Payment Provider Integration Documentation

Table of Contents

1. Creating Transactions
2. Customer Browser Redirect
3. Funding Sources
4. Receive a Payment
5. Payouts
6. Webhooks

Creating Transactions

The first step for executing both Accepting Incoming Payments and Outgoing Payouts is to create a transaction in the Payment Gateway by using the `preparePost` method. In this step, you will have to provide basic information about the transaction you wish to process and will be given 2 unique identifiers which will be used for the rest of the transaction process.

Creating a transaction requires the use of an API Key and returns sensitive information which should never be accessible by the customer. We recommend that you make such API calls server-side. This ensures that such information remains internal to your systems and is never accessible in a browser.r.

HTTPS Post URL:

`https://example.com/preparePost`

Parameters:

parameter name	type	Description
firstName	string	customer first name
LastName	string	customer last Name
clinetId	string	Unique ID to identify the customer by
foundingSourceName	string	Transaction method, for more details, see Funding Sources
amount	float	Transaction amount
currency	string	Transaction Currency in which the amount is given in (USD, EUR, BTC)
returnUrl	string	target URL to which webhook events with the transaction result will be send
adminLink	string	[Optional] Link to customer profile for admin use
emailAddress	string	[Optional] customer email address

Example

curl

```
curl -X POST 'https://example.com/preparePost' \
-H 'Authorization: Bearer <Your API Key>' \
-H 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'firstName=John' \
--data-urlencode 'lastName=Smith' \
--data-urlencode 'foundingSourceName=BTC1' \
--data-urlencode 'currency=USD' \
--data-urlencode 'amount=100' \
--data-urlencode 'returnUrl=https://example.com/webhook' \
--data-urlencode 'clientId=823320'
```

php:

```
<?php
$curl = curl_init();
curl_setopt_array($curl, array(
CURLOPT_URL => 'https://example.com/preparePost',
CURLOPT_RETURNTRANSFER => true,
CURLOPT_ENCODING => '',
CURLOPT_MAXREDIRS => 10,
CURLOPT_TIMEOUT => 0,
CURLOPT_FOLLOWLOCATION => true,
CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
CURLOPT_CUSTOMREQUEST => 'POST',
CURLOPT_POSTFIELDS =>
'firstName=John&lastName=Smith&foundingSourceName=BTC1&cu
rrency=
USD&amount=100&returnUrl=https%3A%2F%2Fexample.com
%2Fwebhook
&clientId=823320',
CURLOPT_HTTPHEADER => array(
'Authorization: Bearer <Your API Key>',
'Content-Type: application/x-www-form-urlencoded'
),
));
$response = curl_exec($curl);
curl_close($curl);
echo $response;
```

Response:

```
{
    "mcTxId":  "c9e5dd42047e4c7a858be92124b06dac",
    "secretId":  "62e0265915dd4dbaad83ad0b373c8862"
}
```

The API returns a unique charge identifier `mcTxID`, along with a unique secret identifier `secretId`. You should store the secret identifier persistently to match and validate Webhook events against the corresponding transaction in your system.

The `mcTxID` will be the primary identifier from this point forwards and will be known to the customer. The `secretId` must only be known to your internal systems and the payment gateway as it is used to send confirmation of successful payment to you once the transaction has been completed in the payment gateway

---

Customer Browser Redirect

After creating a transaction, you will need to redirect customers to a hosted page where they can complete their payment. Customers should be redirected to a hosted page using a URL with format:

`https://example.com/<SOURCE_PLACEHOLDER>/Start?mcTxId=<ID_PLACEHOLDER>`

You need to replace the `SOURCE_PLACEHOLDER` with the `foudingSourceName` used to create the transaction and the `ID_PLACEHOLDER` with the unique identifier `mcTxId` returned in the response. Upon visiting the URL, your customer is presented with a page that displays all the information he needs to complete payment.

Hosted page URLs are unique and not re-usable. You should issue a new URL for every Transaction.

Example hosted URL: `https://example.com/BTC1/Start?mcTxId=c9e5dd42047e4c7a858be92124b06dac`

Funding Sources

Each payment or payout requires a funding source. This deciedes if a transaction is a Incoming payment or an outgoing payout and by which method it will be executred. Here's a list of supported funding sources:

- BTC1: Customer Incoming Payment via direct transfer.
- BTC2: Customer Outgoing Payout via direct transfer.
- BTC3: Customer Incoming Payment via Instacoins.

Receive a Payment

Unlike credit cards, where merchants must obtain payment credentials in order to charge a customer, cryptocurrencies are more like digital cash and rely on the customer explicitly sending money to the merchant.

When creating a transaction, make sure to use either `BTC1` or `BTC3` as the Funding Source to receive an Incoming Payment.

Once a transaction is created, redirect the customer to a hosted page as described in Customer Browser Redirect

After creating a transaction, the Quick Payments Gateway will continuously monitor the activity on the generated payment address. An expiration time for the transaction is set, by default 1 hour after the creation date. If the customer does not make a payment within that time, the transaction is considered expired, but still accepted if the customer makes a delayed payment.

payment is received, your balance is credited, your dashboard automatically updated, and a Webhooks event is sent to your server, allowing you to automatically process the payment in your system. Webhook events will be sent to the `return_url` specified in your initial API call to `POST /preparePost` used to create the transaction . Every Webhook request will include the `secret_id` returned as part of the response when the transaction is created. This allows you to validate that the events were sent by Quick Payments Gateway and not by a third party.

Please note that the `amount` returned in the Webhooks event might differ from the amount provided when the transaction was created via the preparePost due to changing conversion rates or the customer sending a different amount. You will have to update the transaction in your system accordingly.

Payouts

When creating a Transaction, make sure to use `BTC2` as Funding Source in order to send a payout.

Once you created a Transaction, redirect the customer to a hosted page as described in Customer Browser Redirect where they will provide their desired recipient address to which the payout will be sent.

After the customer providedhis recipient address, the payout needs to be finalized by confirming it via the Quick Payments Gateway Dashboard

Confirm a Payout

To finalize a payout, you need confirm it by heading over to Quick Payments Gateway Dashboard. We set an expiration time for the payout which is currently 3 days after the creation date.

Upon confirming a payout, Quick Payments Gateway will first check if there is a sufficient balance to process the payout. In case of an insufficient balance, a payout can be confirmed at a later time. Quick Payments Gateway will then calculate the latest exchange rate on your behalf and finalize the payout.

Once the payout is finalized, a Webhook event is sent to your server, allowing you to automatically process the payout in your system. Webhook events will be sent to the `return_url` specified in your initial API call to `POST /preparePost` used to create a payout. Every Webhook event will include the `secret_id` returned as part of the response when a payout is created. This allows you to validate that the events were sent by Quick Payments Gateway and not by a third party.

Webhooks

A Webhook enables the Quick Payments Gateway to push real-time notifications to your app. The Quick Payments Gateway uses HTTPS to send these notifications to your app as form data. You can then use these notifications to execute actions in your backend systems.

The following is a list of event triggers that are available:

- approve
- deny
- cancel

On successful transactions, the Webhook action will be `approve`. `deny` and `cancel` function identically, the only difference being that `deny` is triggered by the Payment Gateway and `cancel` by the customer.

Please note that the `amount` returned for `approve` events for Incoming Payments might differ from the amount provided when the transaction was created via the preparePost due to changing conversion rates or the customer sending a different amount. You will have to update the transaction in your system accordingly.

Additionally, while this is not an intended way to use the Payment Gateway, it is possible for a customer to send BTC to an Address after the corresponding Transaction has already been canceled, denied, or even approved. This means that in rare cases, you might receive an `approve` Webhook event for a transaction that has previously been completed. There is no way for us to prevent the customer from doing this; however, the Payment Gateway is able to handle such cases.

In the case that a customer sends another BTC payment to the address of a previous Transaction, an additional parameter called `index` will be added to the Webhook event, which will indicate that you are dealing with a separate `approve` Event referring to an additional BTC payment on the same Address rather than a duplicate delivery attempt of the previous event.

This very specific case should only occur if your customers are using direct payments instead of payments via Instacoins and even then should be very rare, so you may choose to ignore this, as it is not essential to the main functionality of the Payment Gateway

Webhook Events and Payloads

Each Webhook event payload is sent as form data (`application/x-www-form-urlencoded`).

parameter name	type	Description
action	string	Type of Webhook event: approved, deny, cancel
secretId	string	The `secretId` returned in `preparePost` to identify the transaction
amount	float	The transaction amount in the currency provided in `preparePost`
btcamount	float	The transaction amount in BTC
trxhash	string	The BTC transaction hash
notes	string	Additional information about the transaction
index	integer	[Optional] The number of transactions received on a given address. Used to distinguish duplicate payments on the same address.

Example Payloads

Approve:

```
action: approve
secretId: 62e0265915dd4dbaad83ad0b373c8862
amount: 50
btcamount: 0.001158
btcaddress: 36vUDn2XXNJYSqZF3gGTMZom6FuUXTcvgS
trxhash: c63b26dfe32c5deb984d53711b6f4895ad9871d748de6d9e3e48cc8352e4bdc5
notes: c9e5dd42047e4c7a858be92124b06dac address:
36vUDn2XXNJYSqZF3gGTMZom6FuUXTcvgS amount: 0.001158
```

Deny:

```
action: deny
secretId: 62e0265915dd4dbaad83ad0b373c8862
amount: 50
notes: Transaction timed out
```

Cancel:

```
action: cancel
secretId: 62e0265915dd4dbaad83ad0b373c8862
amount: 50
notes: Transaction canceled by user
```

PHP Examples:

Approve:

```php
<?php
$curl = curl_init();
curl_setopt_array($curl, array(
  CURLOPT_URL => '<return URL>',
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => '',
  CURLOPT_MAXREDIRS => 10,
  CURLOPT_TIMEOUT => 0,
  CURLOPT_FOLLOWLOCATION => true,
  CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
  CURLOPT_CUSTOMREQUEST => 'POST',
  CURLOPT_POSTFIELDS =>
'action=approve&secretId=62e0265915dd4dbaad83ad0b373c8862&amount=50&btcamount=0.00
1158&btcaddress=36vUDn2XXNJYSqZF3gGTMZom6FuUXTcvgS&notes=c9e5dd42047e4c7a858be9212
4b06dac%20address%3A%2036vUDn2XXNJYSqZF3gGTMZom6FuUXTcvgS%20amount%3A%200.001158',
  CURLOPT_HTTPHEADER => array(
    'Content-Type: application/x-www-form-urlencoded'
  ),
));
$response = curl_exec($curl);
curl_close($curl);
echo $response;
```

Deny:

```php
<?php
$curl = curl_init();
curl_setopt_array($curl, array(
  CURLOPT_URL => '<return URL>',
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => '',
  CURLOPT_MAXREDIRS => 10,
  CURLOPT_TIMEOUT => 0,
  CURLOPT_FOLLOWLOCATION => true,
  CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
  CURLOPT_CUSTOMREQUEST => 'POST',
  CURLOPT_POSTFIELDS =>
'action=deny&secretId=62e0265915dd4dbaad83ad0b373c8862&amount=50&notes=Transaction
%20timed%20out',
  CURLOPT_HTTPHEADER => array(
    'Content-Type: application/x-www-form-urlencoded'
  ),
));
$response = curl_exec($curl);
curl_close($curl);
echo $response;
```

Cancel:

```php
<?php
$curl = curl_init();
curl_setopt_array($curl, array(
  CURLOPT_URL => '<return URL>',
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => '',
  CURLOPT_MAXREDIRS => 10,
  CURLOPT_TIMEOUT => 0,
  CURLOPT_FOLLOWLOCATION => true,
  CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
  CURLOPT_CUSTOMREQUEST => 'POST',
  CURLOPT_POSTFIELDS =>
'action=cancel&secretId=62e0265915dd4dbaad83ad0b373c8862&amount=50&notes=Transacti
on%20canceled%20by%20user',
  CURLOPT_HTTPHEADER => array(
    'Content-Type: application/x-www-form-urlencoded'
  ),
));
$response = curl_exec($curl);
curl_close($curl);
echo $response;
```

Handling Webhook Events

Webhook receiver endpoints are expected to quickly return a successful status code (`200`) as acknowledgement. When a webhook event is acknowledged it will not be sent again.

In case a webhook event gets a response status code other than `200` or no response at all, the Quick Payments Gateway makes use of exponential webhook retry policy.

The Quick Payments Gateway waits for **30 seconds** to receive a response. If the receiver endpoint fails to respond within this timeframe, then the webhook request is timed out.

The exponential retry policy attempts to deliver the webhook event **6** more times with an exponential back-off multiplier of 30 seconds.

If the webhook receiver fails to acknowledge the event after the last retry attempt, the Quick Payments Gateway stops retrying and marks the event with a failed delivery status.