Payment Gateway API

v0.5

BitPay, Inc.

https://bitpay.com

Table of Contents

Introduction

The Bitcoin Payment Gateway API is designed for merchants that need full control over their customers' shopping and checkout experience. An eCommerce site can make use of this API to transmit invoice information to BitPay.com from their back-end server, and receive server notifications when the customer has completed payment and the invoice total has been credited to the merchant account.

A merchant can elect to receive notifications immediately upon receipt of a payment, or when the payment has been completed and credited to the merchant account.

The following interactions with the BitPay.com service available via this API:

- Create an invoice
- Fetch an invoice status
- Receive invoice status updates
- Fetch Bitcoin Best Bid exchange rates
- Retrieve your merchant account ledger

Note: For the documentation on the older version of the API that uses SSL client certificates for authentication, see https://bitpay.com/downloads/bitpayApi-0.2.pdf (we have not yet updated all shopping cart plugins to use the new authentication method). While the old authentication method is now deprecated, it is still supported (we don't have any immediate plans to disable it if you're already setup to use it).

Activate API Access

The BitPay.com JSON API is accessible at https://bitpay.com/api/.

The merchant must obtain an API key from the bitpay website by logging into their merchant account and clicking on My Account, API Access keys. A merchant can create multiple keys for use with different e-commerce stores or API functions. Once an API key has been created, BitPay will use this API key to authenticate your API connections.

The merchant's API key must remain private and should never be visible on any client-facing code. Should it ever be compromised, the merchant can generate a new key in their BitPay account.

When connecting to BitPay, use HTTP Basic Authentication with the username as your API key and leave the password blank (the following page describes the HTTP Basic authentication protocol in detail: http://www.ietf.org/rfc/rfc2617.txt). You should also only communicate with the server if you can validate the bitpay.com SSL certificate with a certificate authority. Most HTTPS client libraries make this as simple as setting a switch. Similarly, inbound notification connections should only be recognized when the SSL certificate is validated. Taking both of these steps will ensure that you are always communicating with the Bitpay server and that your API key will never be exposed.

Bitcoin Payment Protocol

The BitPay server fully supports the Bitcoin Payment Protocol. See the specifications for this implementation:

- Payment Protocol (BIP70) - https://en.bitcoin.it/wiki/BIP_0070
- Payment Protocol MIME Types (BIP71) - https://en.bitcoin.it/wiki/BIP_0071
- URI Extensions for Payment Protocol (BIP72) - https://en.bitcoin.it/wiki/BIP_0072
- Use “Accept” header for response type negotiation with Payment Request URLs (BIP73) - https://en.bitcoin.it/wiki/BIP_0073

The Bitcoin Payment Protocol (as specified in BIPs 70 - 73) eliminates a lot of human error in making a bitcoin payment. A user can just click on a payment link, or scan a QR code, and the wallet software offers two simple choices to the user, pay or don’t pay. The user no longer has to copy the address and amount into their wallet.

Native Refund Address Support

With the Payment Protocol, the wallet supplies a refund address along with the payment. This eliminates another potential source of error in refund situations. This approach to refunds works on the block chain, with any wallet software, and does not require the buyers to have a BitPay account.

Secure, Signed Payment Requests

The payment protocol supports optional SSL signatures (technically X.509 signatures) on payment requests. This offers certainty to users that they are sending their payment to the intended recipient (all BitPay payment requests are signed). When using a wallet that supports the payment procotol (currently Bitcoin-QT and the Android Bitcoin Wallet), you will immediately notice that your wallet tells you that BitPay is requesting a payment.

User Friendly QR Codes (BIP 73)

BitPay also supports BIP-73, which considerably improves the usability of QR codes for Bitcoin payments. BIP-73 reduces the information required to be embedded in a payment request QR code, reducing their density. Less dense QR codes are easier to use in low light situations or from longer distances. These lower density QR codes are also normal HTTP URLs, offering an opportunity to provide additional information and instructions to users of devices that don’t already have a wallet installed. BitPay displays both the older, backward compatible QR code as well as the newer payment protocol QR code. You can toggle between these two QR codes by clicking or tapping on the QR code in our current invoice.

Direct Payment Communication

Perhaps the most exciting thing about the payment protocol is that it eliminates the need to use the mesh network for communicating a payment from sender to recipient.

The Bitcoin mesh network currently serves two purposes: communicating payments from sender to recipient and communicating payments from originator to miners. By communicating payments directly from sender to recipient, the mesh network can be used exclusively for communicating payments from originator to miners. The network is then free to propagate or ignore transactions without adversely affecting the communications between sender and recipient. This allows for the emergence of a true market in transaction fees. And by reducing the load on the mesh network to just those transactions which are profitable for miners, it improves

Bitcoin's scalability.

BitPay Implementation

The BitPay server supports both BIP-72 and BIP-73 compatible URIs.

BIP 72 (old style) compatible URI

The BIP 72 URI scheme begins with the "bitcoin:" protocol and is followed by the bitcoin address to which the payment is sent, the amount to send, and a parameter ('r') that encodes the complete BIP 73 payment URI.

Wallets that scan this protocol and which are not BIP 73 enabled will simply ignore the extra 'r' parameter. This results in the wallet receiving only the bitcoin address and the payment amount.

```
bitcoin:19aDWT1BroEdoJHJfmFz5P43mDCs5EFmqz?amount=0.0016&amp;r=https%3A%2F%2Fbitpay.com%2Fi%2FBK3rjpRjXcA1wxthbCTs1M
```

BIP 73 (new style) compatible URI

The BIP 73 URI scheme is arbitrary and defined by the server implementation. When a wallet scans a BIP 73 URI it won't immediately be able to use the URI information to send a payment to the network. The wallet must first send a request using the scanned URI with additional information in the header of the HTTP request (see the Bitcoin Payment Protocol specifications, BIP 70 to 73). The wallet will expect the response to be usable to send a payment to the Bitcoin network.

The BitPay payment URI for the Bitcoin Payment Protocol is simply the URI of the BitPay invoice.

```
https://bitpay.com/i/BK3rjpRjXcA1wxthbCTs1M
```

Invoice States

A BitPay.com invoice can be in one of the following states: “new”, “paid”, “confirmed”, “complete”, “expired” or “invalid”. Payments sent to the bitcoin address associated with an invoice will only be credited to the invoice when it is in the “new” state.

State	Description
“new”	An invoice starts in this state. When in this state and only in this state, payments to the associated bitcoin address are credited to the invoice. If an invoice has received a partial payment, it will still reflect a status of new to the merchant (from a merchant system perspective, an invoice is either paid or not paid, partia payments and over payments are handled by bitpay.com by either refunding the customer or applying the funds to a new invoice.
“paid”	As soon as full payment (or over payment) is received, an invoice goes into the paid status.
“confirmed”	The transaction speed preference of an invoice determines when an invoice is confirmed. For the high speed setting, it will be confirmed as soon as full payment is received on the bitcoin network (note, the invoice will go from a statu of new to confirmed, bypassing the paid status). For the medium speed setting, the invoice is confirmed after the payment transaction(s) have been confirmed by 1 block on the bitcoin network. For the low speed setting, 6 blocks on the bitcoin network are required. Invoices are considered complete after 6 blocks on the bitcoin network, therefore an invoice will go from a paid status directly to a complete status if the transaction speed is set to low.
“complete”	When an invoice is complete, it means that BitPay.com has credited the merchant’s account for the invoice. Currently, 6 confirmation blocks on the bitcoin network are required for an invoice to be complete. Note, in the future (fo qualified payers), invoices may move to a complete status immediately upon payment, in which case the invoice will move directly from a new status to a complete status.
“expired”	An expired invoice is one where payment was not received and the 15 minute payment window has elapsed.
“invalid”	An invoice is considered invalid when it was paid, but payment was not confirme within 1 hour after receipt. It is possible that some transactions on the bitcoin network can take longer than 1 hour to be included in a block. In such circumstances, once payment is confirmed, BitPay.com will make arrangements with the merchant regarding the funds (which can either be credited to the merchant account on another invoice, or returned to the buyer).

Create an Invoice

An invoice is created by sending an http POST message to https://bitpay.com/api/invoice with the details of the invoice passed in the body of the request. The body of the message must be JSON encoded and the content-type should be set to “application/json”.

On successful creation, the invoice details will be provided in a JSON encoded response. If there is an error, you will receive a JSON encoded error response. All error responses will have an “error” field that is an object with two fields called “type” and “message”. A merchant is restricted to creating no more than 100 invoices per hour (there are also per second and per minute limits). The fields in the request are described below:

Required POST fields

Name	Description
“price”	This is the amount that is required to be collected from the buyer. Note, if this is specified in a currency other than BTC, the price will be converted into BTC at market exchange rates to determine the amount collected from the buyer.
"currency”	This is the currency code set for the price setting. The pricing currencies currently supported are USD, EUR, BTC, and all of the codes listed on this page https://bitpay.com/bitcoin-exchange-rates

Optional Payment Notification (IPN) fields

Name	Description
“posData”	A passthru variable provided by the merchant and designed to be used by the merchant to correlate the invoice with an order or other object in their system. Maximum string length is 100 characters. This passthru variable can be a JSON-encoded string, for example posData: ‘ { “ref” : 711454, “affiliate” : “spring112” } ‘
“notificationURL”	A URL to send status update messages to your server (this must be an https URL, unencrypted http URLs or any other type of URL is not supported). Bitpay.com will send a POST request with a JSON encoding of the invoice to this URL when the invoice status changes.
“transactionSpeed”	default value: set in your https://bitpay.com/order-settings, the default value set i your merchant dashboard is “medium”. ● “high”: An invoice is considered to be "confirmed" immediately upon receipt of payment. ● “medium”: An invoice is considered to be "confirmed" after 1 block

	confirmation (~10 minutes). ● "low": An invoice is considered to be "confirmed" after 6 block confirmations (~1 hour). **NOTE:** Orders are posted to your Account Summary after 6 block confirmations regardless of this setting.
"fullNotifications"	default value: false ● true: Notifications will be sent on every status change. ● false: Notifications are only sent when an invoice is confirmed (accordin to the "transactionSpeed" setting).
"notificationEmail"	Bitpay.com will send an email to this email address when the invoice status changes.

Optional Order Handling fields

Name	Description
"redirectURL"	This is the URL for a return link that is displayed on the receipt, to return the shopper back to your website after a successful purchase. This could be a page specific to the order, or to their account.

Optional Buyer Information to display

Name	Description
"orderID"	Used to display your public order number to the buyer on the BitPay invoice. In the merchant Account Summary page, this value is used to identify the ledger entry. Maximum string length is 100 characters.
"itemDesc"	Used to display an item description to the buyer. Maximum string length is 100 characters.
"itemCode"	Used to display an item SKU code or part number to the buyer. Maximum string length is 100 characters.
"physical"	default value: false ● true: Indicates a physical item will be shipped (or picked up) ● false: Indicates that nothing is to be shipped for this order
"buyerName" "buyerAddress1" "buyerAddress2"	These fields are used for display purposes only and will be shown on the invoice if provided. Maximum string length of each field is 100 characters.

“buyerCity” “buyerState” “buyerZip” “buyerCountry” “buyerEmail” “buyerPhone”	

BitPay Server Response

The response to a create invoice request, the response to a get invoice request, and the content of a status update notification are all identical JSON representations of the invoice object. The fields are described below:

Name	Description
“id”	The unique id of the invoice assigned by bitpay.com
“url”	An https URL where the invoice can be viewed.
“posData”	The passthru variable provided by the merchant on the original invoice creation.
“status”	The current invoice status. The possible states are described earlier in this document. ● “new” ● “paid” ● “confirmed” ● “complete” ● “expired” ● “invalid”
“price”	The price set by the merchant (in terms of the provided currency).
“currency”	The 3 letter currency code in which the invoice was priced.
“btcPrice”	The amount of bitcoins being requested for payment of this invoice (same as the price if the merchant set the price in BTC).
“invoiceTime”	The time the invoice was created in milliseconds since midnight January 1, 1970. Time format is “2014-01-01T19:01:01.123Z”.
“expirationTime”	The time at which the invoice expires and no further payment will be accepted (i milliseconds since midnight January 1, 1970). Currently, all invoices are valid fo 15 minutes. Time format is “2014-01-01T19:01:01.123Z”.
“currentTime”	The current time on the BitPay.com system (by subtracting the current time from the expiration time, the amount of time remaining for payment can be determined). Time format is “2014-01-01T19:01:01.123Z”.

Get Invoice Status

To get the current state of an invoice, an http GET request can be sent to https://bitpay.com/api/invoice/<id>, where the <id> is the invoice id provided when the invoice was created. The format of the response is exactly the same as that which is returned when creating an invoice.

Receive Invoice Status Updates

Invoice status updates can be sent either via email, https or both. The “notificationEmail” and “notificationURL” settings control the destination for the notification. Note, email notification is a human readable format and not intended for use as a system interface. For https notification, BitPay.com sends a POST request to the given URL with a JSON encoding of the invoice that is identical to the format returned from a create invoice or get invoice request. If “fullNotifications” are set to true, then an update will be sent for every change in status. If “fullNotifications” are false, then an update is only sent when an invoice is confirmed (according to the “transactionSpeed” setting).

Get Bitcoin Best Bid (BBB) Rates

BitPay consolidates market depth from multiple exchanges to provide buyers with a Bitcoin Best Bid (BBB) exchange rate. BitPay currently calculates BBB based on Bitcoin/US Dollar rates because of the maximum liquidity. For a complete list of available currencies see https://bitpay.com/bitcoin-exchange-rates.

To calculate the exchange rate for US Dollars, we pull the market depth from exchanges with adequate liquidity and withdrawal capability in USA and the Eurozone. The exchange order books are merged into a Consolidated Level II table.

The BBB is calculated by simulating an auto-routing market sell order, across all exchanges, with zero commission fees. Buyers will always get a better value by spending their bitcoins at a BitPay merchant than by selling them on an exchange.

The BBB is available via JSON API at https://bitpay.com/api/rates. Rates are updated every 1 minute.

BitPay Server Response

The response to a request for BBB is a list of identical JSON representations of the rate object. The fields are described below:

Name	Description
“name”	The full display name of the currency.
“code”	The three letter code for the currency, in all caps.
“rate”	The numeric exchange rate of this currency provided by the BitPay server.

Get the Transaction Ledger

The bitcoin tranaction ledger is retreived by sending an http GET message to https://bitpay.com/api/ledger with the details of the ledger query passed in the request URL.

On successful retrieval, the invoice details will be provided in a JSON encoded response. If there is an error, you will receive an empty JSON array in the response; `[]`.

The parameters in the transaction ledger request are described below:

Required GET parameters

Name	Description
“c”	This is the three letter currency code set for the ledger. The payout currencies currently supported are: ● BTC - Bitcoin ● AUD - Australian Dollar ● CAD - Canadian Dollar ● EUR - Eurozone Euro ● GBP - Pound Sterling ● MXN - Mexican Peso ● NZD - New Zealand Dollar ● USD - US Dollar ● ZAR - South African Rand
"startDate”	The start date for the ledger query. Ledger entries are retrieved from this date (inclusive) forward. The format for this parameter is “yyyy-mm-dd”; example “2014-01-01”.
“endDate”	The end date for the ledger query. Ledger entries are retrieved up to and including this date. The format for this parameter is “yyyy-mm-dd”; example “2014-01-31”.

Example Ledger GET Request

```
https://bitpay.com/api/ledger?c=BTC&startDate=2014-01-01&endDate=2014-01-31
```

BitPay Server Response

The response to a request for BBB is a list of identical JSON representations of the rate object. The fields are described below:

Name	Description
“code”	The transaction code in the BitPay ledger. The code is associated with a transaction type, “txType”. ● 1000 = “sale” ● 1001 = “fee” ● 1002 = "payout" ● 1003 = "ACH/other" ● 1004 = "charity fee refund" ● 1005 = "deposit" ● 1006 = "exchange" ● 1007 = "exchange fee" ● 1008 = "plan charge" ● 1009 = "plan change credit" ● 1010 = "plan underuse credit" ● 1011 = "plan charge transfer"
“amount”	The amount of the credit or debit from the account. The amount is expressed in the currency unit specified in the request (see GET parameter “c”). The amount is a positive number for a ledger credit. The amount is a negative number for ledger debit.
“timestamp”	The date and time the ledger entry was made. Time format is “2014-01-01T19:01:01.123Z”.
“description”	The item description specified when the invoice was created. See invoice field “itemDesc”.
“orderId”	The merchant order identifier specified when the invoice was created. See invoice field “orderID”. If the invoice did not specify an “orderID” then this field w not be present in the result set.
“txType”	Identifies the type of transaction for the ledger entry. One of the following: ● “sale” ● “fee” ● "payout" ● "ACH/other" ● "charity fee refund" ● "deposit" ● "exchange" ● "exchange fee" ● "plan charge" ● "plan change credit" ● "plan underuse credit" ● "plan charge transfer"

“exRates”	This is a JSON block containing the exchange rate used in the transaction. ● currency code - the three letter currency code specifying the units of the associated exchange rate. ● exchange rate - the exchange rate price expressed in the associated currency code. Example: `{ "USD":915.0704604254529 }`
“buyerFields”	This is a JSON block containing the buyer details for the transaction. These values were provided when the invoice was created. All values are strings. Maximum string length is 100 characters. If no buyer information was specified when the invoice was created then this field will be present but the JSON block will be empty; e.g. `{}`. If only several of the buyer information fields were specified when the invoice was created then those that were not specified will not be present in the result set. ● “buyerName” ● “buyerAddress1” ● “buyerAddress2” ● “buyerCity” ● “buyerState” ● “buyerZip” ● “buyerEmail” ● “buyerPhone”
“invoiceId”	The unique id of the invoice assigned by bitpay.com
“sourceType”	An identifier that specifies the type of source used to initiate the ledger entry. One of the following: ● <blank> ● “invoice” ● “bitcoinTx”

Example JSON Response to Retrieving a Transaction Ledger


```
[
  {"code":1000,
   "amount":0.0195,
   "timestamp":"2013-12-02T16:16:29.612Z",
   "description":"2",
   "txType":"sale",
   "exRates":{"USD":1025},
   "buyerFields":{
     "buyerName":"BitPay Customer",
     "buyerAddress1":"3423 Piedmont Rd NE",
     "buyerAddress2":"Suite 516",
     "buyerCity":"Atlanta",
     "buyerState":"GA",
     "buyerZip":"30305",
     "buyerEmail":"customer@bitpay.com",
     "buyerPhone":"1-855-4-BITPAY"},
   "invoiceId":"C8a5bQeRTPDineVSDCw6KJ",
   "sourceType":"invoice",
   "orderId":"2"},
  {"code":1001,
   "amount":-0.0002,
   "timestamp":"2013-12-02T16:16:29.612Z",
   "txType":"fee",
   "exRates":{"USD":1025},
   "buyerFields":{
     "buyerName":"BitPay Customer",
     "buyerAddress1":"3423 Piedmont Rd NE",
     "buyerAddress2":"Suite 516","buyerCity":"Atlanta",
     "buyerState":"GA",
     "buyerZip":"30305",
     "buyerEmail":"customer@bitpay.com",
     "buyerPhone":"1-855-4-BITPAY"},
   "invoiceId":"C8a5bQeRTPDineVSDCw6KJ",
   "sourceType":"invoice",
   "orderId":"2"},
  {"code":1000,
   "amount":0.1093,
   "timestamp":"2014-01-06T19:01:09.522Z",
   "description":"Bill 1",
   "txType":"sale",
   "exRates":{"USD":915.0704604254529},
   "buyerFields":{},
   "invoiceId":"JHfkEPc212HThzB25ngz31",
   "sourceType":"invoice",
   "orderId":""},
```



```
 {"code":1001,
  "amount":-0.0011,
  "timestamp":"2014-01-06T19:01:09.522Z",
  "txType":"fee",
  "exRates":{"USD":915.0704604254529},
  "buyerFields":{},
  "invoiceId":"JHfkEPc212HThzB25ngz31",
  "sourceType":"invoice"}
]
```


Instant Payment Notification (IPN)

An IPN (Instant Payment Notifications) is an HTTP POST message sent from the BitPay server to the merchants eCommerce server. The primary purpose of an IPN is to reconcile a BitPay payment transaction with the customer order status on the merchants eCommerce server. The BitPay server send IPNs for each invoice according to the setting within the invoice. The BitPay invoice payload has an IPN parameter called "fullNotifications" with the following description:

```
default value: false
true:   Notifications will be sent on every status change.
false:  Notifications are only sent when an invoice is confirmed (according
        to the invoice "transactionSpeed" setting).
```

If "fullNotifications" is set true then the BitPay server will POST the complete invoice (all fields) to the invoice's notificationURL when ever the status of the invoice changes. If "fullNotifications" is set false then the BitPay server will only send the IPN when the invoice reaches the "confirmed" state.

The BitPay server attempts to send IPNs multiple times until the send is either successful or the BitPay server gives up. As of this writing the BitPay server attempts retries on the following schedule.

- 0:00 - 1 minute delay
- 1:00 - 4 minute delay
- 5:00 - 9 minute delay
- 14:00 - 16 minute delay
- 30:00 - 25 minute delay
- 55:00 - failed

The BitPay server consider a status code 200 response to be a successful send. The BitPay server does not follow redirects.

The invoice status that is sent with the IPN is the status at the time of IPN delivery (not the status at the time IPN was first attempted). IPNs need to be handled idiomatically (as they can be resent) and you should not assume you will receive an IPN for every invoice status (i.e., if you receive an invoice status of "complete" then you can assume that the invoice has been "paid" and can do whatever you would have done if you received a "paid" status).

In the event that IPNs are not being received or processed as expected, please check the following:

- Verify that your "notificationURL" for the invoice is "https://" (not "http://")
- Verify that your callback handler at the "notificationURL" is properly receiving POSTs. You can verify this by POSTing your own messages to the server from a tool like Chrome Postman.
- Verify that the POST data received is properly parsed and that the logic toward updating order status on your server is as expected.
- Verify that your server is not blocking POSTs from servers it may not recognize.

The BitPay server publishes a list of IP addresses for the origination of IPN POST messages (a whitelist). The content of the whitelist can change at any time and without notice. If your implementation references the BitPay IP whitelist then you should implement logic to consider that the list may have changed if you suspect that IPNs are no longer being received by your server. The BitPay IP whitelist is published at https://bitpay.com/ipAddressList.txt.

The format of the BitPay IP whitelist is as follows and may include any arbitrary number of IP address entries:

```
<ip-addr>|<ip-addr> ...

Example:
107.20.49.105|107.21.82.204|54.198.224.159
```

Integration Hints

Embedded Invoice (iframe) Post to Parent Window

When an invoice is presented in an iframe you have an option to receive invoice status updates in the parent window. This option is useful for updating the parent window presentation or redirecting the parent window to another URL after the invoice has been paid.

When the invoice iframe receives a status update from the BitPay server the new status is posted from the invoice iframe to the parent window via the Window.postMessage method and passing the following event data:

> {status: string}
> where 'status' can be any of the Invoice States according to the descriptions provided.

Your implementation should take into consideration the browser support for this method. See http://caniuse.com/#feat=x-doc-messaging for a list of browsers supporting Window.postMessage.

Following is an HTML code example illustrating a simple interaction between an invoice iframe and its parent document.

```
<html>
<head>
<title>Parent</title>
</head>
<body>
<p>Invoice status: <span id="s1"></span>
<iframe src="https://bitpay.com/invoice?id=UbAd3j1E7ivCe5i9t88obd"
style="width: 800px; height: 800px;"></iframe>
<script language="javascript">
window.addEventListener("message", function(event) {
    document.getElementById("s1").innerHTML=event.data.status;
}, false);
</script>
</body>
</html>
```

Constructing a Custom Invoice

There are instances when it is not possible to directly display the presentation of a BitPay server generated HTML invoice (either the full page or the iframe). This occurs when the system requesting an invoice is interacting with the customer via a device that does not directly present or render HTML content. Example use cases include:

- Proprietary gaming platform
- Custom embedded display device (gas pump, kiosk device)

Constructing an invoice typically requires that the implementation generate it's own QR (Quick Response) code, enabling custom interaction with a Bitcoin wallet, and/or making the bitcoin address visible to receive payment. Additional concerns include:

- Time limit - the client must ensure that presentation of the invoice does not exceed the maximum time allowed by BitPay to present payment. Currently the time limit is 15 minutes.
- Prices - the invoice should display both the bitcoin price as well as the local currency price. It is not recommended to display the bitcoin/local currency exchange rate as this could be confusing to the buyer.
- Bitcoin address - in some settings it may be convenient to display (or otherwise present) the bitcoin address for copy and paste. This is useful if the buyer must transfer the bitcoin address to their bitcoin wallet manually.
- Other information - details about the product or service being purchased, an order identifier, the merchant name, address, etc.

Invoice Status Updates

Use of the BitPay server provided invoice URL for presentation of the invoice includes the benefit of the client presented invoice receiving socket messages from the BitPay server when invoice status changes. For example, subsequent to paying an invoice the BitPay server will update the status of the invoice and send that status via a direct socket connection between the BitPay server and the invoice presenting web browser client. When presenting a custom invoice this socket connection is not possible. The only means for updating the presentation of the invoice (e.g., to show 'paid' status) is to receive the invoice IPN at your server and then to update your invoice presentation (e.g., using AJAX) at the client.

Generating a Custom QR Code Image

Important:

This method uses the Bitcoin Payment Protocol to obtain the backwards compatible "bitcoin:" URI string. Introspection of the content of this URI string is *strongly* discouraged. Any use of the content of the URI string (e.g., the bitcoin address) for sending a payment transaction circumvents the trust and security provided by the signed payment request produced via the Bitcoin Payment Protocol.

The QR code is a digitally encoded image that contains the Bitcoin payment URI. The BitPay server supports both the old style (BIP 72) Bitcoin protocol (for backwards compatibility) as well as the new style (BIP 73) Bitcoin Payment Protocol.

When choosing to generate your own QR code image you must make a choice between the two style of URI you wish to encode in the image. For maximum compatibility with wallets you should choose to encode the BIP 72 compatible URI. This URI scheme can be scanned successfully by all existing Bitcoin wallets. As more wallets provide support for the Bitcoin Payment Protocol you may choose to switch from using the BIP 72 compatible URI to the BIP 73 compatible URI. The density of a BIP 72 QR code is higher than that of a BIP 73 QR code and may result in increased scanning errors.

The BIP 73 URI to encode into a custom QR code image is simply the BitPay invoice URL; the 'uri' field obtained from the invoice payload after creating or GETting the invoice.

If you choose to encode the BIP 72 URI scheme in a custom QR code image then you will need to obtain the "bitcoin:" protocol URI string from the BitPay server using a feature outlined in BIP 73. Perform the following steps to retrieve the "bitcoin:" protocol string.

1. Create the BitPay invoice using this API specification.
2. Obtain the 'url' field of the invoice response payload.
3. Construct a GET request to the BitPay server setting the following HTTP header. Be sure not to Accept "application/bitcoin-paymentrequest". Send the GET to the the 'uri' obtained from the invoice payload.

```
Accept: text/uri-list
```

4. Read the "bitcoin:" URI string from the server response.
5. Encode the "bitcoin:" URI string using your QR image encoding library.

Sample Client Library

For convenience, a sample client library is provided to demonstrate how to interact with the BitPay.com JSON API. You can use this client library as is on your server, you can customize it, or you can use it as a guide for developing a client library in another language. The sample client library is written in JavaScript and is designed to run using nodejs. Nodejs can be downloaded form http://nodejs.org. The examples have been tested on version 0.8.9, but should work on later versions as well. The sample client library can be obtained from https://github.com/bitpay/nodejs-client/archive/master.zip. The zip file contains 3 utilities: createInvoice, getInvoice, invoiceListener. These files are executable and invoke the node runtime using typical Unix shebang notation. They can also be started by passing them as the first argument to the "node" runtime. The files themselves are JavaScript source code.

To use the utilities, modify the config.js file and copy and paste an API key from your merchant account into the apiKey setting. This will associate your API calls with your merchant account. Also, there is a sample SSL key and certificate file that is used by the invoiceListener to setup the HTTPS server that listens for incoming invoice notifications. While these example credentials will work fine, you may want to create your own unique SSL key and certificate.

To create an invoice, run the createInvoice utility and pass in an invoice description on stdin. A sample invoice description is provided in the file sampleInvoice.json. To create an invoice using this sample, run the following command:

```
$ ./createInvoice <sampleInvoice.json
```

The newly created invoice will be output on a single line in JSON format.

To get an invoice, run the getInvoice utility and pass the invoice id as the sole argument as follows:

```
$ ./getInvoice 5_TU2V-M0glicVcZuQkkkq9aiA7qP0MjxRkhdc1MRSY=
```

Just as before, the invoice will be output on a single line in JSON format.

To receive notifications of invoice status updates, use the invoiceListener utility. It takes a single parameter on the command line to specify the port number (or it can be specified in config.js) and listens for incoming notifications from BitPay.com. If you create an invoice with a notificationURL to your server and port, notifications of status changes on that invoice will be delivered to this utility. When a notification is received, the utility will print the JSON encoded invoice on stdout (one line per notification).

With these utilities, it is easy to craft a solution that can create a BitPay.com invoice and receive payment notifications.

Testing

Testing transaction processing is easily accomplished using the BitPay production server.

Merchant Account Setup

In order to test you must have an established BitPay merchant account. It is perfectly safe to test using a single production merchant account, however, this may not be suitable for some organizations due to security and ledger accounting visibility concerns. If you require a test environment to be completely isolated from your production environment then you should request a second BitPay merchant account.

Complete the following steps to setup your merchant account for testing:

1. Set your payout to 100% BTC - see https://bitpay.com/accounting
2. Set your bitcoin address to be the same address from which you send bitcoin while paying invoices
3. Generate an API key for your account - see https://bitpay.com/api-keys

Setting up your merchant account this way allows the bitcoins you send for payment to be circulated back to you the following day when merchant payouts are run.

Testing Considerations

Testing may involve not only the integration of the BitPay service and the payment of invoices but also how you handle common payment exceptions. See the BitPay Customer Support Guide for more information about payment exceptions.

You should consider testing the following kinds of transactions:

- A fully paid invoice, paid on time (within 15mins)
- A fully paid invoice, paid late (after 1 hour, after 24 hours)
- An under paid invoice (pay an amount less than the invoice requires)
- An overpaid invoice (pay an amount more than the invoice requires)
- An invoice with and without wallet fees
    - Try to force an invalid transaction using a low value invoice with no wallet fees
    - Invalid transactions are invoices that have been paid but which have zero confirmations within one hour of being received by the BitPay server

Troubleshooting

Community help and support can be found on the BitPay support forums at https://support.bitpay.com/forums.

Symptom	Possible Cause	Solution
Customer order status is not updated after payment.	The immediate payment notification (IPN) URL is specified incorrectly. The BitPay API supports sending IPN POSTs to only secure socket layer (SSL) URLs. Additionally, the IPN URL must have a valid SSL certificate.	1. Verify that your "notificationURL" for the invoice is "https://" (not "http://") 2. Ensure a valid SSL certificate is installed on your server. 3. Verify that your callback handler at the "notificationURL" is properly receiving POSTs. You can verify this by POSTing your own messages to the server from a tool like Chrome Postman. 4. Verify that the POST data received is properly parsed and that the logic toward updating order status on the merchants eCommerce server is as expected. 5. Verify that the merchants eCommerce server is not blocking POSTs from servers it may not recognize.

Revision History

0.1	September 2011	Original Release
0.2	December 2011	Updated SSL info
0.3	September 2012	Changed Authentication from SSL fingerprint to API token method
0.3.1	December 2013	Added Business Integration Scenarios
0.3.2	January 2014	Added explanation of transaction speed and fulfillment Added BBB API section Added Troubleshooting section Added Testing section Added new implementation scenario
0.4	January 2014	Added Transaction Ledger API section Moved Business Integration Scenarios to the new Quick Start Guide
0.5	March 2014	Added Bitcoin Payment Protocol Added Instant Payment Notification (IPN) Added Constructing a Custom Invoice