchantID' => 100001,
        'action' => 'SALE',
        'type' => 1,
        'currencyCode' => 826,
        'countryCode' => 826,
        'amount' => 1001,
        'cardNumber' => '4012001037141112',
        'cardExpiryMonth' => 12,
        'cardExpiryYear' => 15,
        'cardCVV' => '083',
        'customerName' => 'Test Customer',
        'customerEmail' => 'test@testcustomer.com',
        'customerAddress' => '16 Test Street',
        'customerPostCode' => 'TE15 5ST',
        'orderRef' => 'Test purchase',

        // The following fields are mandatory for 3DS v2
```

```
        'remoteAddress' => $_SERVER['REMOTE_ADDR'],
        'threeDSRedirectURL' => $pageUrl . '&acs=1',
    );

    // Add the browser info as it is mandatory for 3DS v2
    $req += $_POST['browserInfo'];

} else {
    // 3DS continuation request
    $req = array(
        'threeDSRef' => $_SESSION['threeDSRef'],
        'threeDSResponse' => $_POST['threeDSResponse'],
    );

}

// Create the signature using the function called below.
$req['signature'] = createSignature($req, $key);

// Initiate and set curl options to post to the gateway
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($req));
curl_setopt($ch, CURLOPT_HEADER, false);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Send the request and parse the response
parse_str(curl_exec($ch), $res);

// Close the connection to the gateway
curl_close($ch);

// Extract the return signature as this isn't hashed
$signature = null;
if (isset($res['signature'])) {
    $signature = $res['signature'];
    unset($res['signature']);
}

// Check the return signature
if (!$signature || $signature !== createSignature($res, $key)) {
    // You should exit gracefully
    die('Sorry, the signature check failed');
}

// Check the response code
if ((int)$res['responseCode'] === 65802) {
    // Send request to the ACS server displaying response in an IFRAME

    // Render an IFRAME to show the ACS challenge (hidden for fingerprint method)
    $style = (isset($res['threeDSRequest']['threeDSMethodData']) ? 'display: none;' : '');
    echo "<iframe name=\"threeds_acs\" style=\"height:420px; width:420px; {$style}\"></iframe>\n";

    // Silently POST the 3DS request to the ACS in the IFRAME
    echo silentPost($res['threeDSURL'], $res['threeDSRequest'], 'threeds_acs');

    // Remember the threeDSRef as need it when the ACS responds
    $_SESSION['threeDSRef'] = $res['threeDSRef'];

} else if ((int)$res['responseCode'] === 0) {
    echo "<p>Thank you for your payment.</p>";
} else {
    echo "<p>Failed to take payment: " . htmlentities($res['responseMessage']) . "</p>";
}

// Function to create a message signature
```

```php
function createSignature(array $data, $key) {
    // Sort by field name
    ksort($data);

    // Create the URL encoded signature string
    $ret = http_build_query($data, '', '&');

    // Normalise all line endings (CRNL|NLCR|NL|CR) to just NL (%0A)
    $ret = str_replace(array('%0D%0A', '%0A%0D', '%0D'), '%0A', $ret);

    // Hash the signature string and the key together
    return hash('SHA512', $ret . $key);
}

// Return HTML to render a hidden form used to collect some browser details
function collectBrowserInfo(array $options = null) {

    $form_attrs = 'id="collectBrowserInfo" method="post" action="?"';

    if (isset($options['formAttrs'])) {
        $form_attrs .= $options['formAttrs'];
    }

    $device_data = array(
        'deviceChannel'             => 'browser',
        'deviceIdentity'            => (isset($_SERVER['HTTP_USER_AGENT']) ? htmlentities($_SERVER['HTTP_USER_AGENT']) : null),
        'deviceTimeZone'            => '0',
        'deviceCapabilities'        => '',
        'deviceScreenResolution'    => '1x1x1',
        'deviceAcceptContent'       => (isset($_SERVER['HTTP_ACCEPT']) ? htmlentities($_SERVER['HTTP_ACCEPT']) : null),
        'deviceAcceptEncoding'      => (isset($_SERVER['HTTP_ACCEPT_ENCODING']) ? htmlentities($_SERVER['HTTP_ACCEPT_ENCODING']) : null),
        'deviceAcceptLanguage'      => (isset($_SERVER['HTTP_ACCEPT_LANGUAGE']) ? htmlentities($_SERVER['HTTP_ACCEPT_LANGUAGE']) : null),
        'deviceAcceptCharset'       => (isset($_SERVER['HTTP_ACCEPT_CHARSET']) ? htmlentities($_SERVER['HTTP_ACCEPT_CHARSET']) : null),
    );

    $form_fields = fieldToHtml('browserInfo', $device_data);

    if (isset($options['formData'])) {
        foreach ((array)$options['formData'] as $name => $value) {
            $form_fields .= fieldToHtml($name, $value);
        }
    }

    $ret = <<<EOS
        <form {$form_attrs}>
            {$form_fields}
        </form>
        <script>
            var screen_width = (window && window.screen ? window.screen.width : '0');
            var screen_height = (window && window.screen ? window.screen.height : '0');
            var screen_depth = (window && window.screen ? window.screen.colorDepth : '0');
            var identity = (window && window.navigator ? window.navigator.userAgent : '');
            var language = (window && window.navigator ? (window.navigator.language ? window.navigator.language : window.navigator.browserLanguage) : '');
            var timezone = (new Date()).getTimezoneOffset();
            var java = (window && window.navigator ? navigator.javaEnabled() : false);
            var fields = document.forms.collectBrowserInfo.elements;
            fields['browserInfo[deviceIdentity]'].value = identity;
            fields['browserInfo[deviceTimeZone]'].value = timezone;
            fields['browserInfo[deviceCapabilities]'].value = 'javascript' + (java ? ',java' : '');
            fields['browserInfo[deviceAcceptLanguage]'].value = language;
```

```
            fields['browserInfo[deviceScreenResolution]'].value = screen_width + 'x' + screen_height + 'x' + sc
reen_depth;
            window.setTimeout('document.forms.collectBrowserInfo.submit()', 0);
        </script>
EOS;

    return $ret;
}

// Render HTML to silently POST data to URL in target brower window
function silentPost($url = '?', array $post = null, $target = '_self') {

    $url = htmlentities($url);
    $target = htmlentities($target);
    $fields = '';

    if ($post) {
        foreach ($post as $name => $value) {
            $fields .= fieldToHtml($name, $value);
        }
    }

    $ret = "
        <form id=\"silentPost\" action=\"{$url}\" method=\"post\" target=\"{$target}\">
            {$fields}
            <noscript><input type=\"submit\" value=\"Continue\"></noscript
        </form>
        <script>
            window.setTimeout('document.forms.silentPost.submit()', 0);
        </script>
    ";

    return $ret;
}

// Return a value as hidden HTML FORM fields
function fieldToHtml($name, $value) {
    $ret = '';
    if (is_array($value)) {
        foreach ($value as $n => $v) {
            $ret .= fieldToHtml($name . '[' . $n . ']', $v);
        }
    } else {
        // Convert all applicable characters or none printable characters to HTML entities
        $value = preg_replace_callback('/[\x00-
\x1f]/', function($matches) { return '&#' . ord($matches[0]) . ';'; }, htmlentities($value, ENT_COMPAT, 'UTF-
8', true));
        $ret = "<input type=\"hidden\" name=\"{$name}\" value=\"{$value}\" />\n";
    }

    return $ret;
}


?>
```

A-22.2.2 Sale Transaction (without 3-D Secure)

The following sample PHP code shows how to send a SALE transaction without support for 3-D Secure:

```php
<?PHP

// Signature key entered on MMS. The demo account is fixed to this value,
$key = 'Circle4Take40Idea';

// Gateway URL
$url = 'https://gateway.example.com/direct/';

// Request
$req = array(
    'merchantID' => '100001',
    'action' => 'SALE',
    'type' => 1,
    'countryCode' => 826,
    'currencyCode' => 826,
    'amount' => 1001,
    'cardNumber' => '4012001037141112',
    'cardExpiryMonth' => 12,
    'cardExpiryYear' => 15,
    'cardCVV' => '083',
    'customerName' => 'Test Customer',
    'customerEmail' => 'test@testcustomer.com',
    'customerPhone' => '+44 (0) 123 45 67 890',
    'customerAddress' => '16 Test Street',
    'customerPostCode' => 'TE15 5ST',
    'orderRef' => 'Test purchase',
    'transactionUnique' => uniqid(),
);

// Create the signature using the function called below.
$req['signature'] = createSignature($req, $key);

// Initiate and set curl options to post to the gateway
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($req));
curl_setopt($ch, CURLOPT_HEADER, false);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Send the request and parse the response
parse_str(curl_exec($ch), $res);

// Close the connection to the gateway
curl_close($ch);

// Extract the return signature as this isn't hashed
$signature = null;
if (isset($res['signature'])) {
    $signature = $res['signature'];
    unset($res['signature']);
}

// Check the return signature
if (!$signature || $signature !== createSignature($res, $key)) {
    // You should exit gracefully
    die('Sorry, the signature check failed');
}

// Check the response code
if ($res['responseCode'] === "0") {
    echo "<p>Thank you for your payment.</p>";
} else {
```

```
64.     echo "<p>Failed to take payment: " . htmlentities($res['responseMessage']) .  "</p>";
65. }
66. 
67. // Function to create a message signature
68. function createSignature(array $data, $key) {
69.     // Sort by field name
70.     ksort($data);
71. 
72.     // Create the URL encoded signature string
73.     $ret = http_build_query($data, '', '&');
74. 
75.     // Normalise all line endings (CRNL|NLCR|NL|CR) to just NL (%0A)
76.     $ret = str_replace(array('%0D%0A', '%0A%0D', '%0D'), '%0A', $ret);
77. 
78.     // Hash the signature string and the key together
79.     return hash('SHA512', $ret . $key);
80. }
81. 
82. ?>
```

A-22.3 Batch Integration

A-22.3.1 Batch Submission

The following example PHP code shows how to send a batch request containing three SALE transactions:

```php
<?PHP

// Signature key entered on MMS. The demo account is fixed to this value,
$key = 'Circle4Take40Idea';

// Gateway URL
$url = 'https://gateway.example.com/batch/';

// Create a unique multipart boundary
$boundary = uniqid();

// Requests
$reqs = array(
    array(
        'merchantID' => 100001,
        'action' => 'SALE',
        'type' => 1,
        'currencyCode' => 826,
        'countryCode' => 826,
        'amount' => 1001,
        'cardNumber' => '4012001037141112',
        'cardExpiryMonth' => 12,
        'cardExpiryYear' => 15,
        'cardCVV' => '083',
        'customerName' => 'Test Customer',
        'customerEmail' => 'test@testcustomer.com',
        'customerAddress' => '16 Test Street',
        'customerPostCode' => 'TE15 5ST',
        'orderRef' => 'Test purchase',
        'transactionUnique' => uniqid(),
        'threeDSRequired' => 'N',
        'avscv2CheckRequired' => 'N',
    ),
    array(
        'merchantID' => 100001,
        'action' => 'SALE',
        'type' => 1,
        'currencyCode' => 826,
        'countryCode' => 826,
        'amount' => 2002,
        'cardNumber' => '4012001037141112',
        'cardExpiryMonth' => 12,
        'cardExpiryYear' => 15,
        'cardCVV' => '083',
        'customerName' => 'Test Customer',
        'customerEmail' => 'test@testcustomer.com',
        'customerAddress' => '16 Test Street',
        'customerPostCode' => 'TE15 5ST',
        'orderRef' => 'Test purchase',
        'transactionUnique' => uniqid(),
        'threeDSRequired' => 'N',
        'avscv2CheckRequired' => 'N',
    ),
    array(
        'merchantID' => 100001,
        'action' => 'SALE',
        'type' => 1,
        'currencyCode' => 826,
        'countryCode' => 826,
```

```
        'amount' => 3003,
        'cardNumber' => '4012001037141112',
        'cardExpiryMonth' => 12,
        'cardExpiryYear' => 15,
        'cardCVV' => '083',
        'customerName' => 'Test Customer',
        'customerEmail' => 'test@testcustomer.com',
        'customerAddress' => '16 Test Street',
        'customerPostCode' => 'TE15 5ST',
        'orderRef' => 'Test purchase',
        'transactionUnique' => uniqid(),
        'threeDSRequired' => 'N',
        'avscv2CheckRequired' => 'N',
    ),
);

// Create the batch parts
$parts = array();
foreach ($reqs as $req) {

    // Create the signature using the function called below.
    $req['signature'] = createSignature($req, $key);

    $parts[] =
        "Content-Id: TX{$req['transactionUnique']}\r\n" .
        "Content-Type: application/x-www-form-urlencoded; charset=\"UTF-8\"\r\n" .
        "\r\n" .
        http_build_query($req);
}

// Join the parts together separated by the boundary string
$post = "\r\n--{$boundary}\r\n" . join("\r\n--{$boundary}\r\n", $parts) . "\r\n--{$boundary}--\r\n";

// Initiate and set curl options to post to the gateway
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, $post);
curl_setopt($ch, CURLOPT_HEADER, true);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, array(
    'Content-type: multipart/mixed; charset="UTF-8"; boundary=' . $boundary,
    'Content-length: ' . strlen($post),
));

// Send the request
$res = curl_exec($ch);

// Normally would process the response here, but for this example just echo it out
header ('Content-Type: text/plain');
echo $res . PHP_EOL;

// Close the connection to the gateway
curl_close($ch);

// Function to create a message signature
function createSignature(array $data, $key) {
    // Sort by field name
    ksort($data);

    // Create the URL encoded signature string
    $ret = http_build_query($data, '', '&');

    // Normalise all line endings (CRNL|NLCR|NL|CR) to just NL (%0A)
    $ret = str_replace(array('%0D%0A', '%0A%0D', '%0D'), '%0A', $ret);

    // Hash the signature string and the key together
```

```
127.     return hash('SHA512', $ret . $key);
128. }
129. 
130. ?>
```

blink

A-23 Example Library Code

The follow section provides samples of how to integrate with the Gateway using our integration libraries as documented in section A-20.1.

A-23.1 Gateway Integration Library

A-23.1.1 Hosted Sale Transaction

The following example PHP code shows how to send a SALE transaction using the Gateway library:

```php
<?PHP
require('gateway.php');

use \P3\SDK\Gateway;

// Signature key entered on MMS. The demo account is fixed to this value,
Gateway::$merchantSecret = 'Circle4Take40Idea';

// Gateway URL
Gateway::$hostedUrl = 'https://gateway.example.com/hosted/';

if (!isset($_POST['responseCode'])) {
    // Send request to gateway
    $req = array(
        'merchantID' => 100001,
        'action' => 'SALE',
        'type' => 1,
        'currencyCode' => 826,
        'countryCode' => 826,
        'amount' => 1001,
        'orderRef' => 'Test purchase',
        'redirectURL' => ($_SERVER['HTTPS'] == 'on' ? 'https' : 'http') . '://' . $_SERVER['HTTP_HOST'] . $_SERVER['REQUEST_URI'],
    );

    try {
        echo Gateway::hostedRequest($req);
    } catch (\Exception $e) {
        // You should exit gracefully
        die('Sorry, the request could not be sent: ' . $e);
    }

} else {
    // Received response from gateway
    try {
        Gateway::verifyResponse($_POST);
    } catch(\Exception $e) {
        // You should exit gracefully
        die('Sorry, the request could not be sent: ' . $e);
    }

    // Check the response code
    if ($_POST['responseCode'] === 0) {
        echo "<p>Thank you for your payment.</p>";
    } else {
        echo "<p>Failed to take payment: " . htmlentities($_POST['responseMessage']) .  "</p>";
    }
}

?>
```

A-23.1.2 Direct Sale Transaction (with 3-D Secure)

The following example PHP code shows how to send a SALE transaction with support for 3-D Secure using the Gateway library:

```php
<?PHP

require('gateway.php');

use \P3\SDK\Gateway;

// Signature key entered on MMS. The demo account is fixed to this value,
Gateway::$merchantSecret = 'Circle4Take40Idea';

// Gateway URL
Gateway::$directUrl = 'https://gateway.example.com/direct/';

// Requests
$req = array(
    'merchantID' => 100001,
    'action' => 'SALE',
    'type' => 1,
    'currencyCode' => 826,
    'countryCode' => 826,
    'amount' => 1001,
    'cardNumber' => '4012001037141112',
    'cardExpiryMonth' => 12,
    'cardExpiryYear' => 15,
    'cardCVV' => '083',
    'customerName' => 'Test Customer',
    'customerEmail' => 'test@testcustomer.com',
    'customerAddress' => '16 Test Street',
    'customerPostCode' => 'TE15 5ST',
    'orderRef' => 'Test purchase',
    'threeDSMD' => (isset($_POST['MD']) ? $_POST['MD'] : null),
    'threeDSPaRes' => (isset($_POST['PaRes']) ? $_POST['PaRes'] : null),
    'threeDSPaReq' => (isset($_POST['PaReq']) ? $_POST['PaReq'] : null)
);

try {
    $res = Gateway::directRequest($req);
} catch (\Exception $e) {
    // You should exit gracefully
    die('Sorry, the required could not be sent: ' . $e);
}

// Check the response code
if ($res['responseCode'] === 65802) {

    // Send details to 3D Secure ACS and the return here to repeat request
    $pageUrl = (@$_SERVER['HTTPS'] == 'on') ? 'https://' : 'http://';
    if ($_SERVER['SERVER_PORT'] != '80') {
        $pageUrl .= $_SERVER['SERVER_NAME'] . ':' . $_SERVER['SERVER_PORT'] . $_SERVER['REQUEST_URI'];
    } else {
        $pageUrl .= $_SERVER['SERVER_NAME'] . $_SERVER['REQUEST_URI'];
    }

    echo "
<p>Your transaction requires 3D Secure Authentication</p>
<form action=\"" . htmlentities($res['threeDSACSURL']) . "\"method=\"post\">
<input type=\"hidden\" name=\"MD\" value=\"" . htmlentities($res['threeDSMD']) . "\">
<input type=\"hidden\" name=\"PaReq\" value=\"" . htmlentities($res['threeDSPaReq']) . "\">
<input type=\"hidden\" name=\"TermUrl\" value=\"" . htmlentities($pageUrl) . "\">
<input type=\"submit\" value=\"Continue\">
</form>
";
```

```
62.
63. } else if ($res['responseCode'] === 0) {
64.     echo "<p>Thank you for your payment.</p>";
65. } else {
66.     echo "<p>Failed to take payment: " . htmlentities($res['responseMessage']) .  "</p>";
67. }
68.
69. ?>
```

A-23.2 Hosted Payment Page Library

A-23.2.1 Hosted Sale Transaction

The following example code shows how to prepare a payment form to open the Hosted Payment Page in a lightbox style overlay on your website using the Hosted Payment Page library:

```
<html>
  <head>
    <!-- Load the Hosted Payment Page library -->
    <script src="https://gateway.example.com/sdk/web/v1/js/hostedforms.min.js"></script>
  </head>
  <body>
    <!--
      Hosted Payment <form> as created by the Gateway Integration Library hostedRequest() method
      with addition of 'data-hostedforms-modal' attribute to signify a modal form is required.
    -->
    <form name="payment-form" method="post" action="https://gateway.example.com/hosted/" data-hostedforms-modal>
      <input type="hidden" name="merchantID" value="100001" />
      <input type="hidden" name="action" value="SALE" />
      <input type="hidden" name="type" value="1" />
      <input type="hidden" name="currencyCode" value="826" />
      <input type="hidden" name="countryCode" value="826" />
      <input type="hidden" name="amount" value="1001" />
      <input type="hidden" name="orderRef" value="Test purchase" />
      <input type="hidden" name="redirectURL" value="https://www.merchant.com/payment/" />
      <input type="hidden" name="signature" value="07599ef4cdb2e26cb2bf34a9c65190a7ce82494bc1df144c3bb0d20ee2655d8278dc663b2b0421ef12b8f081e821151bb4c644277c5d65b5523a96539b53b5aa" />
      <input type="submit" value="Pay Now">
    </form>
    <script>
        // Create a new Hosted Form object which will cause the above <form> to load into a modal
        // overlay over this page.
        var form = new window.hostedForms.classes.Form(document.forms[0]);
    </script>
  </body>
</html>
```

A-23.2.2 Hosted Sale Transaction (jQuery)

The following example code shows how to prepare a payment form to open the Hosted Payment Page in a lightbox style overlay on your website using the Hosted Payment Page and jQuery libraries:

```
<html>
  <head>
    <!-- Load the jQuery library -->
    <script src="https://code.jquery.com/jquery-3.4.1.min.js" integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo=" crossorigin="anonymous"></script>

    <!-- Load the Hosted Payment Page library -->
    <script src="https://gateway.example.com/sdk/web/v1/js/hostedforms.min.js"></script>
  </head>
  <body>
      <!--
        Hosted Payment <form> as created by the Gateway Integration Library hostedRequest() method
        with addition of 'data-hostedforms-modal' attribute to signify a modal form is required.
      -->
      <form name="payment-form" method="post" action="https://gateway.example.com/hosted/" data-hostedforms-modal>
        <input type="hidden" name="merchantID" value="100001" />
        <input type="hidden" name="action" value="SALE" />
        <input type="hidden" name="type" value="1" />
        <input type="hidden" name="currencyCode" value="826" />
        <input type="hidden" name="countryCode" value="826" />
        <input type="hidden" name="amount" value="1001" />
        <input type="hidden" name="orderRef" value="Test purchase" />
        <input type="hidden" name="redirectURL" value="https://www.merchant.com/payment/" />
        <input type="hidden" name="signature" value="07599ef4cdb2e26cb2bf34a9c65190a7ce82494bc1df144c3bb0d20ee2655d8278dc663b2b0421ef12b8f081e821151bb4c644277c5d65b5523a96539b53b5aa" />
        <input type="submit" value="Pay Now">
      </form>
      <script>
          // Create a new Hosted Form object which will cause the above <form> to load into a modal
          // overlay over this page.
          var form = $(document.forms[0]).hostedForm();
      </script>
  </body>
</html>
```

A-23.2.3 Hosted Sale Transaction #2

The following example code shows how to create a payment form to open the Hosted Payment Page in a lightbox style overlay on your website using the Hosted Payment Page library:

```
<html>
  <head>
    <!-- Load the Hosted Payment Page library -->
    <script src="https://gateway.example.com/sdk/web/v1/js/hostedforms.min.js"></script>
  </head>
  <body>
    <!— Pay button placeholder -->
    <div id="paynow"></div>
    <script>
        // Create a new Hosted Form object which will render a payment button which will load
        // the Hosted Payment Pageo load into a modal overlay over this page.

        // The request can be provided from your server.
        var req = {
          merchantID: '100001',
          action: 'SALE',
          type: '1',
          currencyCode: '826',
          countryCode: '826',
          amount: '1001',
          orderRef: 'Test purchase',
          redirectURL: 'https://www.merchant.com/payment/',
          signature: '07599ef4cdb2e26cb2bf34a9c65190a7ce82494bc1df144c3bb0d20ee2655d8278dc663b2b0421ef12b8f081e821151bb4c644277c5d65b5523a96539b53b5aa',
        };

        var data = {
          id: 'my-payment-form',
          url: 'https://gateway.example.com/hosted/',
          modal: true,
          data: req,
          submit: {
            type: 'button',
            label: 'Pay <i>Now</i>'
          }
        };

        var form = new window.hostedForms.classes.Form('paynow', data);
    </script>
  </body>
</html>
```

blink

A-23.2.4 Hosted Sale Transaction #2 (jQuery)

The following example code shows how to create a payment form to open the Hosted Payment Page in a lightbox style overlay on your website using the Hosted Payment Page and jQuery libraries:

```
<html>
  <head>
    <!-- Load the jQuery library -->
    <script src="https://code.jquery.com/jquery-3.4.1.min.js" integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo=" crossorigin="anonymous"></script>

    <!-- Load the Hosted Payment Page library -->
    <script src="https://gateway.example.com/sdk/web/v1/js/hostedforms.min.js"></script>
  </head>
  <body>
    <!— Pay button placeholder -->
    <div id="paynow"></div>
    <script>
        // Create a new Hosted Form object which will render a payment button which will load
        // the Hosted Payment Pageo load into a modal overlay over this page.

        // The request can be provided from your server.
        var req = {
          merchantID: '100001',
          action: 'SALE',
          type: '1',
          currencyCode: '826',
          countryCode: '826',
          amount: '1001',
          orderRef: 'Test purchase',
          redirectURL: 'https://www.merchant.com/payment/',
          signature: '07599ef4cdb2e26cb2bf34a9c65190a7ce82494bc1df144c3bb0d20ee2655d8278dc663b2b0421ef12b8f081e821151bb4c644277c5d65b5523a96539b53b5aa',
        };

        var data = {
          id: 'my-payment-form',
          url: 'https://gateway.example.com/hosted/',
          modal: true,
          data: req,
          submit: {
            type: 'button',
            label: 'Pay <i>Now</i>'
          }
        };

        var form = $('#paynow').hostedForm(data);
    </script>
  </body>
</html>
```

blink

A-23.3 Hosted Payment Fields Library

The following example code shows how to create and manage Hosted Payment Fields using the Hosted Payment Field library.

The example shows how to style fields using an inline stylesheet and how to listen and react to the field's events.

The example also shows how to set up the payment form both automatically and manually and integrate with the jQuery validator plugin. You should choose the set up method best suited for your needs and whatever validation plugin or functions you are familiar with.

*Note: The example code demonstrates including the static transaction information, such as the **merchantID** and **amount**, in hidden form fields and POSTing the form directly to the Gateway's Direct Integration using partial message signing. We would however recommend that you capture just the information you require and then POST this data to your own website where you can use it to build a new fully signed request to send to the Gateway's Direct Integration as a server to server request.*

```
<html>
  <head>
    <!-- Load the jQuery library -->
    <script src="https://code.jquery.com/jquery-3.4.1.min.js" integrity="sha256-
CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo=" crossorigin="anonymous"></script>

    <!-- Load the jQuery Validator plugin -->
    <script src="https://cdn.jsdelivr.net/npm/jquery-validation@1.19.1/dist/jquery.validate.min.js"></script>

    <!-- Load the Hosted Payment Field library -->
    <script src="https://gateway.example.com/sdk/web/v1/js/hostedfields.min.js"></script>

    <!-- General styles -->
    <style>
      body {
        font-size: 14px;
      }

      .form-group {
        margin: 4px 0 15px 0;
      }

      .form-group LABEL {
        display: inline-block;
        max-width: 100%;
        margin-bottom: 5px;
        font-weight: bold;
      }

      .form-control {
        display: block;
        box-sizing: border-box;
        height: 34px;
        width: 400px;
        padding: 6px 12px;
        font-size: 14px;
        color: #555;
        background-color: #fff;
        background-image: none;
        border: 1px solid #ccc;
        border-radius: 4px;
        -webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, .075);
```

```
        box-shadow: inset 0 1px 1px rgba(0, 0, 0, .075);
        -webkit-transition: border-color ease-in-out .15s, -webkit-box-shadow ease-in-out .15s;
        -o-transition: border-color ease-in-out .15s, box-shadow ease-in-out .15s;
        transition: border-color ease-in-out .15s, box-shadow ease-in-out .15s;
      }

      .form-control.hf-focus {
        border-color: #66afe9;
        outline: 0;
        -webkit-box-shadow: inset 0 1px 1px rgba(0,0,0,.075), 0 0 8px rgba(102,175,233,.6);
        box-shadow: inset 0 1px 1px rgba(0,0,0,.075), 0 0 8px rgba(102,175,233,.6);
      }

      .has-error .form-control.hf-focus {
        border-color: #843534;
        -webkit-box-shadow: inset 0 1px 1px rgba(0,0,0,.075), 0 0 6px #ce8483;
        box-shadow: inset 0 1px 1px rgba(0,0,0,.075), 0 0 6px #ce8483;
      }
    </style>

    <!-- Hosted Field internal styles -->
    <style class="hostedfield">
      /* Grey out when disabled */
      .hostedfield:disabled {
        cursor: not-allowed;
        background-color: #eee;
        opacity: 1;
      }

      /* Change border and text to green when valid */
      .form-control:valid,
      .hostedfield:valid {
        border-color: #28a745 !important;
        color: #28a745 !important;
      }

      /* Change border and text to red when invalid */
      .form-control:invalid,
      .hostedfield:invalid {
        border-color: #a94442 !important;
        color: #a94442 !important;
      }

      /* Change text to light grey when readonly */
      .form-control:readonly,
      .hostedfield:readonly {
        color: lightgrey !important;
      }

      /* Emulate webkit auto fill style */
      .form-control.hf-autofill,
      .hostedfield.hf-autofill {
        background-color: rgb(250, 255, 189) !important;
        background-image: none !important;
        color: rgb(0, 0, 0) !important;
      }

      /* Add light blue placeholder */
      .form-control::placeholder,
      .hostedfield::placeholder {
        color: lightblue;
      }

      /* Show hovering over the control */
      .form-control:hover,
      .hostedfield:hover {
        font-style: italic;
```

```
109.      }
110.
111.      /* Style by id (hosted field will have '-hostedfield' appended to the id) */
112.      #form-card-number, #form-card-number-hostedfield {
113.        color: darkcyan;
114.      }
115.
116.    </style>
117.
118.    <!-- Hosted Field card-number internal styles -->
119.    <style class="card-number">
120.
121.      .hostedfield::placeholder {
122.        color: orange;
123.      }
124.
125.    </style>
126.  </head>
127.
128.  <body>
129.    <!-- tokenise payment data and send directly to the Gateway -->
130.    <form id="form" method="POST" novalidate="novalidate" lang="en"
131.      action="https://gateway.example.com/direct/"
132.      data-hostedform-tokenize='{"#form-customer-name": "customerName"}'>
133.      <input type="hidden" name="merchantID" value="100001">
134.      <input type="hidden" name="action" value="SALE">
135.      <input type="hidden" name="type" value="1">
136.      <input type="hidden" name="countryCode" value="826">
137.      <input type="hidden" name="currencyCode" value="826">
138.      <input type="hidden" name="amount" value="1001">
139.      <input type="hidden" name="orderRef" value="Test purchase">
140.      <input type="hidden" name="transactionUnique" value="1234">
141.      <input type="hidden" name="redirectURL" value="https://www.merchant.com/payment/">
142.      <input type="hidden" name="signature" value="5a0dd6fed71ef68bb3f20175b6a04bbd9d1c904d32ae3f160bd3b8f55740
    207e5d1e8de5e7e9960b136407e7454b82e428b8378003aa0146df3efa91a3e61b17|merchantID,action,type,countryCode,currenc
    yCode,amount,orderRef,transactionUnique,redirectURL">
143.      <input type="hidden" name="paymentToken" value="">
144.
145.      <div class="form-group">
146.        <label for="form-customer-name">Name on card:</label>
147.        <input id="form-customer-name" type="text" name="paymentToken[customerName]" autocomplete="cc-
    name" class="form-control form-control-native hostedfield-tokenise" placeholder="Firstname Surname" required>
148.      </div>
149.
150.      <div class="form-group">
151.        <label for="form-card-number">Card Number:</label>
152.        <input id="form-card-number" type="hostedfield:cardNumber" name="card-number" autocomplete="cc-
    number" class="form-control form-control-
    hosted" style="background: #f2f8fb;" placeholder="**** **** **** ****" required>
153.      </div>
154.
155.      <div class="form-group">
156.        <label for="form-card-expiry-date">Card Expiry Date:</label>
157.        <input id="form-card-expiry-date" type="hostedfield:cardExpiryDate" name="card-expiry-
    date" autocomplete="cc-exp" class="form-control form-control-hosted" required>
158.      </div>
159.
160.      <div class="form-group">
161.        <label for="form-card-start-date">Card Issue Date:</label>
162.        <input id="form-card-start-date" type="hostedfield:cardStartDate" name="card-start-
    date" autocomplete="cc-iss" class="form-control form-control-hosted" data-hostedfield='{"dropdown":true}' data-
    hostedfield-format="N - m | y" data-hostedfield-min-date="-40" data-hostedfeld-max-date="0">
163.      </div>
164.
165.      <div class="form-group">
166.        <label for="form-card-cvv">CVV:</label>
```

```
        <input id="form-card-cvv" type="hostedfield:cardCVV" name="card-cvv" autocomplete="cc-csc" class="form-control form-control-hosted" required>
      </div>

      <button id="form-submit" type="submit">Pay <span>▷</span></button>
    </form>

    <script>
      // This example demonstrates both automatic and manual form setup
      var automatic_setup = true;

      $(document).ready(function () {

        var $form = $('#form');

        // Listen for events on the form to see those sent from the Hosted Payment Fields
        // (For demonstration purposes only)
        $form.on(events);

        if (automatic_setup) {
          ////////////////////////////////////////////////////////////////
          // FORM AUTOMATIC SETUP
          ////////////////////////////////////////////////////////////////

          var opts = {
            // Auto setup the form creating all hosted fields (default)
            autoSetup: true,

            // Auto validate, tokenise and submit the form (default)
            autoSubmit: true,

            // Optional field configuration (by type)
            fields: {
              any: {
                nativeEvents: true,
              },
              cardNumber: {
                selector: $('#form-card-number'),
                style: 'text-decoration: green wavy underline;',
                stylesheet: $('style.hostedfields, style.card-number')
              }
            }
          };

          try {
            // Create form, automatically creating all child Hosted Payment Fields
            $form.hostedForm(opts);
          } catch(e) {
            showError('Failed to create hosted form: ' + e);
            throw e; // Can't continue with this script
          }

          // Listen for some events from the form thrown by the auto methods
          $form.on({
            // Let jQuery Validator check the form on submission
            'hostedform:presubmit': function (event) {
              console.log('Form submitting');
              return $form.valid();
            },

            // Show form is valid
            'hostedform:valid': function (event) {
              console.log('Form valid');
              return true;
            },

            // Show any validation errors
```

```
233.            'hostedform:invalid': function (event, details) {
234.              console.log('Form invalid');
235.              showFieldErrors(details.invalid);
236.              return true;
237.            },
238.
239.            // Show general error
240.            'hostedform:error': function (event, details) {
241.              showError(details.message);
242.              return true;
243.            }
244.          });
245.
246.          // Use jQuery validator to validate the form
247.          $form.validate();
248.
249.          // End of form automatic setup
250.
251.        } else {
252.          ////////////////////////////////////////////////////////////////
253.          // FORM MANUAL SETUP
254.          ////////////////////////////////////////////////////////////////
255.
256.          try {
257.            // Create the card number field with custom options
258.            $('#form-card-number').hostedField({
259.              nativeEvents: true,
260.              style: 'text-decoration: green wavy underline;',
261.              stylesheet: $('style.hostedfields, style.card-number')
262.            });
263.
264.            // Create the remaining hosted fields
265.            $('.form-control-hosted:input', $form).hostedField({nativeEvents: true});
266.
267.          } catch (e) {
268.            showError('Failed to create hosted fields: ' + e);
269.            throw e; // Can't continue with this script
270.          }
271.
272.          $form.validate({
273.            // Get the hosted form widget for the submitted form (Form1 only)
274.            submitHandler: function () {
275.              try {
276.                console.log('getPaymentToken');
277.
278.                // Check we have some enabled fields to submit
279.                if ($($form[0].elements).filter(':enabled:not([type="hidden"])').length === 0) {
280.                  showError('You must enable some fields');
281.                  return false;
282.                }
283.
284.                var hostedform = $form.hostedForm('instance');
285.
286.                var also = {
287.                  customerName: $('#form-customer-name').val()
288.                };
289.
290.                hostedform.getPaymentDetails(also, true).then(
291.
292.                  // Success validating the form and requesting a payment token
293.                  function (details) {
294.                    if (details.success) {
295.                      $form[0].elements['paymentToken'].value = details.paymentToken;
296.                      $form[0].submit();
297.                    } else if (details.invalid) {
298.                      $form.valid();
299.                      showFieldErrors(details.invalid);
```

```
                    } else {
                      showError('There was a problem fetching the payment token. Please seek assistance.');
                    }
                  },

                  // Failure either validating the form or requesting the payment details
                  function (e) {
                    showError('There was a problem fetching the payment token. Please seek assistance.');
                  }
                );

              } catch (e) {
                showError('There was a problem fetching the payment token. Please seek assistance.');
              }
            }
          });

          // End of form manual setup

        }

        // Hide errors once all fields are valid
        $('#form :input').on('valid', function () {
          if ($(this.form).find(':invalid').length === 0) {
            hideError($(this.form));
          }
        })

        // Listen for some events on the none Hosted Fields
        $('.form-control-native').on('invalid', bsMarkInvalid);
        $('.form-control-native').on('valid', bsMarkValid);

        // Check we can see the Hosted Fields via their new class
        // (For demonstration purposes only)
        console.log($('.form-control-hosted.hostedfield-element'));

        // Check we can see the Hosted Fields via the psuedo element
        // (For demonstration purposes only)
        console.log($('.form-control:hostedfield'));

      });

      ////////////////////////////////////////////////////////////////////
      // Supporting functions
      ////////////////////////////////////////////////////////////////////

      // Display events that are passed from hosted field
      var events = {
        'hostedfield:create.example'                          : showEvent,
        'hostedfield:destroy.example'                         : showEvent,
        'hostedfield:ready.example'                           : showEvent,
        'hostedfield:style.example'                           : showEvent,
        'hostedfield:placeholder.example'                     : showEvent,
        'hostedfield:invalid.example invalid.example'         : showEvent,
        'hostedfield:userinvalid.example userinvalid.example' : showEvent,
        'hostedfield:valid.example valid.example'             : showEvent,
        'hostedfield:uservalid.example uservalid.example'     : showEvent,
        'hostedfield:disabled.example disabled.example'       : showEvent,
        'hostedfield:enabled.example enabled.example'         : showEvent,
        'hostedfield:required.example required.example'       : showEvent,
        'hostedfield:optional.example optional.example'       : showEvent,
        'hostedfield:readonly.example readonly.example'       : showEvent,
        'hostedfield:readwrite.example readwrite.example'     : showEvent,
        'hostedfield:focus.example focus.example'             : showEvent,
        'hostedfield:blur.example blur.example'               : showEvent,
        'hostedfield:mouseenter.example mouseenter.example'   : showEvent,
        'hostedfield:mouseleave.example mouseleave.example'   : showEvent,
```

```
367.        'hostedfield:mouseover.example mouseover.example'      : showEvent,
368.        'hostedfield:mouseout.example mouseout.example'        : showEvent,
369.        'hostedfield:mousemove.example mousemove.example'      : showEvent,
370.        'hostedfield:keydown.example keydown.example'          : showEvent,
371.        'hostedfield:keypress.example keypress.example'        : showEvent,
372.        'hostedfield:keyup.example keyup.example'              : showEvent,
373.        'hostedfield:change.example change.example'            : showEvent,
374.        'hostedfield:input.example input.example'              : showEvent,
375.
376.        'hostedfield:invalid.example invalid.example'          : bsMarkInvalid,
377.        'hostedfield:valid.example valid.example'              : bsMarkValid,
378.        'hostedfield:valid.example valid.example'              : hideError,
379.      };
380.
381.      function isInvalid(element) {
382.        return !element[0].checkValidity();
383.      }
384.
385.      function showError(msg) {
386.        $('#error-info').html(msg).show();
387.      }
388.
389.      function hideError($form, msg) {
390.        $('#error-info', $form).hide();
391.      }
392.
393.      function showFieldErrors(errors) {
394.        var msg = '<h5>Error</h5><p>The following fields are invalid:</p><ul>';
395.        for (var p in errors) {
396.          msg += '<li><b>' + p + ':</b> ' + errors[p] + '</li>';
397.        }
398.        msg += '</ul>'
399.        showError(msg);
400.      }
401.
402.      function bsMarkInvalid(e) {
403.        var element = (e instanceof $.Event ? this : e);
404.        $(element).closest('.form-group').addClass('has-error');
405.      }
406.
407.      function bsMarkValid(e) {
408.        var element = (e instanceof $.Event ? this : e);
409.        $(element).closest('.form-group').removeClass('has-error');
410.      }
411.
412.      function showEvent(event) {
413.        console.log(event);
414.        console.log('Field ' + event.type + ' event: ', this, arguments);
415.      }
416.
417.      jQuery.validator.setDefaults({
418.        ignore: [],
419.        rules: {
420.          'customer-name': {
421.            checkValidity: true,
422.            required: false
423.          },
424.          'card-details': {
425.            checkValidity: true,
426.            required: false
427.          },
428.          'card-number': {
429.            checkValidity: true,
430.            required: false
431.          },
432.          'card-expiry-date': {
433.            checkValidity: true,
```

```
            required: false
          },
          'card-start-date': {
            checkValidity: true,
            required: false
          },
          'card-issue-number': {
            checkValidity: true,
            required: false
          },
          'card-cvv': {
            checkValidity: true,
            required: false
          }
        },
        keyup: null, // Don\'t validate on keyup
        showErrors: function (errorMap, errorList) {
          if (errorList && errorList.length) {
            var errors = {};
            for (var i = 0, max_i = errorList.length; i < max_i; i++ ) {
              var label = $('label[for="' + errorList[i].element.id + '"]:not(".error")').text();
              errors[label] = errorList[i].message;
            }
            showFieldErrors(errors);
          }
          this.defaultShowErrors(errorMap, errorList);
        },
        highlight: bsMarkInvalid,
        unhighlight: bsMarkValid,
        errorPlacement: function (error, element) {
          $(element).closest('.form-control:not(".hostedfield-element")').after(error);
        }
      });

      $.validator.addMethod('checkValidity',
        function (value, element, params, message) {
          element.checkValidity();
          var valid = (element.validationMessage === '');
          $(element).attr('aria-invalid', !valid);
          return valid;
        },
        function (params, element) {
          return element.validationMessage;
        }
      );

    </script>

  </body>
</html>
```

blink

A-24 Frequently Asked Questions

1. I'm getting Invalid Credentials. What do I do?

Check your Merchant ID in your integration is correct. Our Gateway Merchant IDs typically begin with 1 and are currently 6 digits long, e.g. 100001.

2. I'm getting an invalid signature error message. How do I fix it?

Check that you are using the correct method for calculating the signature and the correct secret signature key for the Merchant Account used.

Make sure that you are not using an image form submit button because that will add fields to the post that cannot be removed and will render the signature useless.

Refer to appendix A-13 for a step by step guide to creating a signature. If you use the same values as in the example, you can check if your signature generation routine produces the same results.

This test step by step generator is available from the Gateway by adding the following to your Gateway URL:

/devtools/sigtest.php

3. I have more than one Merchant ID - how do I use more than one?

You have a couple of options here. You can set up separate integrations for each MID, which can be a bit inconvenient. Your other option is to request they are connected together. Please contact our support team to get your MIDs connected and you will then only need to use one.

4. I receive a 'Bad Testcard Usage' error message. Why?

If you receive this error message, you are using test cards on a live Merchant ID. Please only use live cards on live Merchant IDs. Our test cards will only work on the test Merchant ID provided when you sign up with us.

blink

INDEX